Developer Guide

Integration Development Guide 2024 R2

Acumatica
The Cloud ERP

Contents

Copyright

3075 112th Avenue NE, Suite 200, Bellevue, WA 98004, USA

Restricted Rights

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

Disclaimer

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

Trademarks

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

Software Version: 2024 R2

Last Updated: 01/15/2025

Integration Development Guide

In this guide, you can find information about how to develop client applications that work with Acumatica ERP through the web services.

Configuring the REST API

Acumatica ERP provides web services for integration with external systems. Through the web services of Acumatica ERP, external systems can get data records from Acumatica ERP, process these records, and save new or updated records to Acumatica ERP.

To access these web services, you can use the contract-based representational state transfer (REST) API and the screen-based SOAP API. In this chapter, you will find the main concepts that are related to the contract-based REST API.

Contract-Based REST API

The contract-based REST API operates with business logic objects that do not depend on Acumatica ERP forms or their properties and methods. (In this context, *contract-based* means based on the object model that the REST API provides.) Each REST API contract is fixed and does not change based on system customization, localization, or any other changes made to Acumatica ERP.

For example, suppose that the contract of the web service contains the definition of the *CustomerID* field, which accesses the **Customer ID** element on the *Customers* (AR303000) form. If you have changed the name of the **Customer ID** element to **Customer Identifier** in a customization project, the contract of the web service remains fully functional and does not require update; also, your application requires no further modifications. You can access the **Customer Identifier** element on the form through the same `CustomerID` field.

To use the REST API in your application, first of all, you should decide which endpoint to use. You can find more information on the endpoints and their contracts in *Endpoints and Contracts*.

After that, you can use the REST API in your application. For examples of the REST API requests, see *REST API Examples*.

Related Links

- *Endpoints and Contracts*
- *API Entities, Fields, and Actions*

Endpoints and Contracts

You access the contract-based REST API through endpoints that are configured on the *Web Service Endpoints* (SM207060) form.

Endpoints and Contracts

An *endpoint* is an entry point to the Acumatica ERP web services. For each endpoint that the REST API provides, a *contract* of the endpoint defines the entities, along with their actions and fields, that are available through the endpoint and the methods that you can use to work with these entities.

The endpoint is identified by the URL that you use to access the web services API. You can see the name and version of an endpoint in its URL. For example, the `http://localhost/AcumaticaDB/entity/Default/24.200.001` endpoint has the *24.200.001* version and the *Default* name. The version of an endpoint defines the list of entities, along with the actions and fields that you can work with through this endpoint.

The contract of an endpoint is identified by the contract version. The version of a contract defines the list of methods for working with the entities that you can use when working with Acumatica ERP through the endpoint with this version of the contract. For details about the contract version, see *Contract Version*.

System and Custom Endpoints

You can use two types of endpoints to access the web services:

- **System endpoint**: The system endpoints are precofigured in the system and have the *Default* name. Each of these endpoints has a predefined contract, which includes the API that is preconfigured in the system. You cannot change the contract of a system endpoint.

  If the API that is available in the contract of a system endpoint is sufficient for the requirements of your application, you should use the system endpoint for accessing Acumatica ERP web services. You can use the same system endpoint in future versions of Acumatica ERP. For example, if you use the system endpoint with Version 24.200.001 and Contract Version 4 to access Acumatica ERP 2024 R2, you can use the same endpoint to access future versions of Acumatica ERP.

  > ⚠ Acumatica ERP can include endpoints preconfigured in the system that have names other than *Default*. The system uses these endpoints internally. We recommend that you not use these endpoints.

- **Custom endpoint**: By default, there are no custom endpoints in the system. If the API provided by the system endpoint is not sufficient for the requirements of your application, you can create a custom endpoint. You can configure the contract of a custom endpoint by adding the needed elements of the API to the contract.

  If you need to use the same custom endpoint in future versions of Acumatica ERP, you should maintain it in future versions.

> ⚠ Obsolete system endpoints may be removed in some version of Acumatica ERP.

The following diagram provides an example of multiple endpoints configured in the system. The diagram shows two system endpoints with Contract Version 4 and two custom endpoints with the names *EastEndpoint* and *WestEndpoint*.

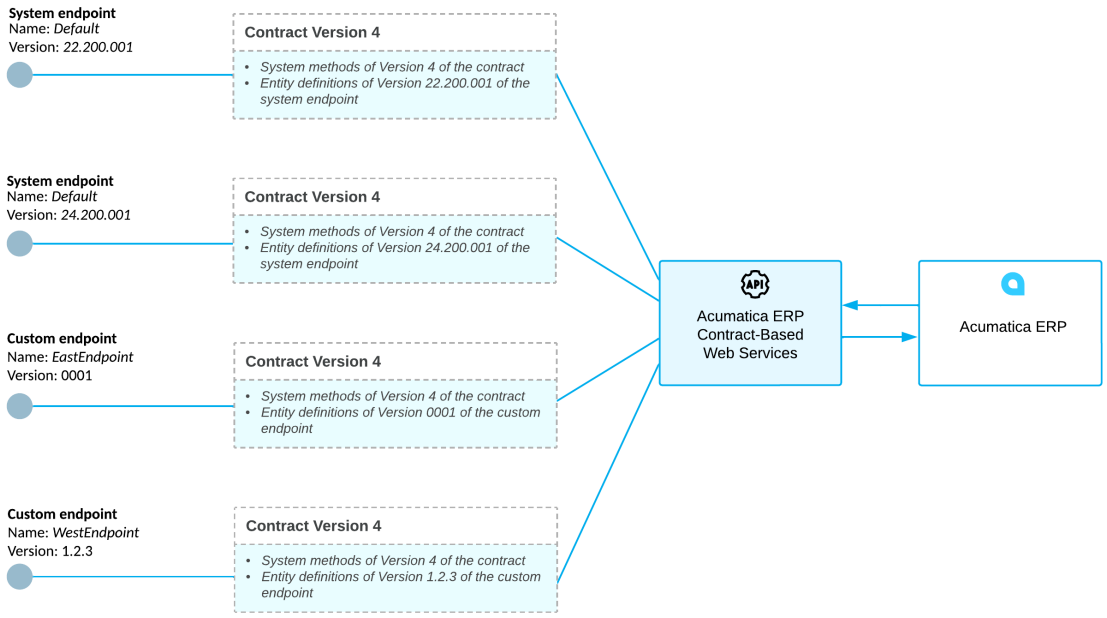

Figure: Contract-based web services

API Entities, Fields, and Actions

The contract of an endpoint defines the following elements of the contract-based REST API: entities, fields, and actions.

Entities

An entity corresponds to a business logic object that you are going to work with. For example, the contract of a system endpoint includes the `Warehouse` entity, which represents a warehouse and holds the data related to the warehouse. This entity is associated with the *Warehouses* (IN204000) form.

For a custom endpoint, if you are going to use an entity to transfer data to or from Acumatica ERP, you should associate this entity with a particular Acumatica ERP form. For example, you can create a `TrialBalance` entity, which represents a trial balance. This entity is associated with the *Trial Balance* (GL303010) form.

Fields

The fields of an entity correspond to the fields of a business logic object. For example, the `Warehouse` entity that is available through a system endpoint has the `Description` and `WarehouseID` fields, among others. In the contract, these fields are mapped to the **Description** and the **Warehouse ID** elements of the Summary area of the *Warehouses* (IN204000) form.

For a custom endpoint, if you need to connect the field with a particular element on an Acumatica ERP form, you should map the field to this element. For example, if you have created the `TrialBalance` entity, which designates a trial balance, you can add the `ImportNbr` field to the entity and connect this field with the **Import Number** element of the Summary area of the *Trial Balance* (GL303010) form.

Actions

The actions of an entity correspond to the actions that can be applied to a business logic object. For example, the `TransferOrder` entity, which is available through a system endpoint, has the `ReleaseTransferOrder` action. This action corresponds to the **Release** button on the form toolbar of the *Transfers* (IN304000) form.

For a custom endpoint, if you need to use an Acumatica ERP action, you should add this action to the contract of the custom endpoint with the needed parameters. For example, suppose that you want to add an action that changes the customer ID of an existing customer. You can add the `ChangeID` action and map it to the **Change ID** command, which is available on the *Customers* (AR303000) form. The new action should have one parameter, which specifies the new ID of a customer, because the **Change ID** command requires the new ID.

Types of Entities

When you add a new entity to a contract, you should specify the type of the entity, which can be one of the following:

- *Top-Level*: Entities of this type are the main entities of the contract. A top-level entity usually corresponds to an Acumatica ERP form. For example, the `Warehouse` entity of the contract of a system endpoint is a top-level entity that corresponds to the *Warehouses* (IN204000) form.
- *Detail*: Detail entities correspond to the detail lines of a master-detail form. A detail entity exists only as a part of a top-level entity. For example, the `SalesOrder` top-level entity of the contract of a system endpoint contains the `SalesOrderDetail` detail entity. This detail entity corresponds to a detail line on the **Details** tab of the *Sales Orders* (SO301000) form, as shown in the following screenshot.

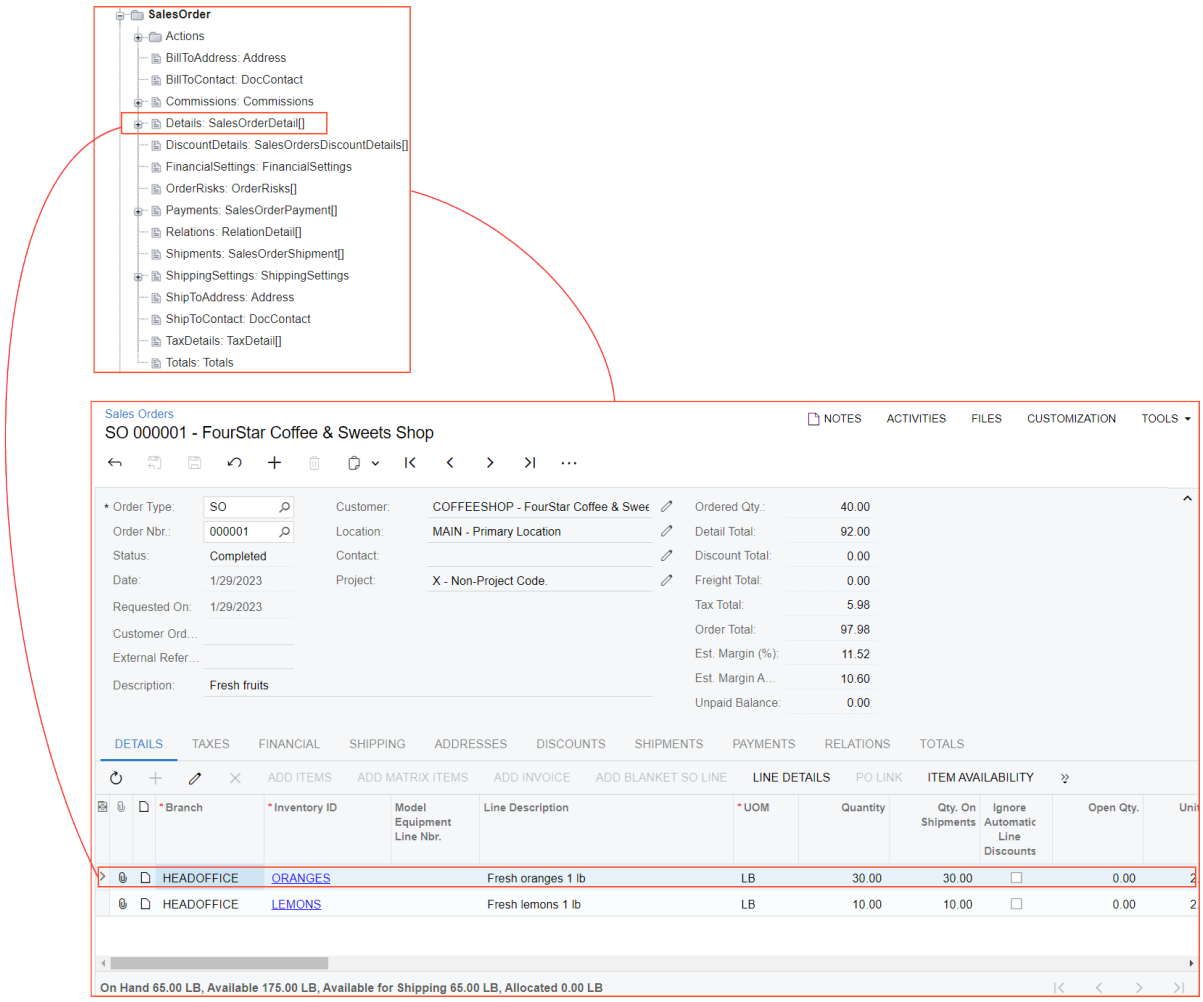

Figure: Detail entity

- *Linked*: Linked entities are supplementary entities of a contract. A linked entity usually corresponds to a part of an Acumatica ERP form and is related to one top-level entity of the contract or multiple top-level entities. For example, the `Contact` top-level entity of the contract of a system endpoint contains the `Address` linked entity . This linked entity corresponds to the **Address** section on the **Details** tab of the *Contacts* (CR302000) form, as shown in the following screenshot.

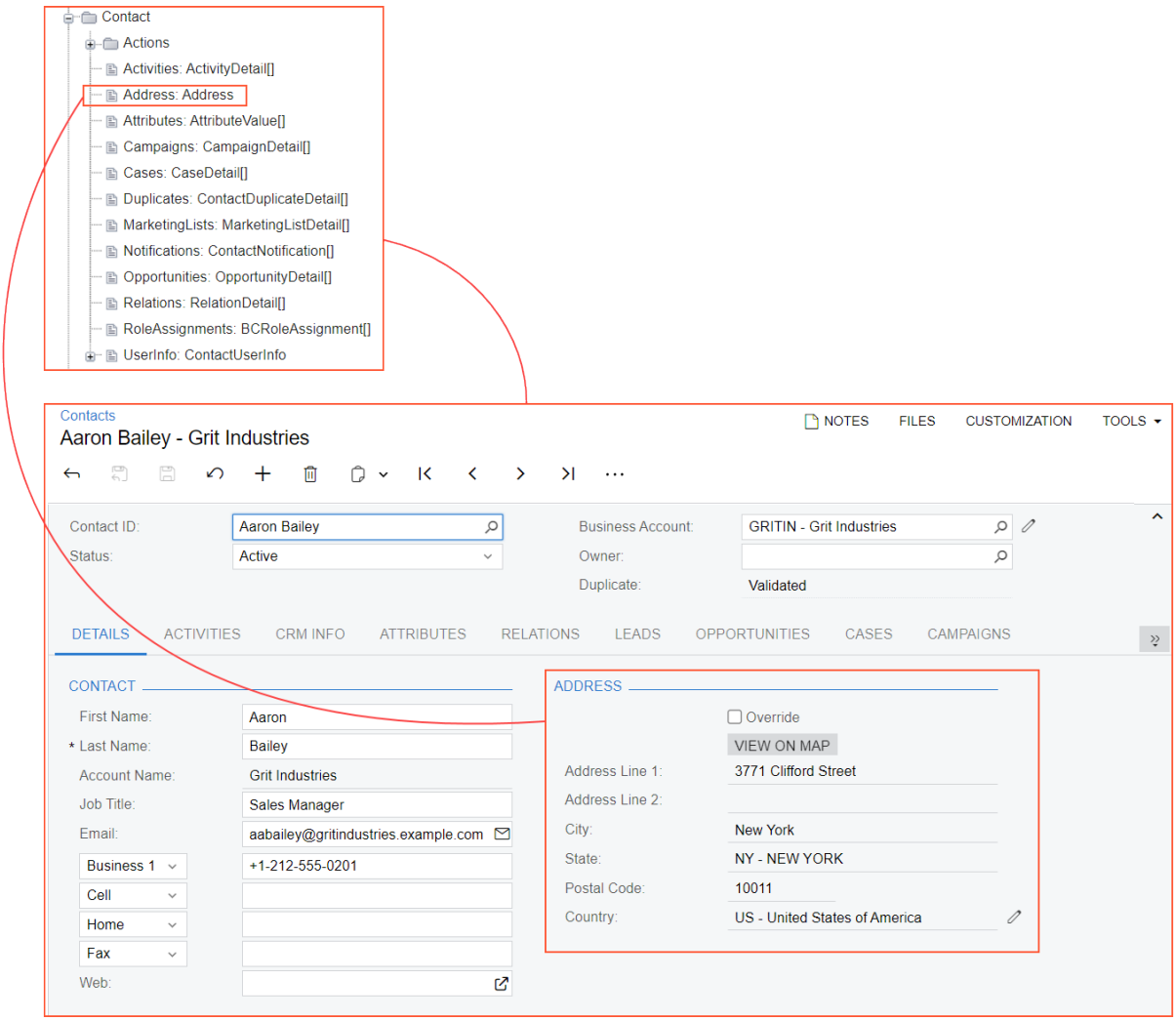

Figure: Linked entity

- *Report*: An entity of this type corresponds to an Acumatica ERP report. For example, in a custom endpoint, you can create the `CashAccountSummary` report entity, which corresponds to the *Cash Account Summary* (CA633000) report.

Custom Fields

You can work with the values of the custom fields that are not included in the entity definition.

Custom fields can correspond to the following elements:

- The predefined elements on an Acumatica ERP form that are not included in the entity definition
- The elements that were added to the Acumatica ERP form in a customization project
- The user-defined fields

To work with the needed custom field, you need to know the name of the data view that contains the corresponding custom element and the name of the field, which are described in detail below.

Field Name and View Name

A field name is the internal name of a particular element of an Acumatica ERP form. A view name is the name of the data view to which a particular element belongs. For example, the **Posting Class** element on the **General** tab of the *Stock Items* (IN202500) form has the `PostClassID` field name and belong to the `ItemSettings` data view.

To find out the field name and view name, on the title bar of the form, you click **Customization > Inspect Element** and click the needed element on the form. In the **Element Properties** dialog box, which opens, you find the field name in the **Data Field** element and the view name in the **View Name** element, as shown in the following screenshot.

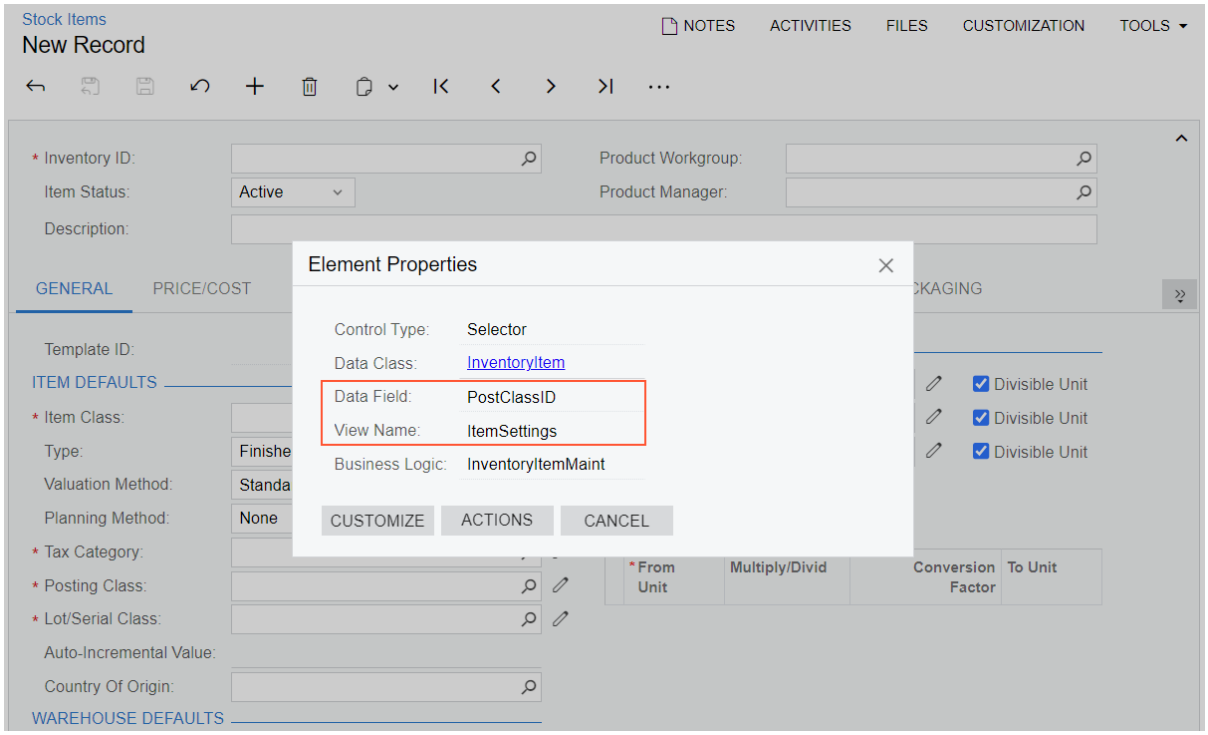

Figure: Field name and view name

In the contract-based REST API, you can also find out the field name and the view name through the special URL. For details on the URL and the HTTP method, see *Retrieve the Schema of Custom Fields*.

Field Name and View Name of a User-Defined Field

For any user-defined field, the field name is `Attribute<AttributeID>`, where you replace `<AttributeID>` with the ID of the attribute that corresponds to the user-defined field. For details on how you can find out the view name, see *Retrieve the Schema of Custom Fields*.

For example, suppose that on the *Sales Orders* (SO301000) form, you have added a user-defined field for the *OPERATSYST* attribute. You work with this user-defined field by using the `Document` view name and the `AttributeOPERATSYST` field name.

Use of Custom Fields

For details on retrieving the values of custom fields by using the contract-based REST API, see *$custom Parameter* and *Retrieve a Record with Custom Fields*. For details on specifying the values of custom fields, see *Representation of a Record in JSON Format* and *Create a Record with Custom Fields*.

Custom Endpoints and Endpoint Extensions

If the API provided by the system endpoint of Acumatica ERP is not sufficient for the requirements of your application, you can create a custom endpoint from scratch or by extending an existing endpoint.

An Extension of an Existing Endpoint

If you are creating an endpoint as an extension of an existing endpoint, for the API elements that were inherited from the base endpoint, you cannot edit the names and types of the entities and fields, and the names, types, and parameters of the actions. In the contract of the new endpoint, you can add new top-level entities, new fields or entities to any entity, and new actions. Then you can use both the API that you added to the contract of the endpoint and the API of the base endpoint in your application. For information on how to extend an existing endpoint, see *To Extend an Existing Endpoint*.

The new endpoint that was created as an extension of an existing endpoint has the version of the contract of the base endpoint; that is, the API methods for working with entities are the same for the base endpoint and the new endpoint.

An Endpoint Created from Scratch

If you are creating an endpoint from scratch, you should add the needed elements of the API to the contract. Then you can use these API elements in your application. For information on how to create an endpoint from scratch, see *To Create a Custom Endpoint*. The new endpoint that is created from scratch always has the latest version of the contract.

Related Links

- *Contract Version*

Naming Rules for Endpoints

When you create a custom endpoint on the *Web Service Endpoints* (SM207060) form (either from scratch or by extending a system endpoint), for the names of the entities, fields, actions, and action parameters of the endpoint, and the endpoint name and version, you should make sure to adhere to the following rules:

- The name of the endpoint can contain only English letters, digits, underscores, and periods, and cannot start with a digit.
- The version of the endpoint can contain only English letters, digits, underscores, and periods.
- The name of the entity, field, action, or action parameter can contain only English letters, digits, and underscores, and cannot start with a digit.
- The name of the field cannot match any of the following reserved names:
  - `Action`
  - `CustomFields`
  - `Delete`
  - `Entity`
  - `ID`
  - `Note`
  - `ReturnBehavior`
  - `RowNumber`

- The name of the field must be unique among the names of the fields of the entity.
- The name of the field that should be used to access a nested entity cannot be `Files` or `Translations`.
- The name of the parameter must be unique among the names of the parameters of the action.
- The name of the entity or action must be unique among the names of the entities and actions of the endpoint.

The system checks whether the names used in the endpoint satisfy these rules each time you enter the name of a new entity, field, action, or action parameter. You can also validate the endpoint manually, as described in *To Validate an Endpoint*.

Related Links

- *To Validate an Endpoint*

Contract Version

Acumatica ERP 2024 R2 supports one version of system contract, which is Contract Version 4. In this topic, you can learn the main characteristics of this contract version.

> ⚠ Contract Version 1 is not supported starting from Acumatica ERP 2018 R2. Contract Versions 2 and 3 are not supported starting from Acumatica ERP 2023 R2.

Characteristics of Contract Version 4

The main characteristics of Contract Version 4 are the following:

- The REST API is supported for the endpoints with this contract version.
- The SOAP API is not supported for the endpoints with this contract version.
- You can specify particular fields of the entity to be returned from the system.
- By default, the system does not return all fields of the entity (including fields of the linked and detail entities defined within the entity).
- Through the endpoint, you can work with the elements that were added to the Acumatica ERP form in a customization project.
- Through the endpoint, you can work with the predefined elements on an Acumatica ERP form that are not included in the entity definition.
- When optimization for speed of the retrieval of the list of records fails, the system returns an error.
- Custom endpoints created from scratch have this contract version.
- The following system endpoints that have this contract version are included in Acumatica ERP 2024 R2: *Default/20.200.001*, *Default/22.200.001*, *Default/23.200.001*, and *Default/24.200.001*

Related Links

- *Endpoints and Contracts*

Comparison of System Endpoints

Acumatica ERP 2024 R2 supports three system endpoints. All these endpoints have Contract Version 4. In this topic, you can learn about the differences between these endpoints.

Changes to the Entities, Fields, and Actions of the Default/24.200.001 Endpoint as Compared to the Default/23.200.001 Endpoint

The following tables contain the new, modified, or removed elements of the *Default/24.200.001* endpoint as compared to the *Default/23.200.001* endpoint.

Table: New Entities

Entity	Related Form Name and ID
AmazonStore	*Amazon Stores* (BC201020)
CashTransaction	*Cash Transactions* (CA304000)
CashTransactionDetail	*Cash Transactions* (CA304000)
SalesInvoiceAddress	*Invoices* (SO303000)
SalesInvoiceDocContact	*Invoices* (SO303000)

Table: New Fields and Actions

Field or Action Name	Related Form Name and ID
AttributeValue.IsActive	*Account Groups* (PM201000), *Business Accounts* (CR303000), *Cases* (CR306000), *Change Orders* (PM308000), *Contacts* (CR302000), *Customers* (AR303000), *Employees* (EP203000), *Leads* (CR301000), *Non-Stock Items* (IN202000), *Opportunities* (CR304000), *Project Tasks* (PM302000), *Project Templates* (PM208000), *Project Template Tasks* (PM208010), *Stock Items* (IN202500), *Template Items* (IN203000), *Vendors* (AP303000)
BillDetail.LCLineNbr	*Bills and Adjustments* (AP301000)
BillDetail.LCNbr	*Bills and Adjustments* (AP301000)
BillDetail.LCType	*Bills and Adjustments* (AP301000)
InventoryReceiptDetail.ReasonCode	*Receipts* (IN301000)
NonStockItem.IsAKit	*Non-Stock Items* (IN202000)
NonStockItemVendorDetail.RecordID	*Non-Stock Items* (IN202000)
SalesInvoice.BillToAddress	*Invoices* (SO303000)
SalesInvoice.BillToAddressOverride	*Invoices* (SO303000)

Field or Action Name	Related Form Name and ID
SalesInvoice.BillToContact	*Invoices* (SO303000)
SalesInvoice.BillToContactOverride	*Invoices* (SO303000)
SalesInvoice.CreatedDate	*Invoices* (SO303000)
SalesInvoice.ExternalRef	*Invoices* (SO303000)
SalesInvoice.LastModifiedDate	*Invoices* (SO303000)
SalesInvoice.ShipToAddress	*Invoices* (SO303000)
SalesInvoice.ShipToAddressOverride	*Invoices* (SO303000)
SalesInvoice.ShipToContact	*Invoices* (SO303000)
SalesInvoice.ShipToContactOverride	*Invoices* (SO303000)
SalesInvoice.TaxCalcMode	*Invoices* (SO303000)
SalesInvoiceDetail.Account	*Invoices* (SO303000)
SalesInvoiceDetail.ExternalRef	*Invoices* (SO303000)
SalesInvoiceDetail.ManualPrice	*Invoices* (SO303000)
SalesInvoiceDetail.NoteID	*Invoices* (SO303000)
SalesInvoiceDetail.SubAccount	*Invoices* (SO303000)
SalesOrder.RecalculatePricesDiscounts	*Sales Orders* (SO301000)
Shipment.UnlimitedPackages	*Shipments* (SO3020000)
StockItem.LastModifiedDateTime	*Stock Items* (IN202500)
TemplateItems.LastModifiedDateTime	*Template Items* (IN203000)
TemplateItemVendorDetail.RecordID	*Template Items* (IN203000)
TransferOrderDetailAllocation.ExpirationDate	*Transfers* (IN304000)

Changes to the Entities, Fields, and Actions of the Default/23.200.001 Endpoint as Compared to the Default/22.200.001 Endpoint

The following tables contain the new, modified, or removed elements of the *Default/23.200.001* endpoint as compared to the *Default/22.200.001* endpoint.

Table: New Entities

Entity	Related Form Name and ID
BCRoleAssignment	*Contacts* (CR302000)
OrderRisks	*Sales Orders* (SO301000)
PaymentCharge	*Payments and Applications* (AR302000)
ProjectAddress	*Projects* (PM301000)
ProjectRetainage	*Projects* (PM301000)
Subcontract	*Subcontracts* (SC301000)
SubcontractDetail	*Subcontracts* (SC301000)
SubcontractTaxDetail	*Subcontracts* (SC301000)
SubcontractVendorAddressInfo	*Subcontracts* (SC301000)
SubcontractVendorContactInfo	*Subcontracts* (SC301000)

Table: New Fields and Actions

Field or Action Name	Related Form Name and ID
AccountGroup.LastModifiedDateTime	*Account Groups* (PM201000)
Appointment.LastModifiedDateTime	*Appointments* (FS300200)
Bill.IsTaxValid	*Bills and Adjustments* (AP301000)
Bill.ReleaseRetainage	*Bills and Adjustments* (AP301000)
BusinessAccount.CreateContactFromBusinessAccount	*Business Accounts* (CR303000)
BusinessAccount.Relations.DocumentDate	*Business Accounts* (CR303000)
Case.Relations.DocumentDate	*Cases* (CR306000)
Contact.CreateAccountFromContact	*Contacts* (CR302000)
Contact.Relations.DocumentDate	*Contacts* (CR302000)
Contact.RoleAssignments	*Contacts* (CR302000)
Customer.Contacts.LastModifiedDateTime	*Customers* (AR303000)
Customer.CreateContactFromCustomer	*Customers* (AR303000)
Customer.CustomerKind	*Customers* (AR303000)
Customer.Email	*Customers* (AR303000)

Field or Action Name	Related Form Name and ID
Customer.PrimaryContactID	*Customers* (AR303000)
CustomerLocation.Default	*Customer Locations* (AR303020)
CustomerLocation.Status	*Customer Locations* (AR303020)
CustomerLocation.RoleAssignments	*Customer Locations* (AR303020)
InventoryAdjustment.LastModifiedDateTime	*Adjustments* (IN303000)
InventoryIssue.LastModifiedDateTime	*Issues* (IN302000)
KitSpecification.LastModifiedDateTime	*Kit Specifications* (IN209500)
Lead.Relations.DocumentDate	*Leads* (CR301000)
Opportunity.Products.SkipLineDiscounts	*Opportunities* (CR304000)
Opportunity.Relations.DocumentDate	*Opportunities* (CR304000)
Payment.Charges	*Payments and Applications* (AR302000)
Payment.IsCCPayment	*Payments and Applications* (AR302000)
Project.BillingAndAllocationSettings.BillingCurrency	*Projects* (PM301000)
Project.ProjectAddress	*Projects* (PM301000)
Project.ProjectProperties.CostBudgetLevel	*Projects* (PM301000)
Project.ProjectProperties.CostTaxZone	*Projects* (PM301000)
Project.ProjectProperties.InventoryTrackingMode	*Projects* (PM301000)
Project.ProjectProperties.ProjectCurrency	*Projects* (PM301000)
Project.ProjectProperties.RateType	*Projects* (PM301000)
Project.ProjectProperties.RevenueTaxZone	*Projects* (PM301000)
Project.ProjectProperties.TimeActivityApprover	*Projects* (PM301000)
Project.Retainage	*Projects* (PM301000)
PurchaseReceipt.LastModifiedDateTime	*Purchase Receipts* (PO302000)
SalesInvoice.LastModifiedDateTime	*Invoices* (SO303000)
SalesOrder.Branch	*Sales Orders* (SO301000)

Field or Action Name	Related Form Name and ID
SalesOrder.MaxRiskScore	*Sales Orders* (SO301000)
SalesOrder.OrderRisks	*Sales Orders* (SO301000)
SalesOrder.Relations	*Sales Orders* (SO301000)
ServiceOrder.LastModifiedDateTime	*Service Orders* (FS300100)
Vendor.CreateContactFromVendor	*Vendors* (AP303000)
Warehouse.LastModifiedDateTime	*Warehouses* (IN204000)
Warehouse.NonStockPickingLocationID	*Warehouses* (IN204000)
Warehouse.UseItemDefaultLocationForPicking	*Warehouses* (IN204000)

Table: Modified Fields and Actions

Field or Action Name	Modification	Related Form Name and ID
Lead.ConvertLeadToBAccount	The list of parameters has been changed.	*Leads* (CR301000)
Lead.ConvertLeadToContact	The list of parameters has been changed.	*Leads* (CR301000)
Lead.ConvertLeadToOpportunity	The list of parameters has been changed.	*Leads* (CR301000)
Opportunity.CreateContactFromOpportunity	The list of parameters has been changed.	*Opportunities* (CR304000)
Opportunity.CreateAccountFromOpportunity	The list of parameters has been changed.	*Opportunities* (CR304000)
Project.ProjectManager	The mapped field has been changed.	*Projects* (PM301000)

Table: Renamed Fields and Actions

Old Field or Action	New Field or Action Name	Related Form Name and ID
StorageDetailsInquiry.StorageDetail.SiteAvailable	QtyAvailable	Storage Details (IN408055)
StorageDetailsInquiry.StorageDetail.SiteAvailableforIssue	QtyAvailableforIssue	Storage Details (IN408055)

Old Field or Action	New Field or Action Name	Related Form Name and ID
StorageDetailsInquiry.StorageDetail.SiteAvailableforShipping	QtyHardAvailable	Storage Details (IN408055)
StorageDetailsInquiry.StorageDetail.SiteID	WarehouseID	Storage Details (IN408055)
StorageDetailsInquiry.StorageDetail.SiteLastModifiedDate	LastModifiedDateofWarehouseQty	Storage Details (IN408055)
StorageDetailsInquiry.StorageDetail.SiteOnHand	QtyOnHand	Storage Details (IN408055)
StorageDetailsByLocationInquiry.StorageDetailByLocation.LocationAvailable	QtyAvailableinLocation	*Storage Details by Item Warehouse Location* (IN408055)
StorageDetailsByLocationInquiry.StorageDetailByLocation.LocationAvailableforIssue	QtyAvailableforIssueinLocation	*Storage Details by Item Warehouse Location* (IN408055)
StorageDetailsByLocationInquiry.StorageDetailByLocation.LocationAvailableforShipping	QtyAvailableforShippinginLocation	*Storage Details by Item Warehouse Location* (IN408055)
StorageDetailsByLocationInquiry.StorageDetailByLocation.LocationLastModifiedDate	LastModifiedDateofLocationQty	*Storage Details by Item Warehouse Location* (IN408055)
StorageDetailsByLocationInquiry.StorageDetailByLocation.LocationOnHand	QtyOnHandinLocation	*Storage Details by Item Warehouse Location* (IN408055)
StorageDetailsByLocationInquiry.StorageDetailByLocation.SiteAvailable	QtyAvailableinWarehouse	*Storage Details by Item Warehouse Location* (IN408055)
StorageDetailsByLocationInquiry.StorageDetailByLocation.SiteAvailableforIssue	QtyAvailableforIssueinWarehouse	*Storage Details by Item Warehouse Location* (IN408055)

Old Field or Action	New Field or Action Name	Related Form Name and ID
StorageDetailsBy-LocationInquiry.S-torageDetailByLoca-tion.SiteAvailable-forShipping	QtyAvailableforShipping-inWarehouse	*Storage Details by Item Warehouse Location* (IN408055)
StorageDetailsByLoca-tionInquiry.StorageDe-tailByLocation.SiteID	WarehouseID	*Storage Details by Item Warehouse Location* (IN408055)
StorageDetailsByLoca-tionInquiry.StorageDe-tailByLocation.SiteLast-ModifiedDate	LastModifiedDateofWare-houseQty	*Storage Details by Item Warehouse Location* (IN408055)
StorageDetailsByLoca-tionInquiry.StorageDe-tailByLocation.SiteOn-Hand	QtyOnHandinWarehouse	*Storage Details by Item Warehouse Location* (IN408055)

Table: Removed Entities

Entity Name	Related Form Name and ID
SubItemStockItem	*Stock Items* (IN202500)
TrialBalance	*Trial Balance* (GL303010)
TrialBalanceDetail	*Trial Balance* (GL303010)

Table: Removed Fields

Field Name	Related Form Name and ID	Comment
Project.BillingAndAllocationSet-tings.Retainage	*Projects* (PM301000)	Use Project.Re-tainage instead.
StockItem.SubItems	*Stock Items* (IN202500)	

Changes to the Entities, Fields, and Actions of the Default/22.200.001 Endpoint as Compared to the Default/20.200.001 Endpoint

The following tables contain the new, modified, or removed elements of the *Default/22.200.001* endpoint as compared to the *Default/20.200.001* endpoint.

Table: New Entities

Entity	Related Form Name and ID
ACAInfoDetail	*Deduction and Benefit Codes* (PR101060)

Entity	Related Form Name and ID
ACAInformation	*Deduction and Benefit Codes* (PR101060)
ApplicableWage	*Deduction and Benefit Codes* (PR101060)
BatchDeductionOrBenefitDetail	*Payroll Batches* (PR301000)
BatchEarningDetail	*Payroll Batches* (PR301000)
BatchOvertimeRules	*Payroll Batches* (PR301000)
BatchOvertimeRulesDetail	*Payroll Batches* (PR301000)
BenefitIncreasingApplWage	*Deduction and Benefit Codes* (PR101060)
BenefitIncreasingApplWageDetail	*Deduction and Benefit Codes* (PR101060)
BigCommerceStores	*BigCommerce Stores* (BC201000)
CalendarSettings	*Work Calendar* (CS209000)
CompensationDetail	*Employee Payroll Settings* (PR203000)
CountryDetail	*Sales Territories* (CS204100)
DeductionBenefitCode	*Deduction and Benefit Codes* (PR101060)
DeductionBenefitWCCCode	*Deduction and Benefit Codes* (PR101060)
DeductionDecreasingApplWage	*Deduction and Benefit Codes* (PR101060)
DeductionDecreasingApplWageDetail	*Deduction and Benefit Codes* (PR101060)
DeductionOrBenefitCodeGLAccounts	*Deduction and Benefit Codes* (PR101060)
DeductionOrBenefitTaxDetailCA	*Deduction and Benefit Codes* (PR101060)
DeductionOrBenefitTaxDetailUS	*Deduction and Benefit Codes* (PR101060)
DeductionsAndBenefits	*Employee Payroll Settings* (PR203000)
DirectDepositDetail	*Employee Payroll Settings* (PR203000)
EarningCodeGLAccounts	*Earning Type Codes* (PR102000)
EarningCodeProjectSettings	*Earning Type Codes* (PR102000)
EarningCodeTaxDetailCA	*Earning Type Codes* (PR102000)
EarningCodeTaxDetailUS	*Earning Type Codes* (PR102000)
EarningIncreasingApplWage	*Deduction and Benefit Codes* (PR101060)
EarningIncreasingApplWageDetail	*Deduction and Benefit Codes* (PR101060)

Entity	Related Form Name and ID
EarningTypeCode	*Earning Type Codes* (PR102000)
EmployeeClassPTOBankDefault	*Employee Payroll Class* (PR202000)
EmployeeClassWorkLocation	*Employee Payroll Class* (PR202000)
EmployeeDeduction	*Deduction and Benefit Codes* (PR101060)
EmployeeDeductionOrBenefitDetail	*Employee Payroll Settings* (PR203000)
EmployeeGeneralInfo	*Employee Payroll Settings* (PR203000)
EmployeeGLAccounts	*Employee Payroll Settings* (PR203000)
EmployeePaidTimeOff	*Employee Payroll Settings* (PR203000)
EmployeePaidTimeOffDetail	*Employee Payroll Settings* (PR203000)
EmployeePaycheckEarningDetail	*Payroll Batches* (PR301000)
EmployeePaycheckEarnings	*Payroll Batches* (PR301000)
EmployeePaycheckSummary	*Payroll Batches* (PR301000)
EmployeePayrollClass	*Employee Payroll Class* (PR202000)
EmployeePayrollClassDefaults	*Employee Payroll Class* (PR202000)
EmployeePayrollSettings	*Employee Payroll Settings* (PR203000)
EmployeeTaxDetail	*Employee Payroll Settings* (PR203000)
EmployeeWorkLocationDetail	*Employee Payroll Settings* (PR203000)
EmployeeWorkLocations	*Employee Payroll Settings* (PR203000)
EmployerContribution	*Deduction and Benefit Codes* (PR101060)
EmployerTaxesIncreasingApplWage	*Deduction and Benefit Codes* (PR101060)
EmployerTaxesIncreasingApplWageDetail	*Deduction and Benefit Codes* (PR101060)
EmploymentDates	*Employee Payroll Settings* (PR203000)
EmploymentRecord	*Employee Payroll Settings* (PR203000)
GarnishmentDetails	*Employee Payroll Settings* (PR203000)
InventoryFileUrls	*Template Items* (IN203000)
InventoryIssue	*Issues* (IN302000)
InventoryIssueDetail	*Issues* (IN302000)

Entity	Related Form Name and ID
InventoryIssueDetailAllocation	*Issues* (IN302000)
InventoryQuantityAvailable	Available Quantity by Inventory Item (GI640590)
InventoryQuantityAvailableDetail	Available Quantity by Inventory Item (GI640590)
LaborRate	*Labor Rates* (PM209900)
MatrixItems	*Template Items* (IN203000)
PaymentPeriod	*Pay Periods* (PR201000)
PayGroup	*Pay Groups* (PR205000)
PayPeriod	*Pay Periods* (PR201000)
PayrollBatch	*Payroll Batches* (PR301000)
PayrollUnionLocal	*Union Locals* (PR209700)
PayrollWCCCode	*Workers' Compensation Codes* (PR209800)
PTOBank	*PTO Banks* (PR204000)
PTOBankGLAccounts	*PTO Banks* (PR204000)
PurchasingDetail	*Sales Orders* (SO303000)
SalesTerritory	*Sales Territories* (CS204100)
SettingsForPR	*Payment Methods* (CA204000)
ShopifyStore	*Shopify Stores* (BC201010)
StateDetail	*Sales Territories* (CS204100)
TaxAndReportingCA	*Earning Type Codes* (PR102000)
TaxAndReportingUS	*Earning Type Codes* (PR102000)
TaxCodeSetting	*Employee Payroll Settings* (PR203000)
TaxesDecreasingApplWage	*Deduction and Benefit Codes* (PR101060)
TaxesDecreasingApplWageDetail	*Deduction and Benefit Codes* (PR101060)
TaxSettingDetail	*Employee Payroll Settings* (PR203000)
TaxSettingsCA	*Deduction and Benefit Codes* (PR101060)
TaxSettingsUS	*Deduction and Benefit Codes* (PR101060)
TemplateItems	*Template Items* (IN203000)

Entity	Related Form Name and ID
TemplateItemVendorDetail	*Template Items* (IN203000)
UnionDeductionOrBenefitDetail	*Union Locals* (PR209700)
UnionEarningRateDetail	*Union Locals* (PR209700)
WCCCode	*Workers' Compensation Codes* (PR209800)
WCCCodeCostCodeSource	*Workers' Compensation Codes* (PR209800)
WCCCodeLaborItemSource	*Workers' Compensation Codes* (PR209800)
WCCCodeMaxInsurableWage	*Workers' Compensation Codes* (PR209800)
WCCCodeMaxInsurableWageDetail	*Workers' Compensation Codes* (PR209800)
WCCCodeProjectSource	*Workers' Compensation Codes* (PR209800)
WCCCodeRate	*Deduction and Benefit Codes* (PR101060)
WCCCodeRateDetail	*Deduction and Benefit Codes* (PR101060)
WorkCalendar	*Work Calendar* (CS209000)
WorkCalendarExceptionDetail	*Work Calendar* (CS209000)
WorkLocation	*Work Locations* (PR101040)

Table: New Fields and Actions

Field or Action Name	Related Form Name and ID
BusinessAccount.CurrencyID	*Business Accounts* (CR303000)
BusinessAccount.EnableCurrencyOverride	*Business Accounts* (CR303000)
BusinessAccount.OverrideSalesTerritory	*Business Accounts* (CR303000)
BusinessAccount.Relations.Status	*Business Accounts* (CR303000)
BusinessAccount.Relations.Description	*Business Accounts* (CR303000)
BusinessAccount.Relations.OwnerID	*Business Accounts* (CR303000)
BusinessAccount.SalesTerritoryID	*Business Accounts* (CR303000)
ChangeOrder.ChangeOrderCommitment.POType	*Change Orders* (PM308000)
ChangeOrder.HoldChangeOrder	*Change Orders* (PM308000)
ChangeOrder.RemoveChangeOrderFromHold	*Change Orders* (PM308000)
Case.Relations.Status	*Cases* (CR306000)

Field or Action Name	Related Form Name and ID
Case.Relations.Description	*Cases* (CR306000)
Case.Relations.OwnerID	*Cases* (CR306000)
Contact.FullName	*Contacts* (CR302000)
Contact.OverrideSalesTerritory	*Contacts* (CR302000)
Contact.Relations.Status	*Contacts* (CR302000)
Contact.Relations.Description	*Contacts* (CR302000)
Contact.Relations.OwnerID	*Contacts* (CR302000)
Contact.SalesTerritoryID	*Contacts* (CR302000)
Customer.BAccountID	*Customers* (AR303000)
Customer.CreditLimit	*Customers* (AR303000)
Customer.EntityUsageType	*Customers* (AR303000)
Customer.IsGuestCustomer	*Customers* (AR303000)
Customer.NoteID	*Customers* (AR303000)
Customer.RestrictVisibilityTo	*Customers* (AR303000)
Customer.TaxExemptionNumber	*Customers* (AR303000)
CustomerLocation.NoteID	*Customer Locations* (AR303020)
CustomerPaymentMethod.CardType	*Customer Payment Methods* (AR303010)
CustomerPriceClass.NoteID	*Customer Price Classes* (AR208000)
Employee.Delegates.DelegationOf	*Employees* (EP203000)
Employee.Delegates.StartsOn	*Employees* (EP203000)
Employee.Delegates.ExpiresOn	*Employees* (EP203000)
Employee.Delegates.IsActive	*Employees* (EP203000)
InventoryAdjustment.Details.CostCode	*Adjustments* (IN303000)
InventoryAdjustment.Details.CostLayerType	*Adjustments* (IN303000)
InventoryAdjustment.Details.Project	*Adjustments* (IN303000)
InventoryAdjustment.Details.ProjectTask	*Adjustments* (IN303000)

Field or Action Name	Related Form Name and ID
InventoryAdjustment.Details.SpecialOrderNbr	*Adjustments* (IN303000)
InventoryReceipt.Details.CostLayerType	*Receipts* (IN301000)
InventoryReceipt.Details.SpecialOrderNbr	*Receipts* (IN301000)
Invoice.TaxCategory	*Invoices and Memos* (AR301000)
ItemClass.CountryOfOrigin	*Item Classes* (IN201000)
ItemClass.TariffCode	*Item Classes* (IN201000)
ItemSalesCategory.SortOrder	*Item Sales Categories* (IN204060)
ItemSalesCategory.NoteID	*Item Sales Categories* (IN204060)
ItemWarehouse.MaxQty	*Item Warehouse Details* (IN204500)
ItemWarehouse.OverrideMaxQty	*Item Warehouse Details* (IN204500)
ItemWarehouse.OverrideReorderPoint	*Item Warehouse Details* (IN204500)
ItemWarehouse.OverrideSafetyStock	*Item Warehouse Details* (IN204500)
ItemWarehouse.ReorderPoint	*Item Warehouse Details* (IN204500)
ItemWarehouse.SafetyStock	*Item Warehouse Details* (IN204500)
LaborCostRate.Results	*Labor Rates* (PM209900)
Lead.OverrideSalesTerritory	*Leads* (CR301000)
Lead.Relations.Status	*Leads* (CR301000)
Lead.Relations.Description	*Leads* (CR301000)
Lead.Relations.OwnerID	*Leads* (CR301000)
Lead.SalesTerritoryID	*Leads* (CR301000)
NonStockItem.CrossReferences.UOM	*Non-Stock Items* (IN202000)
NonStockItem.CurySpecificMSRP	*Non-Stock Items* (IN202000)
NonStockItem.CurySpecificPrice	*Non-Stock Items* (IN202000)
NonStockItem.Availability	*Non-Stock Items* (IN202000)
NonStockItem.ExportToExternal	*Non-Stock Items* (IN202000)
NonStockItem.Categories	*Non-Stock Items* (IN202000)
NonStockItem.Content	*Non-Stock Items* (IN202000)

Field or Action Name	Related Form Name and ID
NonStockItem.CurrentStdCost	*Non-Stock Items* (IN202000)
NonStockItem.CustomURL	*Non-Stock Items* (IN202000)
NonStockItem.DimensionWeight	*Non-Stock Items* (IN202000)
NonStockItem.FileUrls	*Non-Stock Items* (IN202000)
NonStockItem.MetaDescription	*Non-Stock Items* (IN202000)
NonStockItem.MetaKeywords	*Non-Stock Items* (IN202000)
NonStockItem.MSRP	*Non-Stock Items* (IN202000)
NonStockItem.NoteID	*Non-Stock Items* (IN202000)
NonStockItem.PageTitle	*Non-Stock Items* (IN202000)
NonStockItem.SearchKeywords	*Non-Stock Items* (IN202000)
NonStockItem.TemplateItemID	*Non-Stock Items* (IN202000)
NonStockItem.Visibility	*Non-Stock Items* (IN202000)
NonStockItem.VendorDetails.Default	*Non-Stock Items* (IN202000)
Opportunity.OverrideSalesTerritory	*Opportunities* (CR304000)
Opportunity.Relations.Status	*Opportunities* (CR304000)
Opportunity.Relations.Description	*Opportunities* (CR304000)
Opportunity.Relations.OwnerID	*Opportunities* (CR304000)
Opportunity.SalesTerritoryID	*Opportunities* (CR304000)
Payment.AppliedToOrders	*Payments and Applications* (AR302000)
Payment.AvailableBalance	*Payments and Applications* (AR302000)
Payment.BranchID	*Payments and Applications* (AR302000)
Payment.CardOperation	*Payments and Applications* (AR302000)
Payment.CreditCardTransactionInfo.CardType	*Payments and Applications* (AR302000)
Payment.CreditCardTransactionInfo.ExpirationDate	*Payments and Applications* (AR302000)

Field or Action Name	Related Form Name and ID
Payment.CreditCardTransactionInfo.OrigTranNbr	*Payments and Applications* (AR302000)
Payment.CustomerLocationID	*Payments and Applications* (AR302000)
Payment.ExternalRef	*Payments and Applications* (AR302000)
Payment.IsNewCard	*Payments and Applications* (AR302000)
Payment.NoteID	*Payments and Applications* (AR302000)
Payment.OrigTransaction	*Payments and Applications* (AR302000)
Payment.VoidCardPayment	*Payments and Applications* (AR302000)
PaymentMethod.AllowedCashAccounts.LastModifiedDate-Time	*Payment Methods* (CA204000)
PaymentMethod.AllowedCashAccounts.UseInPR	*Payment Methods* (CA204000)
PaymentMethod.SetPaymentDatetoBankTransactionDate	*Payment Methods* (CA204000)
PaymentMethod.SettingsForPR	*Payment Methods* (CA204000)
PaymentMethod.UseInPR	*Payment Methods* (CA204000)
ProFormaInvoice.HoldProFormaInvoice	*Pro Forma Invoices* (PM307000)
ProFormaInvoice.RemoveProFormaInvoiceFromHold	*Pro Forma Invoices* (PM307000)
Project.ActivateProject	*Projects* (PM301000)
Project.CancelProject	*Projects* (PM301000)
Project.CompleteProject	*Projects* (PM301000)
Project.GLAccounts.DefaultCostAccount	*Projects* (PM301000)
Project.GLAccounts.DefaultCostSubaccount	*Projects* (PM301000)
Project.HoldProject	*Projects* (PM301000)
Project.SuspendProject	*Projects* (PM301000)
ProjectTask.ActivateProjectTask	*Project Tasks* (PM302000)
ProjectTask.CancelProjectTask	*Project Tasks* (PM302000)

Field or Action Name	Related Form Name and ID
ProjectTask.CompleteProjectTask	*Project Tasks* (PM302000)
ProjectTask.DefaultValues.DefaultCostAccount	*Project Tasks* (PM302000)
ProjectTask.DefaultValues.DefaultCostSubaccount	*Project Tasks* (PM302000)
ProjectTask.HoldProjectTask	*Project Tasks* (PM302000)
ProjectTemplate.ActivateProjectTemplate	*Project Templates* (PM208000)
ProjectTemplate.HoldProjectTemplate	*Project Templates* (PM208000)
PurchaseOrder.IsTaxValid	*Purchase Orders* (PO301000)
PurchaseOrder.Details.OrderedQty	*Purchase Orders* (PO301000)
PurchaseReceipt.InventoryRefNbr	*Purchase Receipts* (PO302000)
SalesInvoice.IsTaxValid	*Invoices* (SO303000)
SalesOrder.CreatedDate	*Sales Orders* (SO303000)
SalesOrder.DisableAutomaticTaxCalculation	*Sales Orders* (SO303000)
SalesOrder.ExternalOrderOrigin	*Sales Orders* (SO303000)
SalesOrder.ExternalOrderOriginal	*Sales Orders* (SO303000)
SalesOrder.ExternalOrderSource	*Sales Orders* (SO303000)
SalesOrder.ExternalRefundRef	*Sales Orders* (SO303000)
SalesOrder.NoteID	*Sales Orders* (SO303000)
SalesOrder.PaymentRef	*Sales Orders* (SO303000)
SalesOrder.Payments.CreditCardTransactionInfo.Card-Type	*Sales Orders* (SO303000)
SalesOrder.TaxCalcMode	*Sales Orders* (SO303000)
SalesOrder.UsrExternalOrderOriginal	*Sales Orders* (SO303000)
SalesOrder.WillCall	*Sales Orders* (SO303000)
SalesOrder.Details.AssociatedOrderLineNbr	*Sales Orders* (SO303000)
SalesOrder.Details.ExternalRef	*Sales Orders* (SO303000)
SalesOrder.Details.GiftMessage	*Sales Orders* (SO303000)
SalesOrder.Details.InvoiceLineNbr	*Sales Orders* (SO303000)

Field or Action Name	Related Form Name and ID
SalesOrder.Details.InvoiceNbr	*Sales Orders* (SO303000)
SalesOrder.Details.InvoiceType	*Sales Orders* (SO303000)
SalesOrder.Details.LotSerialNbr	*Sales Orders* (SO303000)
SalesOrder.Details.ManualPrice	*Sales Orders* (SO303000)
SalesOrder.Details.NoteID	*Sales Orders* (SO303000)
SalesOrder.Details.PurchaseWarehouse	*Sales Orders* (SO303000)
SalesOrder.Details.PurchasingDetails	*Sales Orders* (SO303000)
SalesOrder.Details.VendorID	*Sales Orders* (SO303000)
SalesOrder.Details.Allocations.CustomerOrderNbr	*Sales Orders* (SO303000)
SalesOrder.Details.Allocations.SchedOrderDate	*Sales Orders* (SO303000)
SalesOrder.Payments.ExternalRef	*Sales Orders* (SO303000)
SalesOrder.Payments.NoteID	*Sales Orders* (SO303000)
SalesOrder.Shipments.InventoryNoteID	*Sales Orders* (SO303000)
SalesOrder.Shipments.LastModifiedDateTime	*Sales Orders* (SO303000)
SalesOrder.Shipments.OrderNoteID	*Sales Orders* (SO303000)
SalesOrder.Shipments.ShippingNoteID	*Sales Orders* (SO303000)
SalesOrder.ShippingSettings.OverrideFreightPrice	*Sales Orders* (SO303000)
SalesOrder.Totals.Freight	*Sales Orders* (SO303000)
SalesOrder.Totals.FreightCost	*Sales Orders* (SO303000)
SalesOrder.Totals.FreightCostIsuptodate	*Sales Orders* (SO303000)
SalesOrder.Totals.FreightTaxCategory	*Sales Orders* (SO303000)
SalesOrder.Totals.OrderVolume	*Sales Orders* (SO303000)
SalesOrder.Totals.OrderWeight	*Sales Orders* (SO303000)
SalesOrder.Totals.OverrideFreightAmount	*Sales Orders* (SO303000)
SalesOrder.Totals.PackageWeight	*Sales Orders* (SO303000)
SalesOrder.Totals.PremiumFreight	*Sales Orders* (SO303000)
SalesPricesInquiry.SalesPriceDetails.NoteID	*Sales Prices* (AR202000)

Field or Action Name	Related Form Name and ID
SalesPricesInquiry.SalesPriceDetails.TaxCalculationMode	*Sales Prices* (AR202000)
SalesPricesInquiry.SalesPriceDetails.Warehouse	*Sales Prices* (AR202000)
SalesPricesInquiry.TaxCalculationMode	*Sales Prices* (AR202000)
Shipment.Details.Allocations.SplitLineNbr	*Shipments* (SO302000)
Shipment.NoteID	*Shipments* (SO302000)
Shipment.Orders.OrderNoteID	*Shipments* (SO302000)
Shipment.Packages.Height	*Shipments* (SO302000)
Shipment.Packages.Length	*Shipments* (SO302000)
Shipment.Packages.LineNbr	*Shipments* (SO302000)
Shipment.Packages.NoteID	*Shipments* (SO302000)
Shipment.Packages.PackageContents.OrigOrderNbr	*Shipments* (SO302000)
Shipment.Packages.PackageContents.OrigOrderType	*Shipments* (SO302000)
Shipment.Packages.Width	*Shipments* (SO302000)
ShippingBox.LinearUOM	*Boxes* (CS207600)
StockItem.Availability	*Stock Items* (IN202500)
StockItem.CountryOfOrigin	*Stock Items* (IN202500)
StockItem.CrossReferences.UOM	*Stock Items* (IN202500)
StockItem.CurySpecificMSRP	*Stock Items* (IN202500)
StockItem.CurySpecificPrice	*Stock Items* (IN202500)
StockItem.CustomURL	*Stock Items* (IN202500)
StockItem.ExportToExternal	*Stock Items* (IN202500)
StockItem.FileURLs	*Stock Items* (IN202500)
StockItem.MetaDescription	*Stock Items* (IN202500)
StockItem.MetaKeywords	*Stock Items* (IN202500)
StockItem.NotAvailable	*Stock Items* (IN202500)
StockItem.NoteID	*Stock Items* (IN202500)

Field or Action Name	Related Form Name and ID
StockItem.PageTitle	*Stock Items* (IN202500)
StockItem.SearchKeywords	*Stock Items* (IN202500)
StockItem.TariffCode	*Stock Items* (IN202500)
StockItem.TemplateItemID	*Stock Items* (IN202500)
StockItem.Visibility	*Stock Items* (IN202500)
TransferOrder.Details.CostCode	*Transfers* (IN304000)
TransferOrder.Details.Project	*Transfers* (IN304000)
TransferOrder.Details.ProjectTask	*Transfers* (IN304000)
TransferOrder.Details.SpecialOrderNbr	*Transfers* (IN304000)
TransferOrder.Details.ToCostCode	*Transfers* (IN304000)
TransferOrder.Details.ToCostLayerType	*Transfers* (IN304000)
TransferOrder.Details.ToProject	*Transfers* (IN304000)
TransferOrder.Details.ToProjectTask	*Transfers* (IN304000)
TransferOrder.Details.ToSpecialOrderNbr	*Transfers* (IN304000)
Vendor.LegalName	*Vendors* (AP303000)

Table: Renamed Fields and Actions

Old Field or Action	New Field or Action Name	Related Form Name and ID
ItemWarehouse.Over-rideServiceLevelOverride	OverrideServiceLevel	*Item Warehouse Details* (IN204500)
LaborCostRate.EmployeeID	Employee	*Labor Rates* (PM209900)
LaborCostRate.ProjectID	Project	*Labor Rates* (PM209900)
LaborCostRate.Project-TaskID	ProjectTask	*Labor Rates* (PM209900)
LaborCostRate.UnionLo-calID	UnionLocal	*Labor Rates* (PM209900)
ProjectTask.Attribute	Attributes	*Project Tasks* (PM302000)
SalesOrder.ShippingSet-tings.Freight	FreightPrice	*Sales Orders* (SO301000)

Old Field or Action	New Field or Action Name	Related Form Name and ID
Shipment.FreightAmount	FreightPrice	*Shipments* (SO302000)

Table: Removed Entities

Field or Action Name	Related Form Name and ID
AllocationRule	*Allocation Rules* (PM207500)
CommonTask	*Common Tasks* (PM208030)
ProjectBilling	*Run Project Billing* (PM503000)
ProjectBillingDetails	*Run Project Billing* (PM503000)
ProjectBillingRules	*Billing Rules* (PM207000)

Table: Removed Fields and Actions

Field or Action Name	Related Form Name and ID	Comment
Bill.IsTaxValid	*Bills and Adjustments* (AP301000)	
ChangeOrder.Commitments.OpenAmount	*Change Orders* (PM308000)	
ChangeOrder.RevenueBudget.ActualAmount	*Change Orders* (PM308000)	
ChangeOrder.RevenueBudget.ActualQty	*Change Orders* (PM308000)	
ChangeOrder.RevenueBudget.Completed	*Change Orders* (PM308000)	
ChangeOrder.RevenueBudget.DraftInvoicesAmount	*Change Orders* (PM308000)	
ChangeOrder.RevenueBudget.OriginalBudgetedAmount	*Change Orders* (PM308000)	
ChangeOrder.RevenueBudget.OriginalBudgetedQty	*Change Orders* (PM308000)	
LaborCostRate.AnnualRate	*Labor Rates* (PM209900)	Use LaborCostRate.Results instead.
LaborCostRate.CurrencyID	*Labor Rates* (PM209900)	Use LaborCostRate.Results instead.

Field or Action Name	Related Form Name and ID	Comment
LaborCostRate.Description	*Labor Rates* (PM209900)	Use LaborCostRate.Results instead.
LaborCostRate.EmployeeName	*Labor Rates* (PM209900)	Use LaborCostRate.Results instead.
LaborCostRate.ExternalRefNbr	*Labor Rates* (PM209900)	Use LaborCostRate.Results instead.
LaborCostRate.HourlyRate	*Labor Rates* (PM209900)	Use LaborCostRate.Results instead.
LaborCostRate.RecordID	*Labor Rates* (PM209900)	Use LaborCostRate.Results instead.
LaborCostRate.RegularHoursPerWeek	*Labor Rates* (PM209900)	Use LaborCostRate.Results instead.
LaborCostRate.TypeOfEmployment	*Labor Rates* (PM209900)	Use LaborCostRate.Results instead.
SalesOrder.Details.PurchasingSettings	*Sales Orders* (SO303000)	
SalesOrder.Relations	*Sales Orders* (SO301000)	
SalesOrder.SalesOrderAddInvoice	*Sales Orders* (SO301000)	

OpenAPI 3.0

Acumatica ERP provides you with the contract-based REST APIs of web service endpoints and with the contract-based REST API of an Acumatica ERP instance. The methods of a REST API are provided in a `swagger.json` file, which is an OpenAPI 3.0 file. For more information about the OpenAPI specification, see *https://www.openapis.org*. You can use the `swagger.json` files to review the APIs of the instance and endpoints and build the client applications for Acumatica ERP based on these files.

OpenAPI 3.0 of an Endpoint

You can retrieve the `swagger.json` file of a web service endpoint by clicking **OpenAPI 3.0** on the More menu of the *Web Service Endpoints* (SM207060) form or by using the following URL.

```
http://<Base endpoint URL>/swagger.json?company=<Tenant name>
```

In this URL, `<Base endpoint URL>` is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format.

```
http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/
```

You can specify the `company` URL parameter to obtain information on the API of the endpoint available in a particular tenant. For example, suppose that you want to retrieve the API reference of the custom endpoint with the name *MyEndpoint* and Version 24.200.001 available in the *MyTenant* tenant from a local Acumatica ERP instance with the name *AcumaticaDB*. You would use the following URL to obtain the `swagger.json` file: *http://localhost/AcumaticaDB/entity/MyEndpoint/24.200.001/swagger.json?company=MyTenant*.

If no tenant is specified, the API of the endpoint available in the tenant to which the user is currently signed in is returned.

OpenAPI 3.0 of an Instance

You can retrieve the `swagger.json` file of an Acumatica ERP instance by using the following URL.

```
http://<Acumatica ERP instance URL>/entity/swagger.json
```

For example, if you use a local instance with the name *AcumaticaU100*, you would retrieve the `swagger.json` file related to this instance by using the following URL: *http://AcumaticaU100/entity/swagger.json*.

The contract-based REST API of an instance contains the methods that do the following:

- Signing in to Acumatica ERP
- Signing out from Acumatica ERP
- Getting the Acumatica ERP version

Representation of a Record in JSON Format

By using the contract-based REST API, you obtain existing records from Acumatica ERP, create new records, update, and delete them. You work with the records in Acumatica ERP by using the entities that are defined in the contract of the endpoint that you use to access the service. You pass records to and receive them from the contract-based REST API in JavaScript object notation (JSON) format. JSON is a text format for transmitting data objects that consist of key-value pairs.

To represent a record in JSON format, you use the rules that are described in the following sections. You do not need to specify the values of all fields of an entity; you can specify the values of only the needed fields.

System Fields

You specify the value of a system field (such as `id` or `rowNumber`) of an entity in the following format.

```
<Field name> : <Value>
```

For example, if you need to specify the row number *1* for an entity, you use the following string.

```
"rowNumber" : 1
```

You specify the value of the `note` field in the same way as you would any general field (see the next section).

General Fields

You specify the value of a general field (that is, a field that is not a system field) of an entity in the following format.

```
<Field name> : {"value" : <Value>}
```

For example, if you need to specify *JOHNGOOD* as the customer ID of a customer record, you use the following string.

```
"CustomerID" : {"value" : "JOHNGOOD"}
```

Linked Entities

You specify the values of the fields of a linked entity in the following format.

```
<Field name> :
  {
    <List of fields of the linked entity with values>
  }
```

For example, if you need to specify the values of an email address and the address of a customer main contact, you use the following string.

```
"MainContact" :
    {
      "Email" : {"value" : "demo@gmail.com" },
      "Address" :
        {
          "AddressLine1" : {"value" : "4030 Lake Washington Blvd NE" },
          "AddressLine2" : {"value" : "Suite 100" },
          "City" : {"value" : "Kirkland" },
          "State" : {"value" : "WA" },
          "PostalCode" : {"value" : "98033" }
        }
    }
```

Detail Entities

You specify the values of the fields of a detail entity in the following format.

```
<Field name> :
  [
    {
      <List of fields of the detail entity with the values>
    },
    {
      <List of fields of the detail entity with the values>
    },
    …
  ]
```

For example, if you need to specify the values of two detail lines of a sales order, you use the following string.

```
"Details" : [
```

```
  {
    "InventoryID" : {"value": "AALEGO500"},
    "Quantity" : {"value": 10},
    "UOM" : {"value": "PIECE"}
  },
  {
    "InventoryID" : {"value": "CONGRILL"},
    "Quantity" : {"value": 1},
    "UOM" : {"value": "PIECE"}
  }
]
```

Custom Fields

You specify the values of the custom fields (that is, the fields that are not included in the contract of the endpoint) in the following format.

Custom fields can correspond to the following elements:

- The predefined elements on an Acumatica ERP form that are not included in the entity definition
- The elements that were added to the Acumatica ERP form in a customization project
- The user-defined fields

```
"custom" :
  {
    <View name> :
     {
       <Field name> :
         {
           "type" : <value>,
           "value" : <value>
         }
     }
  }
```

You use this block in the JSON representation of the entity (top-level, detail, or linked) that contains this custom field.

For details on how to find out the field name and the name of the data view, see *Custom Fields*.

For example, suppose that you added the **Personal ID** element to the **Main Contact** area of the *Customers* (AR303000) form in a customization project. The `Contact` entity, which is available through the `MainContact` property of the `Customer` entity, contains the **Personal ID** custom element. This element has the `UsrPersonalID` field name and belongs to the `DefContact` data view. To specify the value *AB123456* of the **Personal ID** custom element for the customer with ID *JOHNGOOD* through the REST API, you use the following request body.

```
{
  "CustomerID" : {value : "JOHNGOOD" } ,
  "MainContact" :
    {
      "custom" :
        {
          "DefContact" :
```

```
            {
              "UsrPersonalID" :
                {
                  "type" : "CustomStringField",
                  "value" : "AB123456"
                }
            }
        }
    }
}
```

To Create a Custom Endpoint

You use the *Web Service Endpoints* (SM207060) form to create a custom endpoint.

If you need to use a custom endpoint, you can either create an endpoint from scratch or extend an existing endpoint with the needed API. This procedure describes how to create a custom endpoint from scratch. To learn how to extend an existing endpoint, see *To Extend an Existing Endpoint*.

To Create an Endpoint from Scratch

You can create an endpoint that has the latest version of the contract only.

1. Open the *Web Service Endpoints* (SM207060) form.
2. In the **Endpoint Name** box, type the name of the new endpoint.

   

   For details on the characters that can be used in the endpoint name and version, see *Naming Rules for Endpoints*.
3. In the **Endpoint Version** box, type the version of the new endpoint.
4. Add the needed entities, fields, and actions to the contract of the created endpoint, as described in the sections below.
5. Click **Save** on the form toolbar.

To Add a Top-Level Entity to the Contract of the Endpoint

1. In the **Endpoint Name** box, select the name of the endpoint to which you want to add an entity.
2. In the **Endpoint Version** box, select the version of the endpoint to which you want to add an entity.
3. In the left pane, select the *Endpoint* node.
4. On the toolbar of the left pane, click **Insert**, and in the **Create Entity** dialog box, do the following:
   a. In the **Object Name** box, type the name of the entity. This is the name of the API object that you will use in the code of your application to work with the entity.

      

      For details on the characters that can be used in the entity names, see *Naming Rules for Endpoints*.
   b. In the **Object Type** box, select *Top-Level*.
   c. In the **Screen Name** lookup box, select the form to which the entity should correspond.

d. Click **OK**.

5. Add the needed fields, actions, or nested entities to the entity, as described in the sections below.

To Add a Report Entity to the Contract of the Endpoint

1. In the **Endpoint Name** box, select the name of the endpoint to which you want to add an entity.
2. In the **Endpoint Version** box, select the version of the endpoint to which you want to add an entity.
3. In the left pane, select the *Endpoint* node.
4. On the toolbar of the left pane, click **Insert**, and in the **Create Entity** dialog box, do the following:
   a. In the **Object Name** box, type the name of the entity. This is the name of the API object that you will use in the code of your application to work with the entity.

      

      For details on the characters that can be used in the entity names, see *Naming Rules for Endpoints*.

   b. In the **Object Type** box, select *Report*.
   c. In the **Screen Name** lookup box, select the report to which the entity should correspond.
   d. Click **OK**.

   The **Fields** tab is populated with the fields for each parameter of the report.
5. Optional: Modify the list of fields, as described in *To Add Fields to an Entity*.

To Add a Linked or Detail Entity to Another Entity

1. In the **Endpoint Name** box, select the name of the endpoint to which you want to add an entity.
2. In the **Endpoint Version** box, select the version of the endpoint to which you want to add an entity.
3. In the left pane, select the entity node to which you want to add a linked or detail entity.
4. On the toolbar of the left pane, click **Insert**.
5. In the **Field Name** box of the **Create Entity** dialog box, which opens, type the name of the field that should be used to access the nested entity, and specify the values of other elements in one of the following ways:
   - If you want to insert an entity that already exists in the contract, select the **Use Existing Entity** check box, and select the needed entity in the **Entity Type** box.
   - If you want to insert a new entity, in the **Object Name** box, type the name of the entity, and in the **Object Type** box, select the type of the entity: *Linked* or *Detail*.

   

   For details on the characters that can be used in the entity names, see *Naming Rules for Endpoints*.
6. Click **OK**. The new entity appears in the contract.
7. Add fields to the created entity, as described in the following section.

To Add Fields to an Entity

1. In the **Endpoint Name** box, select the name of the endpoint to which you want to add an entity.
2. In the **Endpoint Version** box, select the version of the endpoint to which you want to add an entity.
3. In the left pane, select the entity node to which you want to add fields.
4. On the **Fields** tab of the right pane, do one of the following:

- Click **Populate** on the tab toolbar. In the **Populate Fields** dialog box, select the Acumatica ERP object whose fields you want to include in the entity and the fields that you want to include, and click **OK**. The selected fields are added to the contract.
- Click **Add Row** on the tab toolbar; then type the name of the new field in the **Field Name** column of the added row, select the Acumatica ERP object whose field you want to include in the entity in the **Mapped Object** column, and select the field that you want to include in the **Mapped Field** column.

> ⚠
> - For some fields to be included in the entity, the corresponding Acumatica ERP feature or features must be enabled on the *Enable/Disable Features* (CS100000) form. For information on Acumatica ERP basic functionality and add-on features, see *Preparing an Instance: Acumatica ERP Features*.
> - For details on the characters that can be used in the field names, see *Naming Rules for Endpoints*.

5. Click **Save** on the form toolbar.

To Add an Action to an Entity

> ⚠ You can add actions that are performed on multiple records (such as the removal of all records or those the user selects) to processing forms only. Actions that can be performed on multiple records are not supported for data entry and maintenance forms.

1. In the **Endpoint Name** box, select the name of the endpoint to which you want to add an entity.
2. In the **Endpoint Version** box, select the version of the endpoint to which you want to add an entity.
3. In the left pane, select the *Actions* node in the needed entity.
4. On the toolbar of the left pane, click **Insert**.
5. In the **Create Action** dialog box, which opens, select the needed Acumatica ERP action, type the name that should be used to invoke this action through the API, and click **OK**. The dialog box is closed, and the new action is added to the contract.

> ⚠ For details on the characters that can be used in the action names, see *Naming Rules for Endpoints*.

6. If the action has parameters, add the parameters to the action as follows:
   a. In the left pane, click the action you have created.
   b. On the **Parameters** tab of the right pane, do one of the following:
      - Click **Populate** on the tab toolbar. In the **Populate Fields** dialog box, which opens, select the Acumatica ERP object whose fields you want to use as parameters of the action and the fields that you want to use as parameters, and click **OK**. The selected fields are added to the contract.
      - Click **Add Row** on the tab toolbar; then type the name of the new parameter in the **Parameter Name** column of the added row, select the Acumatica ERP object whose field you want to use a parameter of the action in the **Mapped Object** column, and select the field that you want to use as a parameter in the **Mapped Field** column.

> ⚠
> - For details on the characters that can be used in the parameter names, see *Naming Rules for Endpoints*.
> - If you need to add parameters of a workflow action, select the *Transition Parameters* object as the object whose fields you want to use as parameters. For details about workflow actions, see *Action Configuration: General Information*.

7. On the form toolbar, click **Save**.

To Extend an Existing Endpoint

You use the *Web Service Endpoints* (SM207060) form to create an endpoint as an extension of an existing endpoint.

You may need to create an extension of an endpoint if you want to use the entities that are defined in the contract of the existing endpoint but you also need some additional entities, fields, and actions in the contract. For example, the contract of the system endpoint with the name *Default* and Version *24.200.001* contains the `Address` entity, which includes the following fields: `AddressLine1`, `AddressLine2`, `City`, `Country`, `PostalCode`, `State`, and `Validated`. Suppose that you want to add the new `GPSCoordinates` field to the `Address` entity of the contract and use it with other API of the contract. You cannot edit the contract of the system endpoint; instead, you should create an endpoint that is based on this system endpoint, and add the new `GPSCoordinates` field to the `Address` entity of the contract of the new endpoint.

This procedure describes how to create an endpoint that is based on an existing endpoint.

To Extend an Existing Endpoint

1. Open the *Web Service Endpoints* (SM207060) form.
2. Select the endpoint that you want the new endpoint to be based on as follows:
   a. Select the name of the base endpoint in the **Endpoint Name** box.
   b. Select the version of the base endpoint in the **Endpoint Version** box.
3. Click **Extend Endpoint** on the form toolbar.
4. In the **Extend Current Endpoint** dialog box, which opens, make sure the correct name and version of the base endpoint are specified in the **Base Endpoint Name** and **Base Endpoint Version** boxes. Specify the name of the new endpoint in the **Endpoint Name** box and the version of the new endpoint in the **Endpoint Version** box and click **OK**.

   

   For details on the characters that can be used in the endpoint name and version, see *Naming Rules for Endpoints*.

   The new endpoint with the name and version you specify appears on the form. On the left pane of the form, you can see the list of entities that were inherited from the base endpoint.
5. Add the needed entities, fields, and actions to the contract of the created endpoint, as described in *To Create a Custom Endpoint*, or extend the entities inherited from the base endpoint, as described in *To Extend an Existing Entity*.
6. Click **Save** on the form toolbar.

To Extend an Existing Entity

1. Select the extended endpoint in which you want to extend an entity inherited from the base endpoint as follows:
   a. In the **Endpoint Name** box, select the name of the extended endpoint.
   b. In the **Endpoint Version** box, select the version of the extended endpoint.
2. In the left pane, select the entity inherited from the base endpoint to which you want to add new fields, linked or detail entities, or actions.
3. If you want to add fields to the entity, do the following:

a. On the toolbar of the **Fields** tab in the right pane, click **Extend Entity**.

b. Use the **Add Row**, **Delete Row**, and **Populate** buttons, which have become available on the tab toolbar, to add and delete fields of the entity. For more details, see *To Add Fields to an Entity*.

4. If you want to add actions or linked or detail entities, follow the instructions in *To Create a Custom Endpoint*.
5. Click **Save** on the form toolbar.

To Validate an Endpoint

You use the *Web Service Endpoints* (SM207060) form to validate an endpoint, an entity, or an action. During this validation, the system makes sure the following criteria are met for the elements of the endpoint, entity, or action:

- The names of the elements satisfy the naming rules. For details on these rules, see *Naming Rules for Endpoints*.
- The elements are mapped to objects, fields, and actions that exist in the system.

The validation of the name of a new entity, field, action, or action parameter is performed automatically once you have entered the name on the form. You can validate an endpoint, entity, or action manually, as described in this topic.

To Validate an Endpoint

1. Open the *Web Service Endpoints* (SM207060) form.
2. Select the endpoint that you want to validate as follows:
   a. In the **Endpoint Name** box, select the name of the endpoint.
   b. In the **Endpoint Version** box, select the version of the endpoint.
3. On the form toolbar, click **Validate Endpoint**. The long-running validation operation starts.

   Once the validation is finished, the system displays a message with the results of the validation. If the validation has failed, the error message contains the names of all fields that caused the error.
4. If any errors occur, correct the endpoint accordingly.

To Validate an Entity

1. Open the *Web Service Endpoints* (SM207060) form.
2. Select the endpoint that contains the entity that you want to validate as follows:
   a. In the **Endpoint Name** box, select the name of the endpoint.
   b. In the **Endpoint Version** box, select the version of the endpoint.
3. In the left pane, click the entity that you want to validate.
4. On the toolbar of the **Fields** tab of the right pane, click **Validate Entity**.

   Once the validation is finished, the system displays a message with the results of the validation. If the validation has failed, the error message contains the names of all fields that caused the error.
5. If any errors occur, correct the entity accordingly.

To Validate an Action

1. Open the *Web Service Endpoints* (SM207060) form.
2. Select the endpoint that contains the action that you want to validate as follows:

a. In the **Endpoint Name** box, select the name of the endpoint.

b. In the **Endpoint Version** box, select the version of the endpoint.

3. In the left pane, click the action that you want to validate.
4. On the toolbar of the **Parameters** tab of the right pane, click **Validate Action**.

   Once the validation is finished, the system displays a message with the results of the validation. If the validation has failed, the error message contains the names of all fields that caused the error.
5. If any errors occur, correct the action accordingly.

To Import a REST Schema to a Visual Studio Solution

You can create a Visual Studio project and use the REST schema contained in the Acumatica ERP `swagger.json` files for interaction with Acumatica ERP. (For details about the `swagger.json` files, see *OpenAPI 3.0*.) You import the REST schema to a Visual Studio solution in two stages:

1. You generate Visual Studio packages from the REST schema.
2. You include the generated Visual Studio packages in your Visual Studio solution.

The following sections provide instructions on these stages, as well as an example of the use of the imported REST schema.

To Generate Visual Studio Packages from a REST Schema

1. Obtain the `swagger.json` file you need.
2. Open the website *https://editor.swagger.io*.
3. Insert the contents of the `swagger.json` file into the website's edit box.

   The website suggests that you convert the JSON contents to YAML format.
4. Click **OK** to agree to the suggestion or **Cancel** to leave the contents in JSON format; neither of these actions affects the result of parsing the file.

   The website engine parses the REST schema contained in the `swagger.json` file and create the visual representation of the REST schema.
5. On the **Generate Client** menu, invoke the **csharp** command.

   The `csharp-client-generated.zip` archive is downloaded to your computer. The archive contains the *IO.Swagger* Visual Studio solution, two Visual C# projects (*IO.Swagger* and *IO.Swagger.Test*), and other files.
6. Extract the `csharp-client-generated.zip` archive.
7. Make sure that the `IO.Swagger` solution can be built.

To compile the *IO.Swagger* and *IO.Swagger.Test* projects, you may have to explicitly specify the path to the `RestSharp.dll` library. In addition, you may get compilation errors related to the misuse of the `override` keyword and the absence of the `BaseValidate` method. To fix these errors, remove the misused `override` keyword and the entire command that contains the nonexistent `BaseValidate` method.

To Add the Generated Projects to a Visual Studio Solution

1. In Visual Studio, create a C# project.

2. In your project, add a reference to the `Newtonsoft.Json` library as follows:
   a. In Solution Explorer, right-click your project.
   b. Click **Manage NuGet Packages**.
   c. Select the **Browse** tab.
   d. Search for the `Newtonsoft.Json` library, and install it.
3. In the folders containing the unpacked contents from the `csharp-client-generated.zip` archives, rename the *IO.Swagger* projects and the `IO.Swagger` namespaces to avoid the name conflict.

   You do not have to add the `IO.Swagger.Test` projects to the solution of your C# project.
4. Add the former *IO.Swagger* projects to the solution of your C# project.
5. In your project, add a reference to the `RestSharp.dll` library.
6. Add your code that uses the REST API to your project.

> You can find other examples in the *AcumaticaRESTAPIClientForCSharp* repository on GitHub.

Web Service Endpoints: Endpoint in a Customization Project

If you need to transfer the configuration of a web service endpoint, which is defined on the *Web Service Endpoints* (SM207060) form, to another Acumatica ERP instance, you need to include the respective endpoint in a customization project.

> After you have included all needed items in a customization project, you export the project as a ZIP file. In the target instance, you import the file and publish this customization project. For details about importing, exporting, and publishing customization projects, see *Managing Customization Projects* and *Publishing Customization Projects*.

Web Service Endpoint in a Customization Project

You can configure contract-based web service endpoints in an instance of Acumatica ERP and include the new configuration in a customization project as an *EntityEndpoint* item. You use the *Web Service Endpoints* (AU206002) page of the Customization Project Editor to manage *EntityEndpoint* items in the customization project.

> Web service endpoints are not included in snapshots. To copy an endpoint to a different tenant, create a customization project, add an endpoint to it, and publish the customization project to the needed tenant.

If you have used the *Web Service Endpoints* (SM207060) form to change a custom contract-based web service endpoint included in a customization project and you want to include these changes in the customization project, you have to update the appropriate item in the customization project by clicking **Reload from Database** on the toolbar of the *Web Service Endpoints* (AU206002) page of the Customization Project Editor.

Web Service Endpoints: To Include an Endpoint in a Customization Project

This activity will walk you through the process of including an endpoint in a customization project.

Story

Suppose that you need to distribute an integration application that uses a custom web service endpoint to the other Acumatica ERP instances of the company. You need to include the endpoint in a customization project. You can then export this customization project to a ZIP file, import the file to the target instance, and publish this customization project.

Process Overview

You will include the needed endpoint in a customization project.

System Preparation

Before you begin performing this activity, do the following:

1. Deploy an instance of Acumatica ERP with the name *MyStoreInstance* and a tenant that has the *MyStore* name and contains the *T100* data.
2. Complete the following prerequisite activity: *Generic Inquiries in a Customization ProjectActivity 4.1.1: To Include Generic Inquiries in a Customization Project*. In this activity, a customization project is created.

Step: Including an Endpoint in the Customization Project

You will include in the customization project the *ItemAvailabilityData/0001* custom endpoint, which is preconfigured in the instance.

To include the endpoint in the customization project, do the following:

1. In the navigation pane of the Customization Project Editor, click **Web Service Endpoints** to open the *Web Service Endpoints* page.
2. On the page toolbar, click **Add New Record**.
3. In the **Add Entity Endpoint** dialog box, which opens, select the unlabeled check box in the row with the *ItemAvailabilityData* endpoint and version *0001*.
4. Click **Save**.

   The endpoint has been added to the Web Service Endpoints page.

REST API Examples

In this chapter, you can find examples of HTTP requests that show how to implement various integration scenarios for the integration of Acumatica ERP with external systems through the contract-based REST API. You can reuse these requests in your application.

You can find the REST examples in the `IntegrationDevelopmentGuide.postman_collection.json` file (which is a Postman collection) of the `IntegrationDevelopment\Help` folder in the *Help-and-Training-Examples* repository on GitHub.

Basic Requests

Through the contract-based representational state transfer (REST) application programming interface (API) of Acumatica ERP external systems can get data records from Acumatica ERP, process these records, and save new or updated records to Acumatica ERP.

This chapter includes examples of basic REST API requests.

Sign In to the Service

Each time your application starts working with the Acumatica ERP contract-based REST service, you have to sign in to Acumatica ERP. To sign in to Acumatica ERP, you access the needed URL with the `POST` HTTP method and pass the credentials in the request body. See the following sections for details on the request and the response.

If two-factor authentication is turned on for a user account that is used to sign in to Acumatica ERP through the REST API, this authentication is bypassed by the sign-in method. You can explicitly turn off two-factor authentication for the user accounts that are used only through the REST API by defining the user type of the user account. For details about the creation of a user type to be used by an external application, see *To Limit the Number of API Connections of Integrated Applications*.

Instead of directly signing in to Acumatica ERP, your application can implement the OAuth 2.0 or OpenID Connect (OIDC) mechanism of authorization. For details about OAuth 2.0 and OIDC, see *Authorizing Client Applications to Work with Acumatica ERP*.

HTTP Method and URL

When you need to sign in to Acumatica ERP, you use the `POST` HTTP method and the following URL.

```
POST http://<Acumatica ERP instance URL>/entity/auth/login
```

You replace *<Acumatica ERP instance URL>* with the URL of your Acumatica ERP instance.

For example, suppose that you want to sign in to a local Acumatica ERP instance with the name *AcumaticaDB*. You should use the following URL: *http://localhost/AcumaticaDB/entity/auth/login*.

Parameters

You do not need to use parameters when you sign in to Acumatica ERP.

Request Headers

You have to specify the following header in the request.

Header	Description
`Content-Type`	The format of the request body, which can be one of the following: • `application/json` • `application/x-www-form-urlencoded`

Request Body

In the request body, you pass the credentials for accessing Acumatica ERP in JSON format, as shown in the following example.

```
{
  "name" : "admin",
  "password" : "123",
  "tenant" : "MyStore",
  "branch" : "MYSTORE",
  "locale" : "EN-US"
}
```

You specify the values of the parameters as follows:

- *name*: The username that the application should use to sign in to Acumatica ERP, such as `"admin"`.
- *password*: The password for the username, such as `"123"`.
- *tenant*: The name of the tenant to which the application should sign in, such as `"MyStore"`. You can view the name that should be used for the tenant in the **Login Name** box of the *Tenants* (SM203520) form.
- *branch*: The ID of the branch to which the application should sign in. You can view the ID of the branch in the **Branch ID** box of the *Branches* (CS102000) form.
- *locale*: The locale that should be used in Acumatica ERP. The locale is specified in the `System.Globalization.CultureInfo` format converted to `string`, as with `"EN-US"`.

  

  This parameter has been developed for future use. You do not need to set its value.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a sign-in request.

Code	Description
204	The request has been completed successfully. The response headers contain the cookies that authorize the user to make further requests.
400	The data specified in the request is invalid.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).

Code	Description
500	An internal server error has occurred.

Example

The following request shows an example of a sign-in to Acumatica ERP through the REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /entity/auth/login HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

{
  "name": "admin",
  "password": "123",
  "tenant": "MyStore",
  "branch": "MYSTORE"
}
```

Usage Notes

With each subsequent request to the service, the application has to pass the cookies that it has received during sign-in.

For each attempt to sign in to Acumatica ERP, you must implement the signing out from the service after you finish your work with Acumatica ERP to close the session. If the session is not closed, you may have issues with subsequent sign-ins to Acumatica ERP through the REST API. For details about signing out, see *Sign Out from the Service*.

You should also take into account Acumatica ERP license API limits. For details, see *License Restrictions for API Users*.

Sign Out from the Service

Each time your application finishes working with the Acumatica ERP contract-based REST service, you have to sign out from Acumatica ERP. To sign out from Acumatica ERP, you access the needed URL address with the `POST` HTTP method. See the following sections for details on the request and the response.

HTTP Method and URL

When you need to sign out from Acumatica ERP, you use the `POST` HTTP method and the following URL.

```
POST http://<Acumatica ERP instance URL>/entity/auth/logout
```

You replace *<Acumatica ERP instance URL>* with the URL of your Acumatica ERP instance.

For example, suppose that you want to sign out from a local Acumatica ERP instance with the name *AcumaticaDB*. You should use the following URL: *http://localhost/AcumaticaDB/entity/auth/logout*.

Parameters

You do not need to use parameters when you sign out from Acumatica ERP.

Request Headers

You do not specify any header in the request.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a sign-out request.

Code	Description
204	The request has been completed successfully.
400	The data specified in the request is invalid.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of a sign-out from Acumatica ERP through the REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /entity/auth/logout HTTP/1.1
Host: [<Acumatica ERP instance URL>]
```

Usage Notes

If your applications failed to sign out from Acumatica ERP and the maximum number of concurrent sessions allowed by the license is reached, there is no way to forcibly terminate all sessions of the API users. Instead, a session is automatically closed after a 10-minute timeout.

Create a Record

When you need to create a record by using the contract-based REST API, you access the needed URL with the `PUT` HTTP method and pass the record representation in JSON format in the request body. See the following sections for details on the request and the response.

HTTP Method and URL

To create a record in Acumatica ERP, you use the `PUT` HTTP method and the following URL.

```
PUT http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to create a record.

For example, suppose that you want to create a stock item record in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to create a record: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

You can use the following parameters when you retrieve a record from Acumatica ERP:

- *$expand*: To specify the linked and detail entities to be expanded. For more information, see *$expand Parameter*.

  

  You must list in the *$expand* parameter every detail and related entity that you are going to have in the response body.

- *$select*: To specify the fields of the entity to be returned. For more information about the parameter, see *$select Parameter*.
- *$custom*: To specify the fields that are not defined in the contract to be returned. For details about the parameter, see *$custom Parameter*.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.
`If-None-Match`	Changes the behavior of the `PUT` request, which normally either creates a new record or updates an existing one. If you only want to create a new record, use the optional `If-None-Match` header with the `*` (asterisk) value.

Request Body

You pass a record in JSON format in the request body. You can find details on how to represent a record in JSON format in *Representation of a Record in JSON Format*. See below for an example of a customer record representation in JSON format.

```
{
  "CustomerID" : {value : "JOHNGOOD"},
```

```
  "CustomerName" : {value : "John Good"},
  "CustomerClass" : {value : "DEFAULT"},
  "MainContact" :
    {
      "Email" : {value : "demo@gmail.com"},
      "Address" :
        {
          "AddressLine1" : {"value" : "4030 Lake Washington Blvd NE"},
          "AddressLine2" : {"value" : "Suite 100"},
          "City" : {"value" : "Kirkland"},
          "State" : {"value" : "WA" },
          "PostalCode" : {"value" : "98033"}
        }
    }
}
```

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that creates a record.

Code	Description
200	The request has been completed successfully. The response of a successful method call contains the created record in JSON format in the response body. The response includes only the values of the fields of the created record that were specified during the creation of the record or that were specified to be returned by using the parameters of the request.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
412	You have used the `If-None-Match` header with the `*` value, and the record already exists.

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the creation of a `Customer` record in Acumatica ERP through the REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Customer
Accept: application/json
Content-Type: application/json

{
  "CustomerID" : {"value" : "JOHNGOOD"},
  "CustomerName" : {"value" : "John Good"},
  "CustomerClass" : {"value" : "DEFAULT"},
  "MainContact" :
    {
      "Email" : {"value" : "demo@gmail.com"},
      "Address" :
        {
          "AddressLine1" : {"value" : "4030 Lake Washington Blvd NE"},
          "AddressLine2" : {"value" : "Suite 100"},
          "City" : {"value" : "Kirkland"},
          "State" : {"value" : "WA" },
          "PostalCode" : {"value" : "98033"}
        }
    }
}
```

Usage Notes

Note the following about creation of records through the REST API:

- You can create a record with multiple detail lines by using a single PUT request.
- If you need to create or update documents with a large number of detail lines by using the contract-based REST API, note that the following approaches may have performance implications:
  - You make a single request that contains all information you need to pass. In this case, the operation could time out.
  - You make many requests, each of which contains a single detail line you need to add or alter. In this case, the whole task takes a lot of time.

  A more balanced approach is a compromise between these two: You make multiple requests, each of which contains a part of the detail lines you need to add or alter. You select the number of detail lines in a single request to optimize the performance of the whole task. For example, if you need to send a sales order containing 10,000 detail lines to the server, you can make requests, each of which contains 500 detail lines.
- You can specify a value for a drop-down list (or a multiselect drop-down list) only if that value has been defined for the control. If the value you attempt to set is not present in the control, an error message is returned in the `error` field of the response. For a multiselect drop-down list, only internal (namely, not external or displayed) values can be specified.

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

  To specify the values of custom and user-defined fields through the contract-based REST API, you specify the values of these fields in the body of the `PUT` request, as shown in *Create a Record with Custom Fields*.
- For some text boxes on Acumatica ERP forms, users can type values in multiple languages if multiple locales have been configured in Acumatica ERP. For details about how to specify the values of multi-language fields in REST API requests, see *Specify Any Number of Localized Values of a Multilingual Field* and *Specify All Localized Values of a Multilingual Field*.

Update a Record

When you need to update an existing record by using the contract-based REST API, you access the needed URL with the `PUT` HTTP method and pass the record representation in JSON format in the request body. See the following sections for details on the request and the response.

HTTP Method and URL

When you update a record in Acumatica ERP, you use the `PUT` HTTP method and the following URL.

```
PUT http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to create or update a record.

For example, suppose that you want to update a stock item record in a local Acumatica ERP instance with the name *AcumaticaDB* by using a system endpoint with the name *Default* and Version 24.200.001. You would use the following URL to update a record: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

You can use the following parameters when you are updating a record in Acumatica ERP:

- *$filter*: To specify the filtering conditions that identify the record to be updated. For details about this parameter, see *$filter Parameter*.
- *$expand*: To specify the linked and detail entities to be expanded. For more information, see *$expand Parameter*.

  

  You must list in the *$expand* parameter every detail and related entity that you are going to have in the response body.
- *$select*: To specify the fields of the entity to be returned. For more information about the parameter, see *$select Parameter*.
- *$custom*: To specify the fields that are not defined in the contract to be returned. For details about the parameter, see *$custom Parameter*.

HTTP Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.
`If-Match`	Changes the behavior of the `PUT` request, which normally either creates a new record or updates an existing one. If you only want to update a record, use the optional `If-Match` header with the `*` value.

HTTP Body

You pass a record in JSON format in the request body. You can find details on how to represent a record in JSON format in *Representation of a Record in JSON Format*.

To make it possible for the record to be found by Acumatica ERP, you can specify any of the following:

- The values of the key fields in the record representation in JSON format.
- The value of the `ID` property in the record representation in JSON format.
- The filtering conditions that identify the record in the *$filter* parameter of the method. For details on the parameter, see the *Parameters* section in this topic.

If you want to delete a detail line during update, you should specify `true` as the value of the `delete` property of the corresponding detail entity: `"delete" : true`. To identify the detail line to be deleted, you can specify one of the following:

- The value of the `ID` property of the detail line
- The values of the key fields of the detail line

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that updates a record.

Code	Description
200	The request has been completed successfully. The response body contains the updated record in JSON format.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
412	You have used the `If-Match` header with the `*` value, and the record does not exist.

Code	Description

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the update of an existing customer record that has the *demo@gmail.com* email address.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$filter=MainContact/Email%20eq%20'demo@gmail.com'&
   $select=CustomerID,CustomerClass,MainContact/Email&$expand=MainContact HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Customer
Accept: application/json
Content-Type: application/json

{
  "CustomerClass" : {"value" : "ECCUSTOMER"}
}
```

Usage Notes

Note the following about update of a record:

- You can specify a value for a drop-down list (or a multiselect drop-down list) only if that value has been defined for the control. If the value you attempt to set is not present in the control, an error message is returned in the `error` field of the response. For a multiselect drop-down list, only internal (namely, not external or displayed) values can be specified.
- By using the contract-based REST API, you can remove records from Acumatica ERP. For details about how to remove a record, see *Remove a Record by Key Fields* and *Remove a Record by ID*.
- If you need to create or update documents with a large number of detail lines by using the contract-based REST API, note that the following approaches may have performance implications:
  - You make a single request that contains all information you need to pass. In this case, the operation could time out.
  - You make many requests, each of which contains a single detail line you need to add or alter. In this case, the whole task takes a lot of time.

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

Retrieve a Record by Key Fields

To retrieve a record by the values of its key fields from Acumatica ERP by using the contract-based REST API, you access the needed URL with the `GET` HTTP method and specify the fields that should be returned in the parameters of the method. See the following sections for details on the request and the response.

HTTP Method and URL

If you need to obtain a particular record with the known key fields, you use the `GET` HTTP method and the following URL.

```
http://<Base endpoint URL>/<Top-level entity>/<Key value 1>/<Key value 2>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to retrieve a record.
- *<Key value 1>* and *<Key value 2>* are the values of the key fields of the record to be retrieved. You use the number and order of key fields as they are defined on the corresponding Acumatica ERP form.

  

  You can pass the key fields separated by a vertical bar (|) instead of a slash (/). If the value of a key field contains a slash, you need to refer to a record by its entity ID instead of the values of its key fields.

For example, suppose that you want to retrieve the sales order with order type *SO* and order number *000123* from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to retrieve the sales order: *http://localhost/AcumaticaDB/entity/Default/24.200.001/SalesOrder/SO/000123*.

Parameters

You can use the following parameters when you retrieve a record from Acumatica ERP:

- *$expand*: To specify the linked and detail entities to be expanded. For more information, see *$expand Parameter*.

  

  You must list in the *$expand* parameter every detail and related entity that you are going to have in the response body.

- *$custom*: To specify the fields that are not defined in the contract to be returned. For details about the parameter, see *$custom Parameter*.
- *$select*: To specify the fields of the entity to be returned. For more information about the parameter, see *$select Parameter*.

For detailed descriptions of the parameters, see *Parameters for Retrieving Records*.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves a record.

Code	Description
200	The request has been completed successfully. The response body contains the requested record.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the retrieval of a `SalesOrder` record through the REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /SO/000001?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Retrieve a Record by ID

To retrieve a record by the value of the entity ID from Acumatica ERP by using the contract-based REST API, you access the needed URL with the `GET` HTTP method and specify the fields that should be returned in the parameters of the method. See the following sections for details on the request and the response.

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

Any REST API response contains the `_links/self` field for every top-level entity it returns. The value of this field is part of the URL of a `GET` request to retrieve this top-level entity by its ID. The same URL can be used in a `DELETE` request to remove this entity.

HTTP Method and URL

If you need to obtain a particular record with the entity ID, you use the `GET` HTTP method and the following URL.

```
http://<Base endpoint URL>/<Top-level entity>/<Entity ID>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- `<Top-level entity>` is the name of the entity for which you are going to retrieve a record.
- `<Entity ID>` is the ID of the record to be retrieved.

For example, suppose that you want to retrieve the sales order with entity ID *a6295b33-c7f6-e811-b817-00155d408001* from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to retrieve the sales order.

```
http://localhost/AcumaticaDB/entity/Default/24.200.001/SalesOrder/a6295b33-c7f6-e811-
b817-00155d408001
```

Parameters

You can use the following parameters when you retrieve a record from Acumatica ERP:

- *$expand*: To specify the linked and detail entities to be expanded. For more information, see *$expand Parameter*.

  > You must list in the *$expand* parameter every detail and related entity that you are going to have in the response body.

- *$select*: To specify the fields of the entity to be returned. For more information about the parameter, see *$select Parameter*.
- *$custom*: To specify the fields that are not defined in the contract to be returned. For details about the parameter, see *$custom Parameter*.

For detailed descriptions of the parameters, see *Parameters for Retrieving Records*.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves a record.

Code	Description
200	The request has been completed successfully. The response body contains the requested record.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the retrieval of a `SalesOrder` record with detail records by the entity ID through the REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /a6295b33-c7f6-e811-b817-00155d408001?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Retrieve Records by Conditions

To retrieve the records that satisfy the specified conditions from Acumatica ERP by using the contract-based REST API, you access the needed URL address with the `GET` HTTP method; if necessary, you also specify filtering conditions in the parameters of the method. See the following sections for details on the request and the response.

HTTP Method and URL

If you need to retrieve the list of records, you use the `GET` HTTP method and the following URL.

```
GET http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to retrieve the list of records.

For example, suppose that you want to retrieve the list of stock item records from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to retrieve the list of records: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

You can use the following parameters when you retrieve records from Acumatica ERP:

- *$filter*: To specify the filtering conditions that identify the record to be updated. For details about this parameter, see *$filter Parameter*.
- *$skip*: To specify the number of records to be skipped from the list of returned records. For more information about the parameter, see *$skip Parameter*.
- *$top*: To specify the number of records to be returned in the list. For details about the parameter, see *$top Parameter*.
- *$expand*: To specify the linked and detail entities to be expanded. For more information, see *$expand Parameter*.

  

  You must list in the *$expand* parameter every detail and related entity that you are going to have in the response body.

- *$select*: To specify the fields of the entity to be returned. For more information about the parameter, see *$select Parameter*.
- *$custom*: To specify the fields that are not defined in the contract to be returned. For details about the parameter, see *$custom Parameter*.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves records by conditions.

Code	Description
200	The request has been completed successfully. The response body contains the list of records that satisfy the specified conditions.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the retrieval of `StockItem` records that are *Active* and that have been modified since the specified date.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /entity/Default/24.200.001/StockItem?
    $expand=WarehouseDetails&
    $filter=ItemStatus%20eq%20'Active'%20and
        %20LastModified%20gt%20datetimeoffset'2024-08-18T23%3A59%3A59.999%2B04%3A00'
    HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json
```

Usage Notes

When multiple records are retrieved from Acumatica ERP through an endpoint with Contract Version 4, the system tries to optimize the retrieval of the records and obtain all needed records in one request to the database (instead of requesting the records one by one). If the optimization fails, the system returns an error, which specifies the entities or fields that caused the failure of the optimized request. To prevent the error from occurring, you can do any of the following:

- If you do not need to retrieve the entities or fields that caused the failure, you can exclude these entities or fields from the request as follows:
  - Exclude the entities from the entities specified in the *$expand* parameter
  - Explicitly specify the other fields to be returned (while excluding the fields that caused the failure) by using the *$select* parameter
- If you need to retrieve the entities or fields that caused the failure, you can retrieve the needed records one by one, either by key fields or by IDs. For more information, see *Retrieve a Record by Key Fields* and *Retrieve a Record by ID*.

Retrieve Records Filtered by Custom Fields

If you use the REST API, to retrieve records that satisfy the specified conditions for custom fields from Acumatica ERP, you access the needed URL with the `GET` HTTP method and specify filtering conditions in the parameters of the method. See the following sections for details on the request and the response.

HTTP Method and URL

If you need to retrieve the list of records, you use the `GET` HTTP method and the following URL.

```
GET http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to retrieve the list of records.

For example, suppose that you want to retrieve the list of stock item records from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to retrieve the list of records: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

To retrieve records filtered by custom fields, you use the *$filter* parameter.

You can use the following custom function to filter records by the values of custom fields: `cf.<Type name>(f='<View name>.<Field name>')`, where `<Type name>` is the type of the custom element, `<View name>` is the name of the data view that contains the element, and `<Field name>` is the name of the element.

For more information on the use of the *$filter* parameter, see *$filter Parameter*.

For other parameters that can be used in the request, see *Retrieve Records by Conditions*.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves records by conditions.

Code	Description
200	The request has been completed successfully. The response body contains the list of records that satisfy the specified conditions.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the retrieval of `SalesInvoice` records for which `CuryBalanceWOTotal` is *0* and `DiscDate` is later than February 18, 2024.

> The `CuryBalanceWOTotal` field corresponds to the **Write-Off Total** box on the right pane of the **Applications** tab of the *Invoices* (SO303000) form. The `DiscDate` field corresponds to the **Cash Discount Date** box in the Summary area of the *Invoices* form.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET $filter=cf.Decimal(f='Document.CuryBalanceWOTotal') eq 0M and
cf.DateTime(f='Document.DiscDate') gt datetimeoffset'2024-02-18T23%3A59%3A59.999%2B04%3A00'
  HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json
```

Retrieve Data from an Inquiry Form

To retrieve data from an inquiry form of Acumatica ERP by using the contract-based REST API, you access the needed URL with the `PUT` HTTP method and pass the parameters of the inquiry in JSON format in the request body. See the following sections for details on the request and the response.

HTTP Method and URL

If you need to retrieve data from an inquiry form, you use the `PUT` HTTP method and the following URL.

```
PUT http://<Base endpoint URL>/<Top-level entity>
```

> The `PUT` method is always used to retrieve data from an entity that is mapped to a generic inquiry. This is because generic inquiries can contain parameters whose values you may need to specify by passing their values in the body of the request.

The URL includes the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity that corresponds to the generic inquiry form you are going to retrieve data from.

For example, suppose that you want to retrieve data from the *Inventory Summary* (IN401000) form in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the *Default* name and Version 24.200.001. You would use the following URL to retrieve data: *http://localhost/AcumaticaDB/entity/Default/24.200.001/InventorySummaryInquiry*.

Parameter

When you are retrieving data from an inquiry form, you have to use the *$expand* parameter to expand the detail entity, which contains the results of the inquiry. For a detailed description of the parameter, see *Parameters for Retrieving Records*.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.

Request Body

You pass parameters of the inquiry in JSON format in the request body. The following code shows an example of the representation of parameters of the *Inventory Summary* (IN401000) inquiry form in JSON format.

```
{
  "InventoryID" : {"value" : "APJAM08" } ,
  "WarehouseID" : {"value" : "RETAIL" }
}
```

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves data from an inquiry form.

Code	Description
200	The request has been completed successfully. The response body contains the data retrieved from the inquiry form.

Code	Description
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the retrieval of data from the *Inventory Summary* (IN401000) inquiry form.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT /entity/Default/24.200.001/InventorySummaryInquiry?
    $expand=Results HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

{
  "InventoryID" : {"value" : "SIMCARD" } ,
  "WarehouseID" : {"value" : "YOGI" }
}
```

Example: Custom Generic Inquiry

The following request shows an example of the retrieval of data from the Item Availability Data (INGI0002) custom generic inquiry.

No entity is mapped to the custom generic inquiry in the system endpoints, which are provided in Acumatica ERP by default. Thus, to export data from this generic inquiry by using the contract-based API, you need to use a custom endpoint or an extension of an existing endpoint. In this example, the *ItemAvailabilityData/0001* custom endpoint is used.

```
PUT /entity/ItemAvailabilityData/0001/ItemAvailabilityDataInquiry?
    $expand=ItemAvailabilityDataInquiryDetails HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

{}
```

Retrieve the List of Records in Batches

To retrieve a large number of records of the same type by using the contract-based REST API, you can use several approaches. There are two extremes in performing this task:

- You make a single request that retrieves the whole information you need. In this case there is a risk of an operation timeout.
- You make many requests, each of which retrieves a single record you need. In this case the whole task takes a lot of time.

A balanced approach combines these two extremes: you make multiple requests with the *$top* and *$skip* parameters, each of which retrieves a part of the records you need. The number of records to retrieve in a single request you select empirically to optimize the performance of the whole task or to paginate records.

HTTP Method and URL

If you need to retrieve the list of records, you use the `GET` HTTP method and the following URL.

```
GET http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to retrieve the list of records.

For example, suppose that you want to retrieve the list of stock item records from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to retrieve the list of records: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

For the information on the request parameters, see *$top Parameter* and *$skip Parameter*.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves records by conditions.

Code	Description
200	The request has been completed successfully. The response body contains the list of records that satisfy the specified conditions.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

Suppose that you want to retrieve the sales orders in batches of five. The following example shows how to retrieve the second batch of five sales orders.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$top=5&$skip=5 HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Retrieve Records with Attributes

In Acumatica ERP, some records have attributes that are visible on the **Attributes** tab of the data entry form. For details about attributes, see *Attributes*.

You may have to retrieve the records along with their attributes by using the contract-based REST API. To do this, you access the needed URL with the `GET` HTTP method, and in the parameters of the request, you specify the desired record fields and the fields that correspond to the needed attributes and attribute values. See the following sections for details on the request and the response.

HTTP Method and URL

If you need to retrieve the list of records, you use the `GET` HTTP method and the following URL.

```
GET http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.

- *<Top-level entity>* is the name of the entity for which you are going to retrieve the list of records.

For example, suppose that you want to retrieve the list of stock item records from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to retrieve the list of records: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

When you retrieve records along with their attributes, you should use the following parameters:

- *$select*: To list the desired record fields and the fields that correspond to the needed attributes and attribute values
- *$expand*: To expand the detail entity that contains the fields that correspond to the attributes and attribute values

For example, for the `Contact` entity, to retrieve the values of attributes of records, you need to specify `Attributes/AttributeID` and `Attributes/Value` (or `Attributes/ValueDescription`) in the *$select* parameter, and `Attributes` in the *$expand* parameter. To learn the exact names of the fields that correspond to the attributes for a particular entity, you should review the fields of this entity on the *Web Service Endpoints* (SM207060) form or refer to the OpenAPI specification of the endpoint.

You can also use other parameters that are mentioned in *Retrieve Records by Conditions*. For detailed descriptions of the parameters, see *Parameters for Retrieving Records*.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves records along with their attributes.

Code	Description
200	The request has been completed successfully. The response body contains the data retrieved from the Acumatica ERP instance in JSON format.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request is an example of the retrieval of the `FirstName`, `LastName`, and `JobTitle` fields of contacts (that is, the `Contact` top-level entity), and the `AttributeID` and `Value` fields of the contacts' attributes.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$expand=Attributes&$select=FirstName,LastName,JobTitle,Attributes/
AttributeID,Attributes/Value HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Contact
Accept: application/json
```

Following is an example of a response.

```
[
    ...
    {
        "id": "6a47c888-0b10-e911-9fbe-7c5cf8918e20",
        "rowNumber": 20,
        "note": {
            "value": ""
        },
        "Attributes": [
            {
                "id": "dfe79fb3-b6a0-43a9-afa2-6641b943424d",
                "rowNumber": 1,
                "note": null,
                "AttributeID": {
                    "value": "INTEREST"
                },
                "Value": {},
                "custom": {}
            }
        ],
        "FirstName": {
            "value": "Stephane"
        },
        "JobTitle": {
            "value": "Customer Service & Network Manager"
        },
        "LastName": {
            "value": "Sans"
        },
        "custom": {}
    },
    ...
]
```

Usage Notes

The attributes of top-level entities are exposed in the `AttributeValue` entities. An `AttributeValue` entity has the following fields:

- `AttributeID`: The attribute identifier. Both internal values and external values can be used to set this field, but only the internal value can be retrieved.
- `AttributeDescription`: The external value of the attribute identifier. The field is read-only.
- `Value`: The attribute value. When the value is retrieved, the internal value is returned. To set the value, the internal value and the external value (for control types other than multiselect combo boxes) can be used. For multiselect combo boxes, an external value can be accepted only if it contains a single value without commas. For various control types, the following rules apply to the `Value` field:
  - For check boxes, *0* is returned if the check box is not selected, and *1* is returned if the check box is selected. For setting a value, *0*, *1*, *false* (case-insensitive), or *true* (case-insensitive) can be used.
  - For multiselect combo boxes, values separated by commas (namely, *Value1,Value2,Value3*) compose the internal value.
  - For selectors, the values are set and retrieved in the same way as they are for text boxes.
  - For date edit boxes, the internal values must be parsable through the use of the `System.Globalization.CultureInfo.InvariantCulture` Microsoft .NET object.
  - For combo boxes of all types, date edit boxes, and check boxes, an error message is written to the `error` field when an attempt is made to set an unsupported value.
- `ValueDescription`: The external value of the attribute. The field is read-only. The external value is based on the control type as follows:
  - For text boxes and date edit boxes, the external value is the same as the internal value.
  - For check boxes, the external value can be either *True* or *False*.
  - For combo boxes, the label is used as the external value.
  - For multiselect combo boxes, labels separated by commas and spaces after commas (namely, *Label1, Label2, Label3*) compose the external value.
- `Required`: An indicator of whether the attribute is mandatory. This field is read-only.
- `RefNoteID`: The value of the `NoteID` field of the object to which this attribute is referred. This field is read-only.

Retrieve Archived Records

In Acumatica ERP, you can archive records for particular entities. When you use the contract-based REST API to retrieve records, archived records are hidden by default. To make archived records visible, in the HTTP requests you make, you use the custom `PX-ApiArchive` HTTP header and the *SHOW* value. See the following sections for details on the request and the response.

The updating of an archived record can lead to unpredictable consequences.

HTTP Method and URL

If you need to retrieve the list of records, you use the `GET` HTTP method and the following URL.

```
GET http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to retrieve the list of records.

For example, suppose that you want to retrieve the list of stock item records from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to retrieve the list of records: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

For information on the request parameters, see the *Parameters* section of *Retrieve Records by Conditions*.

Request Headers

You can use the `PX-ApiArchive` optional header and set its value to *SHOW* to make archived records visible for REST API requests. For other possible headers, see *Retrieve Records by Conditions*.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves records by conditions.

Code	Description
200	The request has been completed successfully. The response body contains the list of records that satisfy the specified conditions.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request is an example of the retrieval of shipments (including archived ones) that were created before December 31, 2024.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$filter=ShipmentDate lt datetimeoffset'2024-12-31' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json
```

```
PX-ApiArchive: SHOW
```

Remove a Record by Key Fields

In the contract-based REST API, you can remove a record by the values of its key fields. To remove a record from Acumatica ERP, you access the needed URL address with the `DELETE` HTTP method. See the following sections for details on the request and the response.

> If you need to remove a detail line of a record, you should use the `PUT` HTTP method, as described in *Update a Record*.

HTTP Method and URL

If you need to delete a record with known key fields, you use the `DELETE` HTTP method and the following URL.

```
DELETE http://<Base endpoint URL>/<Top-level entity>/<Key value 1>/<Key value 2>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to remove a record.
- *<Key value 1>* and *<Key value 2>* are the values of the key fields of the record to be removed. You use the number and order of key fields as they are defined on the corresponding Acumatica ERP form.

  > You can pass the key fields separated by a vertical bar (|) instead of a slash (/). If the value of a key field contains a slash, you need to refer to a record by its entity ID instead of the values of its key fields.

For example, suppose that you want to remove the sales order with order type *SO* and order number *000123* from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to remove the sales order: *http://localhost/AcumaticaDB/entity/Default/24.200.001/SalesOrder/SO/000123*.

Parameters

You use no parameters when you remove a record.

Request Headers

You do not specify any header in the request.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that removes the record by the values of its key fields or by its entity ID.

Code	Description
204	The request has been completed successfully. The record is removed.

Code	Description
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the removal of the *CGFEEDER* stock item by the value of the key field.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
DELETE /CGFEEDER HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json
```

Remove a Record by ID

In the contract-based REST API, you can remove a record by its session identifier. To remove a record from Acumatica ERP, you access the needed URL address with the `DELETE` HTTP method. See the following sections for details on the request and the response.

Any REST API response contains the `_links/self` field for every top-level entity it returns. The value of this field is part of the URL of a `GET` request to retrieve this top-level entity by its ID. The same URL can be used in a `DELETE` request to remove this entity.

If you need to remove a detail line of a record, you should use the `PUT` HTTP method, as described in *Update a Record*.

HTTP Method and URL

If you need to remove a record with a known entity ID, you use the `DELETE` HTTP method and the following URL. For details about entity IDs, see *Retrieve a Record by ID*.

```
DELETE http://<Base endpoint URL>/<Top-level entity>/<Entity ID>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.

- *<Top-level entity>* is the name of the entity for which you are going to remove a record.
- *<Entity ID>* is the ID of the record to be removed. Entity ID is a GUID that is assigned to each entity you work with during an Acumatica ERP session. You can obtain the value of the entity ID from the `ID` property of an entity returned from Acumatica ERP.

For example, suppose that you want to remove the sales order with entity ID *03efa858-2351-4bd5-ae06-3d9fb3b3c1e6* from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to remove the sales order: *http://localhost/AcumaticaDB/entity/Default/24.200.001/SalesOrder/03efa858-2351-4bd5-ae06-3d9fb3b3c1e6*.

Parameters

You use no parameters when you remove a record.

Request Headers

You do not specify any header in the request.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that removes a record by its entity ID.

Code	Description
204	The request has been completed successfully. The record is removed.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the removal of the sales order with the ID *286F2AF0-21F5-EB11-9DF1-9828A61840C3*.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
DELETE /286F2AF0-21F5-EB11-9DF1-9828A61840C3 HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Execute an Action That Is Present in an Endpoint

To perform an action that is present in an endpoint by using the contract-based REST API, you access the needed URL with the `POST` HTTP method; you then pass the record representation in JSON format and the parameters of the action in the request body.

See the following sections for details on the request and the response.

HTTP Method and URL

If you need to perform an action on an Acumatica ERP form, you use the `POST` HTTP method and the following URL.

```
http://<Base endpoint URL>/<Top-level entity>/<Action name>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to perform an action.
- *<Action name>* is the name of the action that you are going to perform.

For example, suppose that you want to confirm a shipment in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You would use the following URL to confirm a shipment: *http://localhost/AcumaticaDB/entity/Default/24.200.001/Shipment/ConfirmShipment*.

Parameters

You use no parameters of the request when you execute an action by using the REST API.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.

Request Body

You pass the record to which the action should be applied and the parameters of the action in the request body in JSON format as follows.

```
{
  "entity" : <record in JSON format>,
  "parameters" : <parameters in JSON format>
  }
```

You can find details on how to represent a record in JSON format in *Representation of a Record in JSON Format*.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that performs an action.

Code	Description
202	The operation is in progress. The `Location` header of the response contains the URL that you can use to check the status of the operation by using the `GET` HTTP method. When the `GET` HTTP method with this URL returns *204 No Content*, the operation is completed.
204	The operation that has been initiated by the action has completed or was not created.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
404	An action with this name does not exist.

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Examples

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

The following request shows an example of a *Completed* sales order being reopened through the REST API.

```
POST /ReopenSalesOrder HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
 "entity" :
 {
```

```
  "OrderType" : {"value" : "SO"},
  "OrderNbr" : {"value" : "000001"}
 },
 "parameters" :
 {}
}
```

The following request shows an example of the ID of the *CANDYY* business account being changed to *CANDYYY* through the REST API.

```
POST /ChangeBusinessAccountID HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/BusinessAccount
Accept: application/json
Content-Type: application/json

{
  "entity" : {
    "BusinessAccountID": { "value": "CANDYY" }
  },
  "parameters" : {
    "BusinessAccountID": { "value": "CANDYYY" }
  }
}
```

Usage Notes

Note the following about execution of an action:

- You can execute actions that are not present in the endpoint. These actions can be defined in the forms' graphs or in customization projects.

  For details about execution of custom actions, see *Execute a Custom Action*.
- A processing form contains a Selection area with selection criteria and a table with the records that meet the selection criteria. It also has a form toolbar with two main buttons, which are usually named **Process** and **Process All**. You execute the action associated with either button by using the contract-based REST API in two stages:
  a. You run a request to retrieve the records for processing that meet the desired selection criteria. (For details about this action, see *Narrow the List of Records on a Processing Form*.)
  b. In the body of the second request, you use the response that you received in the first stage to process the records that you have specified (see *Execute a Processing Action for Selected Records*) or process all filtered records (see *Execute a Processing Action for All Filtered Records*).

Execute a Custom Action

You can execute actions that are not present in the endpoint. These actions can be defined in the forms' graphs or in customization projects.

To perform an action by using the contract-based REST API, you access the needed URL with the `POST` HTTP method and pass the record representation in JSON format and the parameters of the action in the request body. See the following sections for details on the request and the response.

HTTP Method and URL

If you need to perform an action on an Acumatica ERP form, you use the `POST` HTTP method and the following URL.

```
http://<Base endpoint URL>/<Top-level entity>/<Action name>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to perform an action.
- *<Action name>* is the name of the action that you are going to perform. If an action is defined for a UI element, you can find out the action name as follows: on the title bar of the form, you click **Customization > Inspect Element** and click the needed element on the form. In the **Element Properties** dialog box, which opens, you find the action name in the **Action Name** element.

For example, suppose that you want to invoke the `Close` action of some *Case* entity in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to invoke the action: *http://localhost/AcumaticaDB/entity/Default/24.200.001/Case/Close*.

Parameters

You use no parameters when you execute an action.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.

HTTP Body

For the invocation of a custom action, an HTTP body with the following pattern is used.

```
{
    "entity": {
        "id": "<entity id>"
    },
    "parameters": {
        "custom": {
            ...
        }
    }
}
```

In this pattern, the following information is provided:

- `<entity id>` is the identifier of the entity for which you need to invoke a custom action. For more information, see *Retrieve a Record by ID*. You can also use the key fields to define the entity.
- The `custom` collection contains the parameters of the action that are normally entered in a dialog box that the system displays upon the execution of the action. You need to specify the parameters' names and the view in which the parameters are defined. You can find out this information as follows: Execute the action from the UI, press Ctrl+Alt, and click the parameter's input box. In the **Element Properties** dialog box, which opens, you find the parameter name in the **Data Field** element and the view name in the **View Name** element.

You can find details on how to represent a record in JSON format in *Representation of a Record in JSON Format*.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that performs an action.

Code	Description
202	The operation is in progress. The `Location` header of the response contains the URL that you can use to check the status of the operation by using the `GET` HTTP method. When the `GET` HTTP method with this URL returns *204 No Content*, the operation is completed.
204	The operation that has been initiated by the action has completed or was not created.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
404	The action with this name does not exist.

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

For example, suppose that you want to invoke the `Close` action of some *Case* entity by using the system endpoint with the name *Default* and Version 24.200.001. The `Close` action has the only *Reason* parameter, which is defined in the *FilterPreview* view. Following is an example of a request.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Case/Close
Accept: application/json
Content-Type: application/json

{
    "entity": {
        "id": "e3f46a39-1a14-e911-816f-bc920a5e0ac8"
    },
    "parameters": {
        "custom": {
            "FilterPreview": {
                "Reason": {
                    "type" : "CustomStringField",
                    "value" : "Abandoned"
                }
            }
        }
    }
}
```

Narrow the List of Records on a Processing Form

A processing form contains a Selection area with selection criteria and a table with the records that meet the selection criteria. It also has a form toolbar with two main buttons, which are usually named **Process** and **Process All**. You execute the action associated with either button by using the contract-based REST API in two stages:

1. You run a request to retrieve the records for processing that meet the desired selection criteria. (This topic provides information on this action.)
2. In the body of the second request, you use the response that you received in the first stage to process the records that you have specified (see *Execute a Processing Action for Selected Records*) or process all filtered records (see *Execute a Processing Action for All Filtered Records*).

You must run both requests during the same session.

HTTP Method and URL

You narrow the list of records to process on a processing form by specifying selection criteria in the Selection area. To specify a filter by using the contract-based REST API, you use the `PUT` HTTP method and the following URL.

```
PUT http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.

- *<Top-level entity>* is the name of the top-level entity that corresponds to the processing form.

For example, suppose that you want to filter records by applying selection criteria for the *Emails Pending Processing* (SM507000) form in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to specify the selection criteria: *http://localhost/AcumaticaDB/entity/Default/24.200.001/EmailProcessing*.

Parameters

You use the *$expand* parameter to retrieve the records to process. As its value, you specify the name of the only nested entity that is defined for the top-level entity. For example, for the `EmailProcessing` entity, you use the following parameter string: *$expand=Result*.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.
`If-None-Match`	Changes the behavior of the `PUT` request, which normally either creates a new record or updates an existing one. If you only want to create a new record, use the optional `If-None-Match` header with the `*` (asterisk) value.

Request Body

For information on the request body, see *Create a Record*.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that creates a record.

Code	Description
200	The request has been completed successfully. The response of a successful method call contains the created record in JSON format in the response body. The response includes only the values of the fields of the created record that were specified during the creation of the record or that were specified to be returned by using the parameters of the request.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
412	You have used the `If-None-Match` header with the `*` value, and the record already exists.

Code	Description

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Response Body

After the request succeeds, the response body contains the same filter settings as the request body does, along with the nested entity filled with the records that can be processed. Each record that can be processed contains the `Selected` field, which is used when executing the **Process** command (see *Execute a Processing Action for Selected Records*).

Example

For example, suppose that you want to filter out both incoming and outgoing pending emails assigned to the current user in an Acumatica ERP instance. On the UI, you use the *Emails Pending Processing* (SM507000) form for this purpose. This form is mapped to the `EmailProcessing` entity, which has the `Result` nested entity. Following is an example of a request that applies the selection criteria to the contents of the form.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Result HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/EmailProcessing
Accept: application/json
Content-Type: application/json

{
    "AssignedToMe": {"value": true},
    "Type": {"value": "All"}
}
```

The following code shows an example of the response.

```
{
    "id": "5cfd60d5-c5d6-4ac9-aa03-06ac083ce329",
    "rowNumber": 1,
    "note": null,
    "Account": {},
    "AccountEmailAccountID": {},
    "AssignedToMe": { "value": true },
    "AssignedToOwner": {},
```

```
    "IncludeFailed": { "value": false },
    "Result": [
        {
            "id": "8f5174a5-b312-ea11-b826-00155d408001",
            "rowNumber": 1,
            "note": { "value": "" },
            "EmailAccount": { "value": "System" },
            "From": { "value": "\"System\" <system@sweetlife.con>" },
            "MailStatus": { "value": "Pending Processing" },
            "Owner": {},
            "Selected": {},
            "StartDate": { "value": "2023-12-03T14:59:30.693+03:00" },
            "Subject": { "value": "Welcome to the Company2 system" },
            "To": { "value": "angelo@sweetlife.com" },
            "custom": {},
            "_links": {
                "files:put": ...
            }
        },
        {
            "id": "4d57b6f3-b312-ea11-b826-00155d408001",
            "rowNumber": 2,
            "note": { "value": "" },
            "EmailAccount": { "value": "System" },
            "From": { "value": "\"System\" <system@sweetlife.con>" },
            "MailStatus": { "value": "Pending Processing" },
            "Owner": {},
            "Selected": {},
            "StartDate": { "value": "2023-12-03T14:59:36.15+03:00" },
            "Subject": { "value": "Welcome to the Company2 system" },
            "To": { "value": "perkins@sweetlife.com" },
            "custom": {},
            "_links": {
                "files:put": ...
            }
        },
        ...
    ],
    "Type": { "value": "All" },
    "custom": {}
}
```


Execute a Processing Action for Selected Records

A processing form contains selection criteria, a table with the filtered records (the records that meet the selection criteria), and two main buttons that are usually named **Process** and **Process All**. You execute the **Process** command by using the contract-based REST API in two stages:

1. You run a request to retrieve the records for processing that meet the desired selection criteria (see *Narrow the List of Records on a Processing Form*).
2. In the body of the second request, you use the response that you received in the first stage to execute the **Process** command for the records that you have retrieved. (This topic provides information on the corresponding action.)

You must run both requests during the same session.

HTTP Method and URL

In the table of a processing form, you select the check boxes in the untitled column for the records you want to process; you then click **Process** on the form toolbar of the form. To process selected records by using the contract-based REST API, you use the `POST` HTTP method and the following URL.

```
POST http://<Base endpoint URL>/<Top-level entity>/<Process action name>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the top-level entity that corresponds to the processing form.
- *<Process action name>* is the name of the action to which the **Process** command is mapped in the endpoint.

For example, suppose that you want to execute the **Process** command of the *Emails Pending Processing* (SM507000) form in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to execute the **Process** command: *http://localhost/AcumaticaDB/entity/Default/24.200.001/EmailProcessing/ProcessEmailProcessing*.

Parameters

You use no parameters when you execute an action.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.

Request Body

As a request body, you create a JSON object with the only *Entity* key; as the value, you use the response body received when selection criteria were applied to filter records (see *Narrow the List of Records on a Processing Form*). Further, you change the request body as follows: for each record, in the `Selected` field, you specify either *true* (if the record must be processed) or *false* (if the record must not be processed).

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Header	Description
`Content-Type`	Specifies the format of the request body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that performs an action.

Code	Description
202	The operation is in progress. The `Location` header of the response contains the URL that you can use to check the status of the operation by using the `GET` HTTP method. When the `GET` HTTP method with this URL returns *204 No Content*, the operation is completed.
204	The operation that has been initiated by the action has completed or was not created.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
404	An action with this name does not exist.

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

For example, suppose that you want to process some pending emails in an Acumatica ERP instance. In the UI, you select the desired emails on the *Emails Pending Processing* (SM507000) form and click **Process** for this purpose. This form is mapped to the `EmailProcessing` entity, which has the `ProcessEmailProcessing` action; this action corresponds to the **Process** button.

Following is an example of a request that processes the first email from the list that is retrieved in the example provided in *Narrow the List of Records on a Processing Form*. The `Selected` field of the first record contains *true* and the rest ones contain *false*.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /ProcessEmailProcessing HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/EmailProcessing
Accept: application/json
Content-Type: application/json

{
    "Entity": {
        "id": "5cfd60d5-c5d6-4ac9-aa03-06ac083ce329",
        "rowNumber": 1,
        "note": null,
        "Account": {},
        "AccountEmailAccountID": {},
        "AssignedToMe": { "value": true },
        "AssignedToOwner": {},
        "IncludeFailed": { "value": false },
        "Result": [
            {
                "id": "8f5174a5-b312-ea11-b826-00155d408001",
                "rowNumber": 1,
                "note": { "value": "" },
                "EmailAccount": { "value": "System" },
                "From": { "value": "\"System\" <system@sweetlife.con>" },
                "MailStatus": { "value": "Pending Processing" },
                "Owner": {},
                "Selected": { "value": true },
                "StartDate": { "value": "2023-12-03T14:59:30.693+03:00" },
                "Subject": { "value": "Welcome to the Company2 system" },
                "To": { "value": "angelo@sweetlife.com" },
                "custom": {},
                "_links": {
                    "files:put": ...
                }
            },
            {
                "id": "4d57b6f3-b312-ea11-b826-00155d408001",
                "rowNumber": 2,
                "note": { "value": "" },
                "EmailAccount": { "value": "System" },
                "From": { "value": "\"System\" <system@sweetlife.con>" },
                "MailStatus": { "value": "Pending Processing" },
                "Owner": {},
                "Selected": { "value": false },
                "StartDate": { "value": "2023-12-03T14:59:36.15+03:00" },
                "Subject": { "value": "Welcome to the Company2 system" },
                "To": { "value": "perkins@sweetlife.com" },
                "custom": {},
                "_links": {
                    "files:put": ...
                }
            },
            ...
```

```
        ],
        "Type": { "value": "All" },
        "custom": {}
    }
}
```

Execute a Processing Action for All Filtered Records

A processing form contains selection criteria, a table with the filtered records (the records that meet the selection criteria), and two main buttons that are usually named **Process** and **Process All**. You execute the **Process All** command by using the contract-based REST API in two stages:

1. You run a request to retrieve the records for processing that meet the desired selection criteria (see *Narrow the List of Records on a Processing Form*).
2. In the body of the second request, you use the `id` value from the response that you received in the first stage to execute the **Process All** command, which affects all filtered records. (This topic provides information on the corresponding action.)

You must run both requests during the same session.

HTTP Method and URL

You click **Process All** on the form toolbar of a processing form to process all records contained in the table of this form. To process all listed records by using the contract-based REST API, you use the `POST` HTTP method and the following URL.

```
http://<Base endpoint URL>/<Top-level entity>/<Process All action name>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the top-level entity that corresponds to the processing form.
- *<Process All action name>* is the name of the action to which the **Process All** command is mapped in the endpoint.

For example, suppose that you want to execute the **Process All** command of the *Emails Pending Processing* (SM507000) form in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to execute the **Process** command: *http://localhost/AcumaticaDB/entity/Default/24.200.001/EmailProcessing/ProcessAllEmailProcessing*.

Parameters

You use no parameters when you execute an action.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.

Request Body

As a request body, you create a JSON object with the only *Entity* key; as the value, you use the key-value pair with the *id* key that is taken from the response body received when the selection criteria were applied (see *Narrow the List of Records on a Processing Form*).

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that performs an action.

Code	Description
202	The operation is in progress. The `Location` header of the response contains the URL that you can use to check the status of the operation by using the `GET` HTTP method. When the `GET` HTTP method with this URL returns *204 No Content*, the operation is completed.
204	The operation that has been initiated by the action has completed or was not created.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
404	An action with this name does not exist.

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

For example, suppose that you want to process all incoming and outgoing pending emails assigned to the current user in an Acumatica ERP instance. On the UI, the *Emails Pending Processing* (SM507000)

form can be used for this purpose. This form is mapped to the `EmailProcessing` entity, which has the `ProcessAllEmailProcessing` action; this action corresponds to the **Process All** button.

Following is an example of a request that processes all the pending emails that are retrieved in the example provided in *Narrow the List of Records on a Processing Form*.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /ProcessAllEmailProcessing HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/EmailProcessing
Accept: application/json
Content-Type: application/json

{
    "Entity": {"id": "e269bafb-a6c0-4463-a3ad-e8b3eab203d4"}
}
```

Retrieve a File Attached to a Record

To retrieve a file that is attached to a record from Acumatica ERP by using the contract-based REST API, you access the URL address of the file with the `GET` HTTP method. See the following sections for details on the request and the response.

HTTP Method and URL

If you need to obtain a file attached to a record, you use the `GET` HTTP method and the following URL.

```
GET http://<Base endpoint URL>/files/<File identifier>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<File identifier>* is the internal identifier of the file in the system.

To get this URL for a particular file attached to a record, you should obtain the record from Acumatica ERP and, in the returned JSON representation of the record, find the value of the `href` property of the needed file in the `files` array. For information on how to retrieve a record from Acumatica ERP, see *Retrieve a Record by Key Fields* and *Retrieve a Record by ID*.

For example, suppose that you retrieved the stock item record that contains the following `files` array from a local Acumatica ERP instance with the name *AcumaticaDB*.

```
{
  ...,
  "files":[
    {
      "id":"9be45eb7-f97d-400b-96a5-1c4cf82faa96",
      "filename":"Stock Items (AAMACHINE1)\\T2MCRO.jpg",
      "href":
"/AcumaticaDB/entity/Default/24.200.001/files/9be45eb7-f97d-400b-96a5-1c4cf82faa96"
```

```
    }
  ]
}
```

You should use the following URL to retrieve the `T2MCRO.jpg` file attached to the stock item record: *http://localhost/AcumaticaDB/entity/Default/24.200.001/files/9be45eb7-f97d-400b-96a5-1c4cf82faa96*.

Parameters

You use no parameters when you retrieve a file attached to a record.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves a file attached to a record.

Code	Description
200	The request has been completed successfully. The response contains the requested file.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

The following request shows an example of the retrieval of the list of files attached to the *JUICER20C* stock item.

```
GET /JUICER20C?$select=InventoryID,files&$expand=files HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json
```

Suppose that the request above returns the following response body.

```
{
    "id": "a0f8594a-7de2-e811-b816-00155d408001",
    "rowNumber": 1,
    "note": {
        "value": ""
    },
    "InventoryID": {
        "value": "EJECTOR03"
    },
    "custom": {},
    "files": [
        {
            "id": "0a8ac74e-53d3-4f13-ba02-dbbf419da119",
            "filename": "Stock Items (EJECTOR03   )\\T2MCRO.jpg",
            "href": "[/<Acumatica ERP instance name>]/entity/Default/24.200.001/
                 files/dbcf82e7-660a-4d56-9b64-1cb654660ddb"
        }
    ]
}
{
    "id": "eb01ae69-7ee2-e811-b816-00155d408001",
    "rowNumber": 1,
    "note": {
        "value": ""
    },
    "InventoryID": {
        "value": "JUICER20C"
    },
    "custom": {},
    "_links": {
        "self": "/<Acumatica ERP instance name>/entity/Default/24.200.001/
StockItem/eb01ae69-7ee2-e811-b816-00155d408001",
        "files:put": "/<Acumatica ERP instance name>/entity/Default/24.200.001/
files/PX.Objects.IN.InventoryItemMaint/Item/eb01ae69-7ee2-e811-b816-00155d408001/
{filename}"
    },
    "files": [
        {
            "id": "e77c0c80-2296-4ef9-9588-877692933f31",
            "filename": "Stock Items (JUICER20C   )\\commercial_citrus_juicer.jpg",
            "href": "/<Acumatica ERP instance name>/entity/Default/24.200.001/
files/e77c0c80-2296-4ef9-9588-877692933f31"
        }
    ]
}
```

The following request retrieves the `commercial_citrus_juicer.jpg` file attached to the *JUICER20C* stock item by the ID of the file.

```
GET /e77c0c80-2296-4ef9-9588-877692933f31 HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/files
Accept: application/octet-stream
Content-Type: application/json
```

Attach a File to a Record

When you need to attach a file to a record by using the contract-based REST API, you access the needed URL with the `PUT` HTTP method and pass the file in the request body. See the following sections for details on the request and the response.

HTTP Method and URL

If you need to attach a file to a top-level record or a detail record of any nesting level in Acumatica ERP, you use the `PUT` HTTP method.

To get the URL for attaching a file to a particular record, you should obtain the record from Acumatica ERP; in the returned JSON representation of the record, you find the value of the `_links/files:put` field. For information on how to retrieve a record from Acumatica ERP, see *Retrieve a Record by Key Fields* and *Retrieve a Record by ID*. The value of the `_links/files:put` field for the desired record is a part of the URL template that must be used in a request for attaching a file.

For example, suppose that you have retrieved the stock item record from a local Acumatica ERP instance with the name *AcumaticaDB*. This record contains the following `_links/files:put` field value: */AcumaticaDB/entity/Default/24.200.001/files/PX.Objects.IN.InventoryItemMaint/Item/cae53ce0-1614-e511-9b82-c86000dddf0b/{filename}*. If you want to attach the `file.dat` file to this record, you use the following HTTP method and URL.

```
PUT http://AcumaticaDB/entity/Default/24.200.001/files/PX.Objects.IN.InventoryItemMaint/
Item/cae53ce0-1614-e511-9b82-c86000dddf0b/file.dat
```

For backward compatibility, Acumatica ERP also maintains an old way to attach a file to a top-level record. This way involves using the `PUT` HTTP method and the following URL: `http://<Base endpoint URL>/<Top-level entity>/<Key value 1>/<Key value 2>/files/<File name>`. You address the record to which you are going to attach the *<File name>* file by using the *<Key value 1>* and *<Key value 2>* values of its key fields.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/octet-stream`.

Header	Description
`PX-CbFileComment`	Specifies a comment for the attached file. A user can see this comment in the **Comment** column of the **Files** dialog box. The dialog box opens in any of the following situations: • The user clicks **Files** in the title bar of an Acumatica ERP form that is open to view a record with at least one attached file. • The user clicks Files () at the beginning of a detail row in the table.

Request Body

You pass the file to be attached in the request body.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that attaches a file to a record.

Code	Description
204	The request has been completed successfully. The file is attached. The `Location` header of the response contains the URL that you can use to retrieve the file from the system.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

The following request shows an example of the retrieval of the *EJECTOR03* stock item.

```
GET /EJECTOR03?$select=InventoryID HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json
```

Suppose that the request above returns the following response body.

```
{
```

```
    "id": "a0f8594a-7de2-e811-b816-00155d408001",
    "rowNumber": 1,
    "note": {
        "value": ""
    },
    "InventoryID": {
        "value": "EJECTOR03"
    },
    "custom": {},
    "_links": {
        "self": "/<Acumatica ERP instance name>/entity/Default/24.200.001/
StockItem/a0f8594a-7de2-e811-b816-00155d408001",
        "files:put": "/<Acumatica ERP instance name>/entity/Default/24.200.001/
files/PX.Objects.IN.InventoryItemMaint/Item/a0f8594a-7de2-e811-b816-00155d408001/
{filename}"
    }
}
```

The following request attaches the `T2MCRO.jpg` file to the *EJECTOR03* stock item and specifies *Test comment* as the comment for the file.

```
PUT /a0f8594a-7de2-e811-b816-00155d408001/T2MCRO.jpg HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/files/
    PX.Objects.IN.InventoryItemMaint/Item
Accept: application/json
Content-Type: application/octet-stream
PX-CbFileComment: Test comment

"<file contents here>"
```

Usage Notes

The following usage details apply to comments that you add to a file as described in this topic:

- If a new version of the file is added without a comment, the **Comment** column in the **Files** dialog box becomes empty for the file.
- If multiple `PX-CbFileComment` headers are specified in the request, the **Comment** column contains the comments from all headers. The comments are separated with a comma.
- A comment for the file cannot exceed 500 characters.

Retrieve Comments for Attached Files

You can retrieve comments that have been specified for the files attached to a record from Acumatica ERP by using the contract-based REST API. To do this, you access the needed URL with the `GET` HTTP method and specify the `files` array to be returned. See the following sections for details on the request and the response.

HTTP Method and URL

To retrieve a record with the list of files attached to the record and comments specified for these files, you use the `GET` HTTP method and the same URL as you would use to retrieve a record (see *Retrieve a Record by Key Fields* and *Retrieve a Record by ID*).

Parameters

A comment for a file is returned in the `comment` field for the file item in the `files` array. To retrieve a record with comments for the files attached to the record, you use the *$expand* and *$select* parameters with *files* as the value. For detailed descriptions of the parameters, see *Parameters for Retrieving Records*.

You can also use other parameters that are available for a request to retrieve a record. For details, see *Retrieve a Record by Key Fields* and *Retrieve a Record by ID*.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves a record.

Code	Description
200	The request has been completed successfully. The response body contains the requested record.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following example request retrieves the list of files attached to the *EJECTOR03* stock item.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /EJECTOR03?$select=InventoryID,files&$expand=files HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json
```

The response body, shown below, includes the comment in the `comment` field.

```
{
  "id":"a0f8594a-7de2-e811-b816-00155d408001",
  "rowNumber":1,
  "note":
  {
    "value":""
  },
  "InventoryID":
  {
    "value":"EJECTOR03"
  },
  "custom":{},
  "_links":
  {
    "self":"...",
    "files:put":"..."
  },
  "files":
  [
    {
      "id":"8050623c-b31f-42a1-876a-13598c90fd29",
      "filename":"Stock Items (EJECTOR03   )\\T2MCRO.jpg",
      "href":"...",
      "comment":"Test comment"
    }
  ]
}
```


Retrieve the Schema of Custom Fields

To retrieve the schema of custom fields of an entity—that is, the field name, view name, and type of the fields that are not defined in the contract of the endpoint for this entity—by using the contract-based REST API, you access the needed URL with the `GET` HTTP method. See the following sections for details on the request and the response.

Custom fields can correspond to the following elements:

- The predefined elements on an Acumatica ERP form that are not included in the entity definition
- The elements that were added to the Acumatica ERP form in a customization project
- The user-defined fields

HTTP Method and URL

If you need to obtain the schema of custom fields of an entity, you use the `GET` HTTP method and the following URL.

```
http://<Base endpoint URL>/<Top-level entity>/$adHocSchema
```

The URL includes the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.

- *<Top-level entity>* is the name of the entity for which you are going to retrieve the schema of custom fields.

For example, suppose that you want to obtain the schema of custom fields of a stock item entity from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You would use the following URL to retrieve the schema: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem/$adHocSchema*.

Parameters

You use no parameters when you retrieve the schema of custom fields.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves the schema of the custom fields of an entity.

Code	Description
200	The request has been completed successfully. The response contains the field name, view name, and type of the fields that are not defined in the contract of the endpoint for the entity.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request shows an example of the retrieval of the schema of custom fields of the `StockItem` entity.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /$adHocSchema HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
```

```
Content-Type: application/json
```

Retrieve a Record with Custom Fields

To retrieve a record with custom fields by using the contract-based REST API, you access the needed URL with the `GET` HTTP method. For more information on retrieval of records, see *Retrieve a Record by Key Fields*, *Retrieve a Record by ID*, *Retrieve Records by Conditions*.

HTTP Method and URL

To retrieve a record with custom fields, you use the `GET` HTTP method and the same URL as you would use to retrieve a record that contains only those fields that are defined in the endpoint (see *Retrieve a Record by Key Fields*and *Retrieve a Record by ID*).

Parameters

To retrieve a record with custom fields, you use the *$custom* parameter. For more information on the use of the *$custom* parameter, see *$custom Parameter*.

You can also use other parameters that are available for a request for retrieval of a record. For details, see *Retrieve a Record by Key Fields*and *Retrieve a Record by ID*.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves a record.

Code	Description
200	The request has been completed successfully. The response body contains the requested record.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

Suppose that you have added to the *Sales Orders* (SO301000) form the `PRODUCT` user-defined field. The following request shows an example of the retrieval of sales order's fields along with the `PRODUCT` field.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /SO/000087?$custom=Document.AttributePRODUCT HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Related Links

- *$custom Parameter*

Create a Record with Custom Fields

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

To create a record with custom fields by using the contract-based REST API, you access the needed URL with the `PUT` HTTP method. For more information on the creation of records, see *Create a Record*.

HTTP Method and URL

To create a record in Acumatica ERP, you use the `PUT` HTTP method and the following URL.

```
PUT http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to create a record.

For example, suppose that you want to create a stock item record in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to create a record: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

To create a record with custom fields, you use the *$custom* parameter. For more information on the use of the *$custom* parameter, see *$custom Parameter*.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.
`If-None-Match`	Changes the behavior of the `PUT` request, which normally either creates a new record or updates an existing one. If you only want to create a new record, use the optional `If-None-Match` header with the `*` (asterisk) value.

Request Body

To create a record with custom fields, you compose the request body as described in the *Custom Fields* section of *Representation of a Record in JSON Format*.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that creates a record.

Code	Description
200	The request has been completed successfully. The response of a successful method call contains the created record in JSON format in the response body. The response includes only the values of the fields of the created record that were specified during the creation of the record or that were specified to be returned by using the parameters of the request.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
412	You have used the `If-None-Match` header with the `*` value, and the record already exists.

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

Suppose that you have added to the *Sales Orders* (SO301000) form the `PRODUCT` user-defined field. The following request shows an example of creation of a sales order that contains the `PRODUCT` field.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$custom=Document.AttributePRODUCT HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {
        "value": "GOODFOOD"
    },
    "Hold": {
        "value": false
    },
    "LocationID": {
        "value": "MAIN"
    },
    "OrderType": {
        "value": "SO"
    },
    "PaymentMethod": {
        "value": "CHECK"
    },
    "custom": {
        "Document": {
            "AttributePRODUCT": {
                "type": "CustomStringField",
                "value": "Apple jam 8 oz."
            }
        }
    }
}
```

Usage Notes

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

Retrieve Localized Values of a Multilingual Field

For some text boxes on Acumatica ERP forms, users can type values in multiple languages if multiple locales are configured in Acumatica ERP. For example, if your Acumatica ERP instance has the English and French locales

activated and multilingual user input configured, you can specify the value of the **Description** box on the *Stock Items* (IN202500) form in English or French.

In this topic, you can learn how to retrieve localized values of a multilinqual field.

HTTP Method and URL

If you need to retrieve the list of records, you use the `GET` HTTP method and the following URL.

```
GET http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to retrieve the list of records.

For example, suppose that you want to retrieve the list of stock item records from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to retrieve the list of records: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

You retrieve localized values by using the *$custom* parameter, in which you specify the name of a custom field.

To find out the field name and the view name of the needed custom field with localized values, you find out the field name and the view name of the multilingual text box and append *Translations* to the field name. (For details on how to find out the field name and the view name of an element on the form, see *Custom Fields*.) For example, the multilingual **Description** box on the *Stock Items* form has the *Descr* field name and the *Item* view name; therefore, the custom field that contains the localized descriptions of a stock item has the *DescrTranslations* field name and the *Item* view name. So you should use the following parameter string in the request URL: *$custom=Item.DescrTranslations*.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves records by conditions.

Code	Description
200	The request has been completed successfully. The response body contains the list of records that satisfy the specified conditions.
401	The user is not signed in to the system.

Code	Description
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

For example, suppose that you want to retrieve the localized values of the **Description** box on the *Stock Items* form. Following is an example of a request.

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

```
GET ?$custom=Item.DescrTranslations HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json
```

The returned value of a `Translations` custom field is a string with the available localized values of the field. The language to which the value belongs is identified by the two-letter ISO code of the language. For example, suppose that for a particular record, the **Description** element of the *Stock Items* form has the value *Item* in English and *Pièce* in French. In this case, the value of the `DescrTranslations` custom field, which corresponds to the **Description** element, is the following string: `[{en:Item},{fr:Pièce}]`.

Related Links

- *Locales and Languages*
- *Boxes That Have Multilanguage Support*
- *Custom Fields*

Retrieve Localized Values of All Multilingual Fields

For some text boxes on Acumatica ERP forms, users can type values in multiple languages if multiple locales are configured in Acumatica ERP. For example, if your Acumatica ERP instance has the English and French locales activated and multilingual user input configured, you can specify the value of the **Description** box on the *Stock Items* (IN202500) form in English or French.

In this topic, you can learn how to retrieve localized values of all requested multilinqual fields.

HTTP Method and URL

If you need to retrieve the list of records, you use the `GET` HTTP method and the following URL.

```
GET http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to retrieve the list of records.

For example, suppose that you want to retrieve the list of stock item records from a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to retrieve the list of records: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

You retrieve all localized values of all requested fields by adding the *$expand=Translations* parameter to the request.

Request Headers

You specify the following header in the request.

Header	Description
`Accept`	Specifies the format of the response body, which should be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves records by conditions.

Code	Description
200	The request has been completed successfully. The response body contains the list of records that satisfy the specified conditions.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

You can also use the following request to retrieve all localized values on the *Stock Items* form.

```
GET ?$expand=Translations HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json
```

Suppose that the default language is English. If for a particular record, the **Description** element of the *Stock Items* form has the value *Item* in English and *Pièce* in French (as in the previous example), the response has the following data for this record.

```
...
"Description": {
    "value": "Item",
    "translations": {
        "en": "Item",
        "fr": "Pièce"
    }
...
```

Related Links

- *Locales and Languages*
- *Boxes That Have Multilanguage Support*
- *Custom Fields*

Specify Any Number of Localized Values of a Multilingual Field

For some text boxes on Acumatica ERP forms, users can type values in multiple languages if multiple locales are configured in Acumatica ERP. For example, if your Acumatica ERP instance has the English and French locales activated and multilingual user input configured, you can specify the value of the **Description** box on the *Stock Items* (IN202500) form in English or French.

This topic describes a way to specify any number of localized values of a multilingual field. This is the recommended way to do this, as opposed to the way that allows you to specify only all localized values of a multilingual field at once (see *Specify All Localized Values of a Multilingual Field*).

HTTP Method and URL

To specify all localized values of a multilingual field of a record that you create or update in Acumatica ERP, you use the `PUT` HTTP method and the following URL.

```
PUT http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to create or update a record.

For example, suppose that you want to create a stock item record in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to create a record: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

For information on the request parameters, see *Create a Record* or *Update a Record* depending on the operation you are performing (creation or update of a record).

Request Headers

For information on the request headers, see *Create a Record* or *Update a Record* depending on the operation you are performing (creation or update of a record).

Request Body

In the HTTP body, in the `value` field, you specify the value in the default language (see *Setting Up Languages*). You specify the values in other languages in the `Translations/<language_code>` fields. If you specify the values for only some of the configured languages, the field values in other languages remain unchanged.

If multiple localized values are provided in the `Translations` property of a field and in the `value` property of the same field in the format that is described in *Specify All Localized Values of a Multilingual Field*, the request fails with the following error message: *Localized values of a multilingual field cannot be passed in Value and Translations properties.*

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that creates a record.

Code	Description
200	The request has been completed successfully. The response of a successful method call contains the created record in JSON format in the response body. The response includes only the values of the fields of the created record that were specified during the creation of the record or that were specified to be returned by using the parameters of the request.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
412	You have used the `If-None-Match` header with the `*` value, and the record already exists.

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

Suppose that you need to specify values in English (which is set as the default language) and French in the **Description** box on the *Stock Items* (IN202500) form. To do this, you specify the English value of the `Description.value` field of the `StockItem` entity, and the French value in the `Description.Translations.fr` field of the entity. See below for an example of update of the *LAPTOP15* stock item record with localized `Description` field values in JSON format.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$select=InventoryID,Description,Translations&$expand=Translations HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json

{
  "InventoryID" : { "value" : "LAPTOP15" },
  "Description" : {
    "value": "new description in English",
    "Translations": {
      "fr": "nouvelle description en français"
    }
  }
}
```

Related Links

- *Locales and Languages*
- *Boxes That Have Multilanguage Support*
- *Custom Fields*

Specify All Localized Values of a Multilingual Field

For some text boxes on Acumatica ERP forms, users can type values in multiple languages if multiple locales are configured in Acumatica ERP. For example, if your Acumatica ERP instance has the English and French locales activated and multilingual user input configured, you can specify the value of the **Description** box on the *Stock Items* (IN202500) form in English or French.

This topic describes a way to specify all localized values of a multilingual field. However, we recommend that you use another way that allows you to specify any number of localized values of a multilingual field (see *Specify Any Number of Localized Values of a Multilingual Field*).

HTTP Method and URL

To specify all localized values of a multilingual field of a record that you create or update in Acumatica ERP, you use the `PUT` HTTP method and the following URL.

```
PUT http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to create or update a record.

For example, suppose that you want to create a stock item record in a local Acumatica ERP instance with the name *AcumaticaDB* by using the system endpoint with the name *Default* and Version 24.200.001. You should use the following URL to create a record: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

For information on the request parameters, see the *Parameters* section of *Create a Record* and the *Parameters* section of *Update a Record*.

Request Headers

For the information on HTTP headers to use, see *Create a Record* or *Update a Record* depending on the operation you perform (creation or update of a record).

Request Body

In the HTTP body, you specify values in the following format: `[{language_code1:value1}, {language_code2:value2},...]`. As the language code, you use the two-letter ISO code of the language with which the value should be associated.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that creates a record.

Code	Description
200	The request has been completed successfully. The response of a successful method call contains the created record in JSON format in the response body. The response includes only the values of the fields of the created record that were specified during the creation of the record or that were specified to be returned by using the parameters of the request.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
403	The user has insufficient rights to access the Acumatica ERP form that corresponds to the API entity.
412	You have used the `If-None-Match` header with the `*` value, and the record already exists.

422	The data specified in the request is invalid, and the validation errors are returned in the `error` fields of the response body, as shown in the following example.
	`"CustomerID": {`
	`"value": "ABARTENDE1",`
	`"error": "'Customer' cannot be found in the system."`
	`}`

Code	Description
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

In the example that is mentioned at the beginning of the topic, if you need to specify values in English and French in the **Description** box on the *Stock Items* form, you specify the value of the `Description` field of the `StockItem` entity in the following format: `[{en:English description},{fr:French description}]`. See below for an example of a stock item record with localized `Description` field values.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json

{
  "InventoryID" : {"value" : "BASESERV" },
  "ItemClass" : {"value" : "COMPUTERS" },
  "Description" : {"value" : "[{en:Item},{fr:Pièce}]" }
}
```

Usage Notes

In the payload, you should specify the actual values of the field in all languages that are configured for multilingual user input. If you specify the values of the field in particular languages, the values of the field in other languages configured for multilingual user input become empty. For example, suppose that in your instance of Acumatica ERP, multilingual fields can have values in English and French. If you pass the value of a field in the following format `[{en:English description}]`, the French value of the field becomes empty.

If you specify the value of a multilingual field as plain text, this text is saved as the value of the corresponding box in the default language of Acumatica ERP. For details on how to specify the default language, see *Setting Up Languages*.

Related Links

- *Locales and Languages*
- *Boxes That Have Multilanguage Support*
- *Custom Fields*

Request a Report

To request a report by using the contract-based REST API, you access the needed URL with the `POST` HTTP method. You pass the parameters of the report in JSON format in the request body.

See the following sections for details on the request and the response.

HTTP Method and URL

If you need to retrieve a report from Acumatica ERP, you use the `POST` HTTP method and the following URL.

```
POST http://<Base endpoint URL>/<Report entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Report entity>*: The name of the entity that corresponds to the report that you are going to obtain.

For example, suppose that the created custom endpoint has the *http://localhost/AcumaticaDB/entity/Report/0001/* URL and the *CashAccountSummary* report entity. The developer would use the following URL: *http://localhost/AcumaticaDB/entity/Report/0001/CashAccountSummary*.

Parameters

You use no parameters when you request a report from Acumatica ERP.

Request Headers

You can specify the following headers in the request.

Header	Description
`Accept`	Specifies the format in which the report should be returned. The format (and the respective header value) can be one of the following: • PDF: `application/pdf` • HTML: `text/html` • Excel: `application/vnd.openxmlformats-officedocument.spreadsheetml.sheet` If the `Accept` header is not specified, the report is returned in PDF format.
`Content-Type`	Specifies the format of the request body, which should be `application/json`.

Header	Description
`Accept-Language`	Specifies the language in which the report should be returned from Acumatica ERP. The header value can be one of the following: • A locale name that is specified on the *System Locales* (SM200550) form, such as *fr-FR*: If this locale is defined on the form and is active, the report in this language will be returned. If this locale is not defined on the form or is inactive, the active locale with the lowest sequence number that is defined on the form is used for the returned report. • Empty or *: The returned report uses the active locale with the lowest sequence number that is defined on the form. • Multiple locale names that are weighted with the quality value syntax, such as *fr-FR, en;q=0.8, de;q=0.7, *;q=0.5*: To generate the returned report, the system uses the requested locale that is defined on the form, is active, and has the highest priority specified. For more information about quality values, see *https://developer.mozilla.org/en-US/docs/Glossary/Quality_values*.

Request Body

In the request body, you specify the parameters of the report in JSON format, as shown in the following example.

```
{
    "CompanyBranch": {"value": "SOFT"},
    "IncludeNonClearedTransactions": {"value": true}
}
```

If no parameters are specified in the request body, the default parameters of the report are used.

You can find details on how to represent a record in JSON format in *Representation of a Record in JSON Format*.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request for a report.

Code	Description
202	The operation is in progress. The `Location` header of the response contains the URL that you can use to obtain the requested report by using the `GET` HTTP method. When the report is ready, this `GET` request returns the *200 OK* status code. The requested report is returned in the response body.
400	The data specified in the request is invalid.
401	The user is not signed in to the system.
500	An internal server error has occurred.

Response Body

After the request succeeds, the response body is empty.

Example

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

The following request shows an example of a PDF version being requested of the *Cash Account Summary* (CA633000) report through the REST API of the custom *Report/0001* endpoint.

```
POST /entity/Report/0001/CashAccountSummary HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/pdf
Content-Type: application/json

{
    "CompanyBranch": {"value": "SOFT"},
    "IncludeNonClearedTransactions": {"value": true}
}
```

The response of the successful request contains the `Location` header similar to the following: `/<instance name>/entity/Report/0001/CashAccountSummary/report/PDF/7049baf4-9387-4dd4-9eb5-e975cc8d1a96`. When the report is ready, a `GET` request to this URL returns the requested report in the response body.

Change the Business Date or Current Branch

When you use the contract-based REST API, the current date is used as the business date, and the branch that you specified when signing in is used as the current branch. When you create or update a record by using the contract-based REST API, you may need to use other values of the business date or current branch. To do this, you use the custom HTTP headers *PX-CbApiBusinessDate* and *PX-CbApiBranch* in the HTTP requests you make. See the following sections for details on the request and the response.

> The new values that you specify for the business date or current branch are valid only for the current HTTP request.

HTTP Method and URL

When you update a record in Acumatica ERP, you use the `PUT` HTTP method and the following URL.

```
PUT http://<Base endpoint URL>/<Top-level entity>
```

The URL has the following components:

- *<Base endpoint URL>* is the URL of the contract-based endpoint through which you are going to work with Acumatica ERP. This URL has the following format: *http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/*.
- *<Top-level entity>* is the name of the entity for which you are going to create or update a record.

For example, suppose that you want to update a stock item record in a local Acumatica ERP instance with the name *AcumaticaDB* by using a system endpoint with the name *Default* and Version 24.200.001. You would use the following URL to update a record: *http://localhost/AcumaticaDB/entity/Default/24.200.001/StockItem*.

Parameters

For information on the request parameters, see the *Parameters* section of *Create a Record* and the *Parameters* section of *Update a Record*.

Request Headers

You can use the following optional headers for changing the business date or current branch.

Header	Description
`PX-CbApiBusiness-Date`	Optional. Specifies the new business date. The date can be specified in any format. If you omit this header, the current date is used as the business date.
`PX-CbApiBranch`	Optional. Specifies the new current branch. The branch should be specified as a branch name. If you omit this header, the branch that you specified when signing in is used as the current branch.

Request Body

For information on the request body, see the *Request Body* section of *Create a Record* and the *Request Body* section of *Update a Record*.

Response Status Codes

For information on possible response status codes, see the *Response Status Codes* section of *Create a Record* and the *Response Status Codes* section of *Update a Record*.

Example

The following request is an example of creating a new journal transaction with *SweetLife Store* set as the new current branch and *01-Jan-2024* set as the business date.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/JournalTransaction
Accept: application/json
Content-Type: application/json
PX-CbApiBusinessDate: 2024/01/01
PX-CbApiBranch: SweetLife Store

{
    "Description": {"value": "Test transaction description"},
    "PostPeriod": {"value": "01-2024"}
}
```

Retrieve the Acumatica ERP Version and the List of Endpoints

To obtain the Acumatica ERP version and the list of contract-based endpoints available in this version by using the REST API, you access the needed URL with the `GET` HTTP method. The remainder of this topic provides details on the request and the response.

If the request is sent without the authentication information, the list contains only the endpoints available in the system by default. If the request is sent with the authentication information for a particular tenant, the list also includes the custom endpoints configured in this tenant of the Acumatica ERP instance.

HTTP Method and URL

To retrieve the Acumatica ERP version and the list of contract-based endpoints available in this version, you use the `GET` HTTP method and the following URL.

```
GET http://<Acumatica ERP instance URL>/entity
```

In this URL, *<Acumatica ERP instance URL>* is the URL of the Acumatica ERP instance for which you want to obtain information about the version and endpoints.

For example, suppose that you work with a local Acumatica ERP instance with the name *AcumaticaDB*. You would use the following URL to retrieve the information: *http://localhost/AcumaticaDB/entity*.

Parameters

You use no parameters when you retrieve the Acumatica ERP version and the list of contract-based endpoints available in this version.

Request Headers

You can specify the following header in the request.

Header	Description
`Accept`	Optional. Specifies the format of the response body, which can be `application/json`.

Response Status Codes

The following table lists the HTTP status codes that the system returns for a request that retrieves the Acumatica ERP version and the list of endpoints.

Code	Description
200	The request has been completed successfully. The response body contains the Acumatica ERP version and the list of contract-based endpoints available in this version.
429	The number of requests has exceeded the limit imposed by the license (see *License Restrictions for API Users*).
500	An internal server error has occurred.

Example

The following request obtains the Acumatica ERP version and the list of contract-based endpoints available in this version.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /entity HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
```

Following is an example of a response in JSON format.

```
{
    "version": {
        "acumaticaBuildVersion": "24.193.0113",
        "databaseVersion": "24.193.0113"
    },
    "endpoints": [
        {
            "name": "Default",
            "version": "20.200.001",
            "href": "/testdata/entity/Default/20.200.001/"
        },
        {
            "name": "eCommerce",
            "version": "20.200.001",
            "href": "/testdata/entity/eCommerce/20.200.001/"
        },
        {
            "name": "MANUFACTURING",
            "version": "21.200.001",
            "href": "/testdata/entity/MANUFACTURING/21.200.001/"
        },
        {
            "name": "Default",
            "version": "22.200.001",
            "href": "/testdata/entity/Default/22.200.001/"
        },
        {
            "name": "eCommerce",
            "version": "22.200.001",
            "href": "/testdata/entity/eCommerce/22.200.001/"
        },
        {
            "name": "GLConsolidation",
            "version": "22.200.001",
            "href": "/testdata/entity/GLConsolidation/22.200.001/"
        },
        {
            "name": "MANUFACTURING",
            "version": "23.100.001",
            "href": "/testdata/entity/MANUFACTURING/23.100.001/"
```

```
        },
        {
            "name": "Default",
            "version": "23.200.001",
            "href": "/testdata/entity/Default/23.200.001/"
        },
        {
            "name": "DeviceHub",
            "version": "23.200.001",
            "href": "/testdata/entity/DeviceHub/23.200.001/"
        },
        {
            "name": "eCommerce",
            "version": "23.200.001",
            "href": "/testdata/entity/eCommerce/23.200.001/"
        },
        {
            "name": "MANUFACTURING",
            "version": "23.200.001",
            "href": "/testdata/entity/MANUFACTURING/23.200.001/"
        },
        {
            "name": "Default",
            "version": "24.200.001",
            "href": "/testdata/entity/Default/24.200.001/"
        },
        {
            "name": "eCommerce",
            "version": "24.200.001",
            "href": "/testdata/entity/eCommerce/24.200.001/"
        },
        {
            "name": "MANUFACTURING",
            "version": "24.200.001",
            "href": "/testdata/entity/MANUFACTURING/24.200.001/"
        }
    ]
}
```


Parameters for Retrieving Records

When you retrieve records from Acumatica ERP by using the contract-based REST API, you can use the URL parameters to filter records by the conditions you specify, expand particular entities, and retrieve the values of fields that are not defined in the contract of the endpoint. In this chapter, you can find descriptions of these parameters.

$filter Parameter

When you retrieve records from Acumatica ERP by using the contract-based REST API, you use the *$filter* parameter to specify the conditions that determine which records should be selected from Acumatica ERP. You use *OData URI conventions* to specify the value of the parameter.

You can specify multiple conditions for the same field or different fields in a filter by using the AND and OR operators.

When you specify the value of the parameter, you can use the following functions as they are defined in OData:

- `substringof`
- `startswith`
- `endswith`

You can use the following custom function to filter records by the values of custom fields: `cf.<Type name>(f='<View name>.<Field name>')`, where `<Type name>` is the type of the custom element, `<View name>` is the name of the data view that contains the element, and `<Field name>` is the name of the element.

Example: Simple Condition

To obtain stock item records that have the *Active* status in Acumatica ERP, you use the following filter: *$filter=ItemStatus eq 'Active'*.

Example: Condition on a Linked Entity

To obtain a customer record that has the *demo@gmail.com* email address, you use the following filter: *$filter=MainContact/Email eq 'demo@gmail.com'*. (The `Email` field is defined in a linked entity, which is available through the `MainContact` property.)

> The REST API does not support filtering on detail records. If you specify a filter on detail records, the results of the request cannot be predicted.

Example: Multiple Conditions

To obtain stock item records that have the *Active* status in Acumatica ERP and have been modified later than July 15, 2024, you use the following filter: *$filter=ItemStatus eq 'Active' and LastModified gt datetimeoffset'2024-07-15T10%3A31%3A28.402%2B03%3A00'*.

> You should encode date and time values in URL format before passing them as the value of the parameter. For example, you can encode the current date and time by using the `System.Net.WebUtility.URLEncode()` method as follows: `WebUtility.UrlEncode(new DateTimeOffset(DateTime.Now).ToString("yyyy-MM-ddTHH:mm:ss.fffK"))`.

Example: Condition on a Field Not Defined in the Endpoint

Suppose that in an extension of the *Default/24.200.001* endpoint, you added the `RepairItemType` field to the top-level `StockItem` entity. This field corresponds to the **Repair Item Type** custom element, which has been added to the **General** tab (**Item Defaults** section) of the *Stock Items* (IN202500) form. If you want to obtain all records on the *Stock Items* form for which the value of the custom **Repair Item Type** element is *Battery*, you would use the following parameter string: *$filter=cf.String(f='ItemSettings.UsrRepairItemType') eq 'Battery'*.

> For details on how to find out the name of a custom element and the name of its data view, see *Custom Fields*.

$top Parameter

When you retrieve records from Acumatica ERP by using the contract-based REST API, you use the *$top* parameter to specify the number of records to be returned from Acumatica ERP. That is, if you specify *N* as the value of this parameter, the first *N* records will be returned from Acumatica ERP.

If you do not use the *$top* parameter in a request, all records will be returned. However, if a limit is specified in SQL Server Resource Governor and the number of available records exceeds the limit, an error will be returned.

Example: Retrieval of the First Five Records

To obtain only first five records from the list, you use the following parameter string: *$top=5*.

$skip Parameter

When you retrieve records from Acumatica ERP by using the contract-based REST API, you use the *$skip* parameter to specify the number of records to be skipped from the list of returned records. That is, if you specify *N* as the value of this parameter, the first *N* records will be skipped from the list of returned records.

If you use the *$skip* and *$top* parameters together, the *$skip* parameter is applied first.

By using the *$skip* and *$top* parameters, you can implement the pagination of records. However, there is no parameter to learn the number of records that meet a particular request.

Example: Skipping the First Five Records

If you want to skip the first five records from the list and obtain the rest of the records, you use the following parameter string: *$skip=5*.

$expand Parameter

When you retrieve records from Acumatica ERP by using the contract-based REST API, you use the *$expand* parameter to specify the linked and detail entities that should be expanded. By default, no linked or detail entities are expanded; that is, only fields of the top-level entity are returned. You need to explicitly specify each linked or detail entity to be expanded.

You use *OData URI conventions* to specify the value of the *$expand* parameter.

If you use the *$expand* parameter in a request, the concurrent request limit or rate limit imposed by the license will not be exhausted sooner because of the use of this parameter.

Example: Expanding Detail Lines

To obtain the values of the warehouse detail lines of stock item records, you use the following parameter string: *$expand=WarehouseDetails*.

Example: Nested Expanding

If you specify *$expand=MainContact* for the `Customer` entity, only the `Contact` linked entity of the `Customer` entity is expanded, but the `Address` linked entity within `MainContact` is not. To expand the `Address` entity,

you should explicitly specify the `Address` entity to be expanded as follows: *$expand=MainContact,MainContact/Address*.

Example: Expanding File Data for a Top-Level Entity

You use the *$expand=files* parameter for a top-level entity to retrieve the data of the files attached to the entity.

Example: Expanding File Data for a Detail Entity

You use the *$expand=WarehouseDetails/files* parameter to retrieve the data of the files attached to the warehouse detail lines of stock item records.

$select Parameter

When you retrieve records from Acumatica ERP by using the contract-based REST API, you use the *$select* parameter to specify the fields of the entity to be returned from Acumatica ERP. By default, all fields of the entity are returned.

You use *OData URI conventions* to specify the value of the *$select* parameter.

Example: Top-Level Fields

To obtain only the order types and order numbers of sales orders, you use the following parameter string: *$select=OrderType,OrderNbr*.

Example: Fields of Nested Entities

To obtain only the customer name, email address, and city, you use the following parameter string: *$select=CustomerName,MainContact/Email,MainContact/Address/City*.

You also need to expand the `MainContact` and `Address` linked entities by using the *$expand* parameter. For a detailed example, see *Retrieve the List of Customers with Contacts*.

$custom Parameter

When you retrieve records from Acumatica ERP by using the contract-based REST API, you use the *$custom* parameter to specify the fields that are not defined in the contract of the endpoint to be returned from Acumatica ERP. That is, you can use this parameter to obtain the values of the predefined elements on an Acumatica ERP form that are not included in the entity definition, the values of user-defined fields, and the values of elements that were added to the Acumatica ERP form in a customization project.

You use one of the following formats to specify the element whose value should be returned:

- If a top-level entity contains the custom field that corresponds to the element: `<View name>.<Field name>`, where you replace `<View name>` with the name of the data view that contains the element and `<Field name>` with the internal name of the element.
- If a linked or detail entity contains the custom field that corresponds to the element: `<Entity name>/<View name>.<Field name>`, where you replace `<Entity name>` with the name of the linked or detail entity that contains the field, `<View name>` with the name of the data view that contains the element, and `<Field name>` with the internal name of the element.

> ⚠ If you want to obtain the value of a custom field of a linked or detail entity, in addition to specifying the *$custom* parameter, you have to specify this entity in the *$expand* parameter.

- If the element is a user-defined field: `<View name>.Attribute<AttributeID>`, where you replace `<View name>` with the name of the data view that contains the element, and `<AttributeID>` with the ID of the attribute that corresponds to the user-defined field.

  For details about user-defined fields, see *User-Defined Fields*.

If you want to obtain the values of multiple custom elements, you specify the custom elements to be returned, separated by commas. For details on how to find out the field name and the name of the data view, see *Custom Fields*.

Example: Custom Field of a Top-Level Entity

Suppose that in an extension of the *Default/24.200.001* endpoint, you added the `RepairItemType` field to the top-level `StockItem` entity. This field corresponds to the **Repair Item Type** custom element that has been added to **General** tab of the *Stock Items* (IN202500) form. If you want to obtain the value of this element, you use the following parameter string: *$custom=ItemSettings.UsrRepairItemType*.

Example: Custom Field of a Detail Entity

Suppose that in an extension of the *Default/24.200.001* endpoint, you added the `RepairItemType` field to the `SalesOrderDetail` detail entity. This field corresponds to the **Repair Item Type** custom column, which has been added to the **Details** tab of the *Sales Orders* (SO301000) form. If you want to obtain the value of this element, you use the following parameter string: *$custom=Details/Transactions.UsrRepairItemType*. You also use the *$expand* parameter as follows: *$expand=Details*.

Example: User-Defined Field

Suppose that on the *Sales Orders* (SO301000) form, you have added a user-defined field for the *OPERATSYST* attribute. If you want to obtain the value of this user-defined field through the *Default/24.200.001* endpoint, you use the following parameter string: *$custom=Document.AttributeOPERATSYST*.

Related Links

- *Custom Fields*

Account

This chapter presents code examples showing requests that use the `Account` entity.

Add an Account to an Account Group

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can add an account to an account group.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to add the *40000* account to the *ACCG02* account group through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Account
Accept: application/json
Content-Type: application/json

{
  "AccountCD" : {"value" : "40000"},
  "AccountGroup" : {"value" : "ACCG02"}
}
```

Retrieve the List of Accounts in a Group

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the list of accounts in a group.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve the list of accounts of the *ACCG02* account group through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$filter=AccountGroup%20eq%20'ACCG02'&$select=AccountCD HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Account
Accept: application/json
Content-Type: application/json
```

Remove an Account from a Group

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can remove an account from a group.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to remove the *40010* account from the *ACCG02* account group through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Account
Accept: application/json
Content-Type: application/json

{
```

```
  "AccountCD" : {"value" : "40010"},
  "AccountGroup" : {"value" : null}
}
```

AccountDetailsForPeriodInquiry

This chapter presents code examples showing requests that use the `AccountDetailsForPeriodInquiry` entity.

Get General Ledger Transactions for Some Period

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve general ledger transactions for the specified period from Acumatica ERP. For this purpose, you can use the Account Details for Period (GL404001) generic inquiry.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve general ledger transactions made from March to April 2022 through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Results HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/
AccountDetailsForPeriodInquiry
Accept: application/json
Content-Type: application/json

{
    "FromPeriod": { "value": "032022" },
    "ToPeriod": { "value": "042022" }
}
```

AccountGroup

This chapter presents code examples showing requests that use the `AccountGroup` entity.

Create an Account Group

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create an account group.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create an account group with the *ACCG02* identifier through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/AccountGroup
Accept: application/json
Content-Type: application/json

{
  "AccountGroupID" : {"value" : "ACCG02"},
  "Description" : {"value" : "Test Account Group"}
}
```

Specify the Default Account of an Account Group

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can specify the default account of an account group.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to specify the *40000* account as the default account of the *ACCG02* account group through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/AccountGroup
Accept: application/json
Content-Type: application/json

{
  "DefaultAccountID" : {"value" : "40000"},
  "AccountGroupID" : {"value" : "ACCG02"}
}
```

Activity

This chapter presents code examples showing requests that use the `Activity` entity.

Create an Activity that Is Linked to a Case

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can establish the relationship of the `Activity`, `Event`, `Task`, and `Email` objects with cases in Acumatica ERP. You can also establish this relationship by using the **Create Activity**, **Create Event**, **Create Task**, or **Create Email** button on **Activities** tab of the *Cases* (CR306000) form. For details about the management of cases, see *Case Management: General Information*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create an activity that is linked to a case through the contract-based REST API. In the `RelatedEntityNoteID` field, you use the `ID` value of the case for linking. This `ID` value is present, for example, in the response body when you create a case, as described in *Create a Case*.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Activity
Accept: application/json
Content-Type: application/json

{
    "Summary": {"value": "Automated Test"},
    "Type": {"value": "M"},
    "RelatedEntityNoteID": {"value": "{{NoteID}}"},
    "RelatedEntityType": {"value": "PX.Objects.CR.CRCase"},
    "ActivityDetails":{"value": "Automated Test"}
}
```

Usage Notes

You use the following fields of the `Activity` entity:

- `RelatedEntityNoteID`: The `NoteID` value of the object with which the relationship is established.

- `RelatedEntityType`: The full name of the type of the object with which the relationship is established (namely, `PX.Objects.CR.CRCase`).
- `RelatedEntityDescription`: The description of the related object. This field is read-only.

Create an Activity that Is Linked to a Customer

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can establish the relationship of the `Activity`, `Event`, `Task`, and `Email` objects with customers in Acumatica ERP. You can also establish this relationship by using the **Create Activity**, **Create Event**, **Create Task**, or **Create Email** button on **Activities** tab of the *Customers* (AR303000) form.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create an activity that is linked to a customer through the contract-based REST API. In the `RelatedEntityNoteID` field, you use the `ID` value of the customer for linking. This `ID` value is present, for example, in the response body when you create a customer, as described in *Create a Customer*.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Activity
Accept: application/json
Content-Type: application/json

{
    "Summary": {"value": "Automated Test 2"},
    "Type": {"value": "M"},
    "RelatedEntityNoteID": {"value": "f37200d6-35ea-eb11-9dee-9828a61840c3"},
    "RelatedEntityType": {"value": "PX.Objects.AR.Customer"},
    "ActivityDetails":{"value": "Automated Test 2"}
}
```

Usage Notes

You use the following fields of the `Activity` entity:

- `RelatedEntityNoteID`: The `NoteID` value of the customer with which the relationship is established.
- `RelatedEntityType`: The full name of the type of the object with which the relationship is established (namely, `PX.Objects.AR.Customer`). This field is optional, but we recommend using it. The field can be omitted because an object with which the relationship is established has a persisting note record in the database.
- `RelatedEntityDescription`: The description of the related object. This field is read-only.

Create an Activity that Is Linked to a Lead

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can establish the relationship of the `Activity`, `Event`, `Task`, and `Email` objects with leads in Acumatica ERP. You can also establish this relationship by using the **Create Activity**, **Create Event**, **Create Task**, or **Create Email** button on **Activities** tab of the *Leads* (CR301000) form. For more information about the creation of activities, see *Emails and Activities: Activities*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create an activity that is linked to a case through the contract-based REST API. In the `RelatedEntityNoteID` field, you use the `ID` value of the case for linking. This `ID` value is present, for example, in the response body when you create a lead, as described in *Create a Lead*.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Activity
Accept: application/json
Content-Type: application/json

{
    "Summary": {"value": "Automated Test"},
```

```
    "Type": {"value": "M"},
    "RelatedEntityNoteID": {"value": "{{NoteID}}"},
    "RelatedEntityType": {"value": "PX.Objects.CR.CRLead"},
    "ActivityDetails":{"value": "Automated Test"}
}
```

Usage Notes

You use the following fields of the `Activity` entity:

- `RelatedEntityNoteID`: The `NoteID` value of the lead with which the relationship is established.
- `RelatedEntityType`: The full name of the type of the object with which the relationship is established (namely, `PX.Objects.CR.CRLead`).
- `RelatedEntityDescription`: The description of the related lead. This field is read-only.

Bill

This chapter presents code examples showing requests that use the `Bill` entity.

Create a Bill for Particular Lines of a Purchase Order

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create bills for particular lines of purchase orders. For details about the processing of inventory purchases, see *Purchases of Stock Items: General Information*.

To create an AP bill for particular lines of a purchase order, you need to specify the line numbers in the request by using the `POLine` property of the `BillDetail` detail entity of the `Bill` entity.

If you want to add all lines of a purchase order to an AP bill, in the `BillDetail` detail entity of the `Bill` entity, you need to specify only the order type and number of the purchase order.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

> Make sure the financial periods are open for the year for which you are creating an AP bill. If the financial periods are not open, the creation of the AP fill fails with an error, such as *Post Period cannot be empty*. For details about how to open financial periods, see an example in *Opening Financial Periods: Process Activity*.

Request

You can use the following example of an HTTP request to create an AP bill for a line of a purchase order.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$select=ReferenceNbr,Details/POOrderNbr,Details/POOrderNbr,
    Details/InventoryID,Details/Qty&$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Bill
Accept: application/json
Content-Type: application/json

{
    "Vendor": {
        "value": "PRINTICO"
    },
    "VendorRef": {
        "value": "123"
    },
    "Description": {
        "value": "Bill for particular lines of a purchase order"
    },
    "Details": [
        {
            "POOrderType": {
                "value": "Normal"
            },
            "POOrderNbr": {
                "value": "000001"
            },
            "POLine": {
                "value": 1
            },
            "Qty":
            {
             "value": 5
            }
        }
    ]
}
```

Create a Bill for Particular Lines of a Purchase Receipt

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create bills for particular lines of purchase receipts. For details about the processing of inventory purchases, see *Purchases of Stock Items: General Information*.

To create an AP bill for particular lines of a purchase receipt, you need to specify the line numbers in the request by using the `POReceiptLine` property of the `BillDetail` detail entity of the `Bill` entity.

> ⚠ If you want to add all lines of a purchase receipt to an AP bill, in the `BillDetail` detail entity of the `Bill` entity, you need to specify only the receipt number of the purchase receipt.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

> ⓘ Make sure the financial periods are open for the year for which you are creating an AP bill. If the financial periods are not open, the creation of the AP fill fails with an error, such as *Post Period cannot be empty*. For details about how to open financial periods, see an example in *Opening Financial Periods: Process Activity*.

Request

You can use the following example of an HTTP request to create an AP bill for a line of a purchase receipt.

> ⓘ In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$select=ReferenceNbr,Details&$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Bill
Accept: application/json
Content-Type: application/json

{
 "Vendor": {
        "value": "OFFICEUP"
```

```
    },
    "VendorRef": {
        "value": "123"
    },
    "Description": {
        "value": "Bill for particular lines of a purchase receipt"
    },
    "Details": [
        {
            "POReceiptNbr": {
                "value": "000001"
            },
             "POReceiptLine": {
                "value": 1
            }
        }
    ]
}
```

Usage Notes

In the request above, you can specify the `Details/POReceiptType` value to create a bill for particular lines of a purchase receipt or purchase return. If you do not specify this value, the *Receipt* value is used by default.

Create a Bill with Tax Parameters Overridden

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create bills with the tax parameters overridden.

System Preparation

Before you test the code below, do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

To further prepare the system, you need to create a tax and a tax category that uses this tax, and modify the existing tax zone so that it will use the new tax. You perform these tasks as follows:

> If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

1. On the *Taxes* (TX205000) form, create a tax with the tax schedule as follows:

a. Create a tax with the following parameters:
   - **Tax ID**: *PST*
   - **Description**: *Tax ID for testing*
   - **Tax Type**: *VAT*
   - **Include in VAT Taxable Total**: Selected
   - **Exclude from Tax on Tax Calculation**: Selected
   - **Tax Agency**: *CANADABC*

b. On the **Tax Schedule** tab of the form, add two rows with the following data.

Column	First Row	Second Row
Start Data	*1/1/1900*	*1/1/1900*
Tax Rate	*5*	*5*
Min. Taxable Amount	*0*	*0*
Max. Taxable Amount	*0*	*0*
Reporting Group	*Input Tax (purchases)*	*Output Tax (sales)*
Group Type	*Input*	*Output*

c. Click **Save**.

2. On the *Tax Categories* (TX205500) form, create the *TEST* tax category with the *Tax category for test* description and add the *CAGST* and *PST* taxes to the tax category.
3. On the **Applicable Taxes** tab of the *Tax Zones* (TX206000) form, add the *PST* tax to the *CANADABC* tax zone.

Request

You can use the following example of an HTTP request to create an AP bill with the tax parameters that are different from those registered in the system.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details,TaxDetails HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Bill
Accept: application/json
Content-Type: application/json

{
    "Amount": {
        "value": 1120.0
    },
    "CashDiscountDate": {
        "value": "11/18/2021"
    },
    "CurrencyID": {
```

```
        "value": "USD"
    },
    "Date": {
        "value": "10/18/2021"
    },
    "DueDate": {
        "value": "11/18/2021"
    },
    "Hold": {
        "value": false
    },
    "LocationID": {
        "value": "MAIN"
    },
    "PostPeriod": {
        "value": "102021"
    },
    "Status": {
        "value": "Balanced"
    },
    "TaxAmount": {
        "value": 120.0
    },
    "Terms": {
        "value": "30D"
    },
    "Type": {
        "value": "Bill"
    },
    "Vendor": {
        "value": "AAVENDOR"
    },
    "VendorRef": {
        "value": "test-API22"
    },
    "Description": {
        "value": "test-API"
    },
    "IsTaxValid": {
        "value": true
    },
    "Details": [
        {
            "Account": {
                "value": "50000"
            },
            "Branch": {
                "value": "PRODWHOLE"
            },
            "CostCode": {
                "value": null
            },
            "ExtendedCost": {
                "value": 1000.0
            },
            "Project": {
                "value": "X"
```

```
            },
            "Subaccount": {
                "value": "000000"
            },
            "TaxCategory": {
                "value": "TEST"
            }
        }
    ],
    "TaxDetails": [
        {
            "TaxID": {
                "value": "CAGST"
            },
            "TaxableAmount": {
                "value": 1000.0
            },
            "TaxAmount": {
                "value": 150.0
            },
            "ExpenseAmount": {
                "value": 0.0
            }
        },
        {
            "TaxID": {
                "value": "PST"
            },
            "TaxableAmount": {
                "value": 1000.0
            },
            "TaxAmount": {
                "value": 150.0
            },
            "custom": {
                "Taxes": {
                    "CuryExpenseAmt": {
                        "type": "CustomDecimalField",
                        "value": 70
                    }
                }
            }
        }
    ],
    "custom": {
        "CurrentDocument": {
            "TaxZoneID": {
                "type": "CustomStringField",
                "value": "CANADABC"
            }
        }
    }
}
```

In the request above, you have set the `IsTaxValid` value to *true* to use in the bill the tax parameters specified in the `TaxDetails` field. If you do not specify this value, the tax parameters that are registered in the system will be

used instead. In particular, the tax amounts equal to *50* will be set both for the *CAGST* and *PST* taxes in the created bill.

Usage Notes

If the `IsTaxValid` field is specified in the request body, its value is set to *true*, and an external tax zone is specified, then the tax details from the request body are used without modification, and no other taxes are applied.

If the `IsTaxValid` field is specified in the request body, its value is set to *true*, and an internal tax zone is specified, then the tax details from the request body are compared to those calculated by the system. If the taxes specified in the request body are present among those calculated by the system, then the tax details from the request body are used without modification and no other taxes are applied.

If the `IsTaxValid` field is not specified in the request body or its value is not set to *true*, tax calculation is performed by the system.

Approve a Bill

If you are using the REST API to integrate Acumatica ERP with an external system, this external system can approve a bill.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, enable the *Inventory and Order Management* and *Approval Workflow* features, or make sure that they are enabled.
3. On the form toolbar of the *Assignment and Approval Maps* (EP205500) form, click **Add Approval Map**.
4. On the *Approval Maps* (EP205015) form, which opens, configure a new approval map as follows:
   a. In the **Name** box, type *Bill Approval*.
   b. In the **Entity Type** box, select *Bills and Adjustments*.
   c. In the **Steps** pane, click **Add Step**.
   d. Select the **Rule** tree item that has appeared.
   e. On the **Conditions** tab, click **Add Row**, and specify the following settings in the row:
      - **Active**: Selected
      - **Entity**: *Vendor*
      - **Field Name**: *Vendor*
      - **Condition**: *Equals*
      - **Value**: *IRS*
   f. On the **Rule Actions** tab, change the approval settings to the following:
      - **Approver**: *Employee*
      - **Employee**: *Anna Johnson*
   g. Save the approval map.
5. On the *Accounts Payable Preferences* (AP101000) form, make the following changes:

a. On the **General** tab, in the **Data Entry Settings** section, select the **Require Approval of Bills Prior to Payment** check box.

b. On the **Approval** tab, add a row to the table, and specify the following settings in the row:

- **Active**: Selected
- **Type**: *Bill*
- **Approval Map**: *Bill Approval*
- **Pending Approval Notification**: *AP Bill*

c. Save your changes.

6. On the *Bills and Adjustments* (AP301000) form, open the *000159* bill.
7. On the form toolbar, click **Remove Hold**, then click **Save**.
8. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
9. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance), the *HEADOFFICE* branch, the *johnson* name, and the *123* password.

Request

You can use the following example of an HTTP request to approve the *000159* AP bill.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /Approve HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Bill
Accept: application/json
Content-Type: application/json

{
    "entity": {
        "Type": {"value": "Bill"},
        "ReferenceNbr": {"value": "000159"}
    },
    "parameters": {}
}
```

Related Links

- *Execute a Custom Action*

Release Retainage

In some industries, such as construction, it is necessary to withhold a portion of the contract amount until the work has been completed to ensure that the vendor will satisfy its obligations and complete a particular project. If you are using the REST API to integrate Acumatica ERP with an external system, this external system can process bills and debit adjustments with retained amounts, including the creation of retainage documents (that is, the release

of retainage) on the *Bills and Adjustments* (AP301000) form. For details about releasing retainage, see *Processing AP Documents with Retainage*.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Retainage Support* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a retainage bill in the amount of $500 through the REST API. In the example, you specify *April 1, 2025* as the business date. As a result of this request, a new bill is created on the *Bills and Adjustments* (AP301000) form with the **Retainage Document** check box selected.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /Bill/ReleaseRetainage HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001
Accept: application/json
Content-Type: application/json
PX-CbApiBusinessDate: 2025/04/01

{
    "entity": {
        "Type": {
            "value": "Bill"
        },
        "ReferenceNbr": {
            "value": "000072"
        }
    },
    "parameters": {
        "Date": {
            "value": "2025-03-01T11:00:00.000Z"
        },
        "PostPeriod": {
            "value": "032025"
        },
        "AmtToRelease": {
            "value": 500
        }
    }
```

```
}
```

Usage Notes

In the body of the request, you can identify the entity by using the entity ID or the key fields, which are the following:

- `Type`: The type of the document, which can be one of the following: *Bill*, *Debit Adj.*
- `ReferenceNbr`: The number that identifies the AP document in the system

In the body of the request, you specify the following parameters of the `ReleaseRetainage` action:

- `Date`: Required. The date of the created AP retainage document.
- `PostPeriod`: Required. The post period of the created AP retainage document.
- `AmtToRelease`: Optional. The retained amount to be released. If `AmtToRelease` is not specified, the AP retainage document is created in the amount of 100% of the unreleased retainage from each line in the original document. An error is returned as a result of the action execution in any of the following cases:
  - If the unreleased retainage amount of the document is less than the specified `AmtToRelease` value
  - If `AmtToRelease` is greater than 0 and the AP document for which the action is executed has the **Pay by Line** check box selected on the *Bills and Adjustments* (AP301000) form
  - If `AmtToRelease` is less than or equal to 0

Related Links

- *Processing AP Documents with Retainage*

BillOfMaterial

This chapter presents code examples showing requests that use the `BillOfMaterial` entity.

Create a Bill of Material

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create bills of material (BOMs). For details about the management of bills of material, see *Managing Bills of Material*.

System Preparation

Before you test the code in the following section, you need to:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Manufacturing* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Requests

> In the request examples below, <Acumatica ERP instance URL> is the URL of the Acumatica ERP instance (such as https://my.acumatica.com/MyInstance). You can omit the instance name in the URL (such as https://my.acumatica.com) if the instance is installed in the root of the website.

You can use the following HTTP request example to create a BOM containing information about an operation and a material used.

```
PUT ?$expand=Operations,Operations/Material HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/MANUFACTURING/24.200.001/BillOfMaterial
Accept: application/json
Content-Type: application/json

{
    "BOMID": {
        "value": "<NEW>"
    },
    "Description": {
        "value": "Test BOM"
    },
    "InventoryID": {
        "value": "CABINET"
    },
    "Operations": [
        {
            "OperationNbr": {
                "value": "0010"
            },
            "WorkCenter": {
                "value": "WC10"
            },
            "Material" :[
                {
                    "InventoryID": {
                        "value": "HINGE"
                    },
                    "UOM":{
                        "value": "EA"
                    }
                }
            ]
        }
  ],
    "Revision": {
        "value": "A"
    }
}
```

In this example, the *<NEW>* value is specified for the BOMID field, and the next identifier is assigned to this field for the newly created BOM. You can also specify an empty string as a value for the BOMID field, which will lead to the same result.

```
{
```

```
    "BOMID": { "value": "" },
    ...
}
```

Retrieve the Bills of Material

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the list of existing bills of material (BOMs). For details about the management of bills of material, see *Managing Bills of Material*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Manufacturing* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Request

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

You can use the following HTTP request example to retrieve the list of BOMs registered in the system along with their operations.

```
GET ?$expand=Operations HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/MANUFACTURING/24.200.001/BillOfMaterial
Accept: application/json
Content-Type: application/json
```

If you want to retrieve additional details of the operations of BOMs, you cannot specify the `Operations/Material`, `Operations/Overheads`, `Operations/Steps`, or `Operations/Tools` fields in the *$expand* parameter of the above request, because this would cause the *Optimization cannot be performed* error to be generated. Instead, you should retrieve a single BOM and specify these fields in the *$expand* parameter of the request (see *Retrieve the Details of a Bill of Material's Operations*).

Retrieve the Details of a Bill of Material's Operations

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve a bill of material (BOM) along with details of its operations. For details about the management of bills of material, see *Managing Bills of Material*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Manufacturing* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Request

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

You can use the following HTTP request example to retrieve the BOM with the *BOM000001* ID and the *A* revision along with the details of the BOM's operations.

```
GET /BOM000001/A?$expand=Operations,Operations/Material,Operations/Overheads,
    Operations/Steps,Operations/Tools HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/MANUFACTURING/24.200.001/BillOfMaterial
Accept: application/json
Content-Type: application/json
```

BusinessAccount

This chapter presents code examples showing requests that use the `BusinessAccount` entity.

Retrieve the List of Business Accounts

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the list of available business accounts along with their data from Acumatica ERP. This list is

exposed by the Business Accounts (CR3030PL) generic inquiry. The *Business Accounts* (CR303000) form displays the data of an individual business account.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following HTTP request example to retrieve data of the business account with the *ABAKERY* ID.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$expand=Activities,Campaigns,Cases,Orders
     &$filter=BusinessAccountID%20eq%20'ABAKERY' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/BusinessAccount
Accept: application/json
Content-Type: application/json
```

Case

This chapter presents code examples showing requests that use the `Case` entity.

Create a Case

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create cases in Acumatica ERP. For more information about the creation of cases, see *Creating Cases*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.

2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a case through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Case
Accept: application/json
Content-Type: application/json

{
    "ClassID": {"value": "JREPAIR"},
    "BusinessAccount": {"value": "ABAKERY"},
    "ContactID": {"value":  "100211"},
    "Subject": {"value":  "Some Subject"},
    "custom": {
        "Case": {
            "AttributeMODEL": {
                "value": "JUICER15:Commercial juicer with a prod rate of 1.5 l per min"
            }
        }
    }
}
```

Link a Case to Another Case

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create cases in Acumatica ERP and link them to another case defined in Acumatica ERP. For details about the relations between cases, see *Case Management: General Information*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make

sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a case and link it to another case through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Case?$expand=RelatedCases
Accept: application/json
Content-Type: application/json

{
    "ClassID": {
        "value": "SERVCONS"
    },
    "BusinessAccount": {
        "value": "GOODFOOD"
    },
    "Subject": {
        "value": "Billing plan"
    },
    "RelatedCases": [
        {
            "CaseID": {
                "value": "000004"
            }
        }
    ]
}
```

Usage Notes

To create a case and establish its relation with another case, you need to specify the case ID of the related case by using the `CaseID` field of the `RelatedCases` detail entity of the `Case` entity.

> The `ParentCaseID` field of the `RelatedCases` detail entity of the `Case` entity is never returned by the system and cannot be used for the assignment of values.

Related Links

- *Case Management: General Information*

CompaniesStructure

This chapter presents code examples showing requests that use the `CompaniesStructure` entity.

Retrieve the Companies' Structure

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve information about the companies' structure—that is, the structure of the companies and branches in the tenant.. For this purpose, the Company Branch (CS401000) generic inquiry is used.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

You can use the following HTTP request example to retrieve the companies' structure.

```
PUT ?$expand=Results HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/CompaniesStructure
Accept: application/json
Content-Type: application/json

{}
```

ConfigurationEntry

This chapter presents code examples showing requests that use the `ConfigurationEntry` entity.

Retrieve a Configuration Entry

Through the contract-based REST API, an external application can retrieve features and options of the manufacturing product configurator from Acumatica ERP Manufacturing Edition and expose them to a user of the external application. These features and options are available for Acumatica ERP Manufacturing Edition users on the *Configuration Entry* (AM306000) form.

System Preparation

Before you test the code in the following section, you need to do the following:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Manufacturing* and *Product Configurator* features are enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Request

You can use the following request example to retrieve the configuration entry with the *AMC000001* ID through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /entity/MANUFACTURING/24.200.001/ConfigurationEntry/AMC000001?
    $expand=Attributes,Features/Options HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json
```

Update a Configuration Entry

Through the contract-based REST API, an external application can submit the user-selected features and options of the manufacturing product configurator to Acumatica ERP Manufacturing Edition. These features and options are available for Acumatica ERP Manufacturing Edition users on the *Configuration Entry* (AM306000) form.

System Preparation

Before you test the code in the following section, you need to do the following:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Manufacturing* and *Product Configurator* features are enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Request

You can use the following request example to submit a configuration entry through the REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT /entity/MANUFACTURING/24.200.001/ConfigurationEntry?
    $expand=Attributes,Features/Options HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

{
    "ProdOrderNbr": {"value": "AM000022"},
    "ProdOrderType": {"value": "RO"},
    "ConfigResultsID": {"value": "5"},
    "ConfigurationID": {"value": "AMC000003"},
    "Features": [
        {
            "FeatureLineNbr": { "value": 1 },
            "ConfigResultsID": { "value": "5" },
            "Options": [
                {
                    "FeatureLineNbr": { "value": 1 },
                    "OptionLineNbr": { "value": 1 },
                    "ConfigResultsID": { "value": "5" },
                    "Included": { "value": true }
                },
                {
                    "FeatureLineNbr": { "value": 1 },
                    "OptionLineNbr": { "value": 2 },
                    "ConfigResultsID": { "value": "5" },
                    "Included": { "value": true }
                }
            ]
        }
    ]
}
```

Contact

This chapter presents code examples showing requests that use the `Contact` entity.

Create a Contact with Attributes

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create contacts with attributes.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Customer Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following request example to create the Brent Edds lead through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Contact
Accept: application/json
Content-Type: application/json

{
    "FirstName": {"value": "Brent"},
    "LastName": {"value": "Edds"},
    "Email": {"value": "brent.edds.test27@avantehs.com"},
    "ContactClass": {"value": "ENDCUST"},
    "Attributes": [
        {
            "AttributeID": {"value": "INTEREST"},
            "Value": {"value": "Jam,Maint"}
        }
    ]
}
```

Usage Notes

The attributes of top-level entities are exposed in the `AttributeValue` entities. An `AttributeValue` entity has the following fields:

- `AttributeID`: The attribute identifier. Both internal values and external values can be used to set this field, but only the internal value can be retrieved.
- `AttributeDescription`: The external value of the attribute identifier. The field is read-only.
- `Value`: The attribute value. When the value is retrieved, the internal value is returned. To set the value, the internal value and the external value (for control types other than multiselect combo boxes) can be used. For multiselect combo boxes, an external value can be accepted only if it contains a single value without commas. For various control types, the following rules apply to the `Value` field:
  - For check boxes, *0* is returned if the check box is not selected, and *1* is returned if the check box is selected. For setting a value, *0*, *1*, *false* (case-insensitive), or *true* (case-insensitive) can be used.
  - For multiselect combo boxes, values separated by commas (namely, *Value1,Value2,Value3*) compose the internal value.
  - For selectors, the values are set and retrieved in the same way as they are for text boxes.
  - For date edit boxes, the internal values must be parsable through the use of the `System.Globalization.CultureInfo.InvariantCulture` Microsoft .NET object.
  - For combo boxes of all types, date edit boxes, and check boxes, an error message is written to the `error` field when an attempt is made to set an unsupported value.
- `ValueDescription`: The external value of the attribute. The field is read-only. The external value is based on the control type as follows:
  - For text boxes and date edit boxes, the external value is the same as the internal value.
  - For check boxes, the external value can be either *True* or *False*.
  - For combo boxes, the label is used as the external value.
  - For multiselect combo boxes, labels separated by commas and spaces after commas (namely, *Label1, Label2, Label3*) compose the external value.
- `Required`: An indicator of whether the attribute is mandatory. This field is read-only.
- `RefNoteID`: The value of the `NoteID` field of the object to which this attribute is referred. This field is read-only.

Deactivate a Contact

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can activate and deactivate contacts by setting the `Active` field of a `Contact` object to *true* or *false*, respectively.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make

sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to deactivate the Brent Edds contact through the contract-based REST API. The creation of a contact is described in *Create a Contact with Attributes*.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT $filter=FirstName%20eq%20'Brent'%20and%20LastName%20eq%20'Edds' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Contacts
Accept: application/json
Content-Type: application/json

{
    "Active": {"value": false}
}
```

Retrieve the List of Contacts

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the list of available contacts, along with their data, from Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve the detailed information about the *100073* contact through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$expand=Activities,Address,Campaigns,Cases,Notifications,Opportunities&
    $filter=ContactID%20eq%20100073 HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Contact
Accept: application/json
Content-Type: application/json
```

Link Multiple Contacts to a Customer

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can link multiple contacts to a customer in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to link the *100196* contact to the *ABAKERY* business account through the contract-based REST API. As a result of this request, the *ABAKERY* customer will have two linked contacts.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Contact
Accept: application/json
Content-Type: application/json

{
    "BusinessAccount": {"value": "ABAKERY"},
    "ContactID": {"value": 100196},
    "LastName": {"value": "Doe"}
}
```

Customer

This chapter presents code examples showing requests that use the `Customer` entity.

Create a Customer

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a customer.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a customer through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Customer
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "Cust01"},
    "CustomerName": {"value": "Customer 01"},
    "CustomerClass": {"value": "DEFAULT"}
}
```

Retrieve the List of Customers with Contacts

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the list of customers with contacts.

You will use the `GET` HTTP method and the `Customer` entity of the *Default/24.200.001* endpoint to list the customer records. The `Customer` entity is mapped to the *Customers* (AR303000) form.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

System Preparation

Before you can test the activity, you need to sign in to Acumatica ERP, as described in *Activity 1.2.1: To Sign In to Acumatica ERP*. For the request in this example, you have to pass the cookies that you have received during the sign-in.

Request

You can use the following sample request to retrieve the list of contacts through the contract-based REST API. For each contact, the main contact, including the address, is retrieved.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /entity/Default/24.200.001/Customer?
    $expand=MainContact,MainContact/Address&
    $select=CustomerID,CustomerName,CustomerClass,MainContact/Email,
        MainContact/Phone1,MainContact/Address/AddressLine1,
        MainContact/Address/AddressLine2,MainContact/Address/City,
        MainContact/Address/State,MainContact/Address/PostalCode HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
```

Usage Notes

The database can contain thousands of customer records, and each record includes dozens of fields. Thus, to achieve the best performance of the integration application during the export of records, you need to specify the fields of the customer records that should be returned. You use the `$select` parameter to specify the fields whose values should be retrieved from Acumatica ERP for each customer record.

You use the `$expand` parameter to specify the nested entities to be returned. In this parameter, you have to specify all the nested entities whose fields you need to retrieve from Acumatica ERP.

Update a Customer

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can update a customer record.

You will use the `PUT` HTTP method to update the record. You will specify the `$filter` parameter to find the needed customer record by using the email address.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

You can use the following request example to update a customer record that has the *info@jevy-comp.con* email address through the contract-based REST API. As a result of this request, the customer class and the billing contact of the customer are modified.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT /entity/Default/24.200.001/Customer
    $expand=MainContact,BillingContact&
    $select=CustomerID,CustomerClass,BillingContact/Email&
    $filter=MainContact/Email%20eq%20'info@jevy-comp.con' HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

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

Enable Currency Overriding and Rate Overriding for a Customer

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can enable currency overriding and rate overriding for a customer.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* and *Multicurrency Accounting* features are enabled.
5. Configure multicurrency support as follows:
   a. On the *Currencies* (CM202000) form, open *EUR*, select the **Active** and **Use for Accounting** boxes, and specify the following values in the other boxes:
      - **Realized Gain Account**: *83100*
      - **Realized Loss Account**: *83100*
      - **Unrealized Gain Account**: *84000*
      - **Unrealized Loss Account**: *84000*
      - **Revaluation Gain Account**: *83200*
      - **Revaluation Loss Account**: *83200*
      - **Rounding Gain Account**: *83100*
      - **Rounding Loss Account**: *83100*

      Save your changes.
   b. On the *Currency Management Preferences* (CM101000) form, click **Save**.
   c. On the **Currency Rate Entry** tab of the *Currency Rates* (CM301000) form, add a row with the following settings:
      - **From Currency**: *EUR*
      - **Currency Rate Type**: *SPOT*
      - **Currency Effective Date**: Today
      - **Currency Rate**: 1.1
      - **Mult./Div.**: *Multiply*

      Save your changes.

Request

You can use the following request example to update the *FRUITICO* customer through the contract-based REST API. Through the executing of this request, currency overriding and rate overriding are enabled and the *SPOT* currency rate type is specified for the customer.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Customer
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "EnableCurrencyOverride": {"value": true},
    "EnableRateOverride": {"value": true},
    "CurrencyRateType": {"value": "SPOT"}
}
```

Retrieve the Shipping Contact of a Customer

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the shipping contact of a customer.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve the shipping contact of the *ABAKERY* customer through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$expand=ShippingContact&$filter=CustomerID%20eq%20'ABAKERY' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Customer
Accept: application/json
Content-Type: application/json
```

CustomerPaymentMethod

This chapter presents code examples showing requests that use the `CustomerPaymentMethod` entity.

Register a Customer Credit Card

In this example, through the contract-based REST API, you will create a customer payment method that corresponds to a customer credit card. To create this customer payment method, you will use the `PUT` HTTP method and the `CustomerPaymentMethod` entity of the *Default/24.200.001* endpoint.

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

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

To be able to process credit card payments in the system, you need to do the following before you complete the example:

1. Make sure that you have configured the HTTPS connection.
2. To get a test account at the Authorize.Net payment gateway, create a sandbox account at *https://developer.authorize.net/hello_world.html*.

   After you create this account, you will get the credentials to be used in payment processing (that is, the API login ID and transaction key). After these credentials are generated, you can view the API login ID and generate a new transaction key and signature key on the **API Credentials & Keys** page of your account on

https://sandbox.authorize.net/. (You go to this page as follows: **Account > Settings > Security Settings > General Security Settings > API Credentials & Keys**; see the following screenshots.)

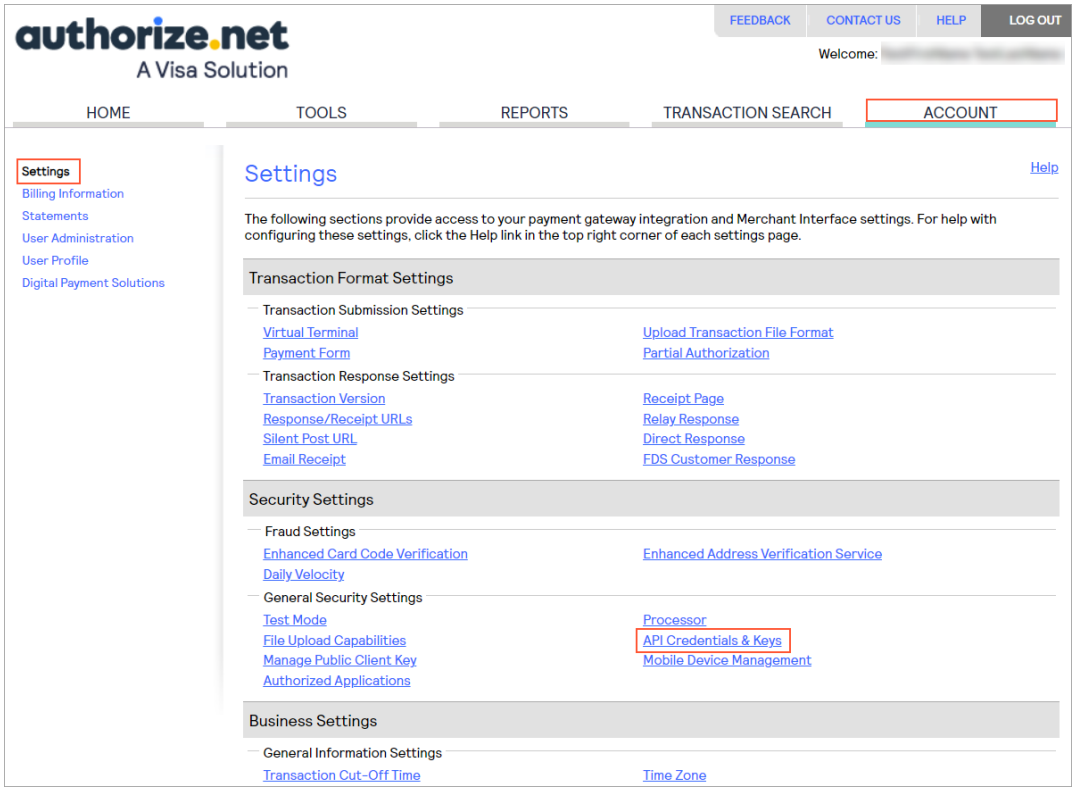

Figure: Sandbox account settings

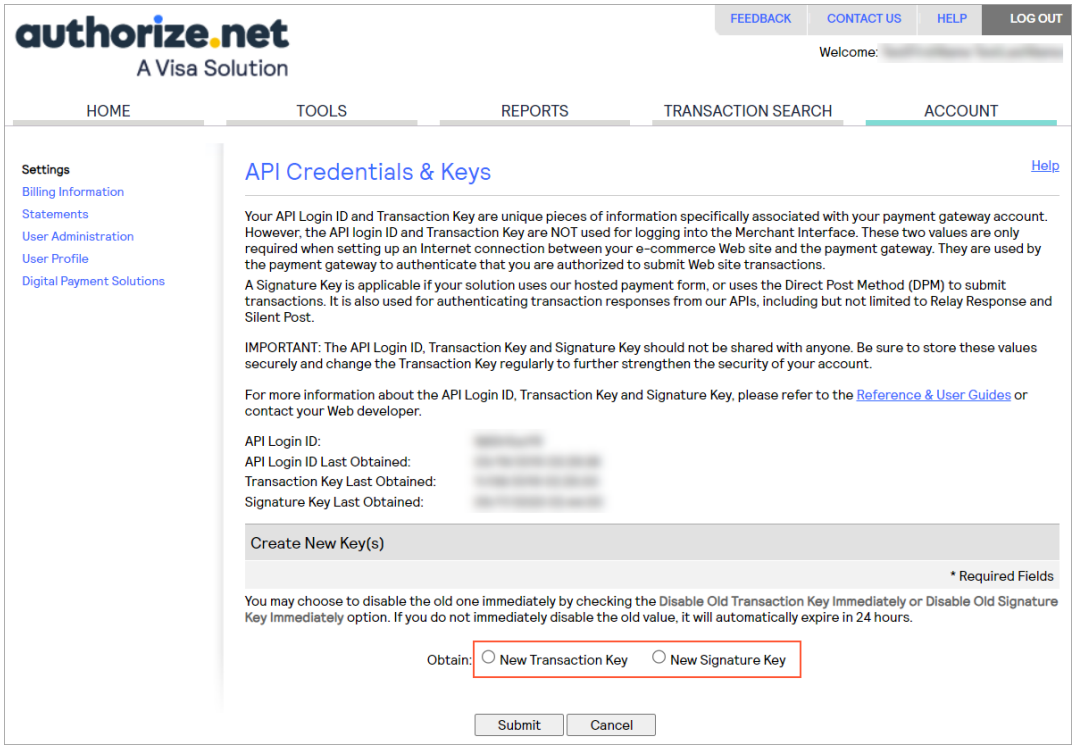

Figure: API Credentials & Keys page

3. Set up the connection with the payment gateway by using your credentials as follows:

a. On the *Processing Centers* (CA205000) form, select the *AUTHNETUSD* processing center, which has been preconfigured in the system. This processing center is associated with the *VISA* payment method, which is used in this example.

b. To use the payment gateway in test mode (as opposed to live mode), on the **Plug-In Parameters** tab, specify the following settings and save your changes:

- *MERCNAME*: The API login ID of your sandbox account
- *SIGNKEY*: The signature key of your sandbox account
- *TRANKEY*: The transaction key of your sandbox account

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

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

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

If the test credentials are accepted by the processing center, the connection to the payment gateway is ready.

4. To process the credit card payments of the customer you will work with in this activity by using Authorize.Net, create a customer profile for this customer in your Authorize.Net developer sandbox account as follows:

a. In the Customer Information Manager tool of Authorize.Net on *https://sandbox.authorize.net/*, add a new customer profile. In the **Customer ID** box of the new customer profile, specify the ID of the customer in Acumatica ERP: *C000000003*, as shown in the screenshot below.

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

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

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

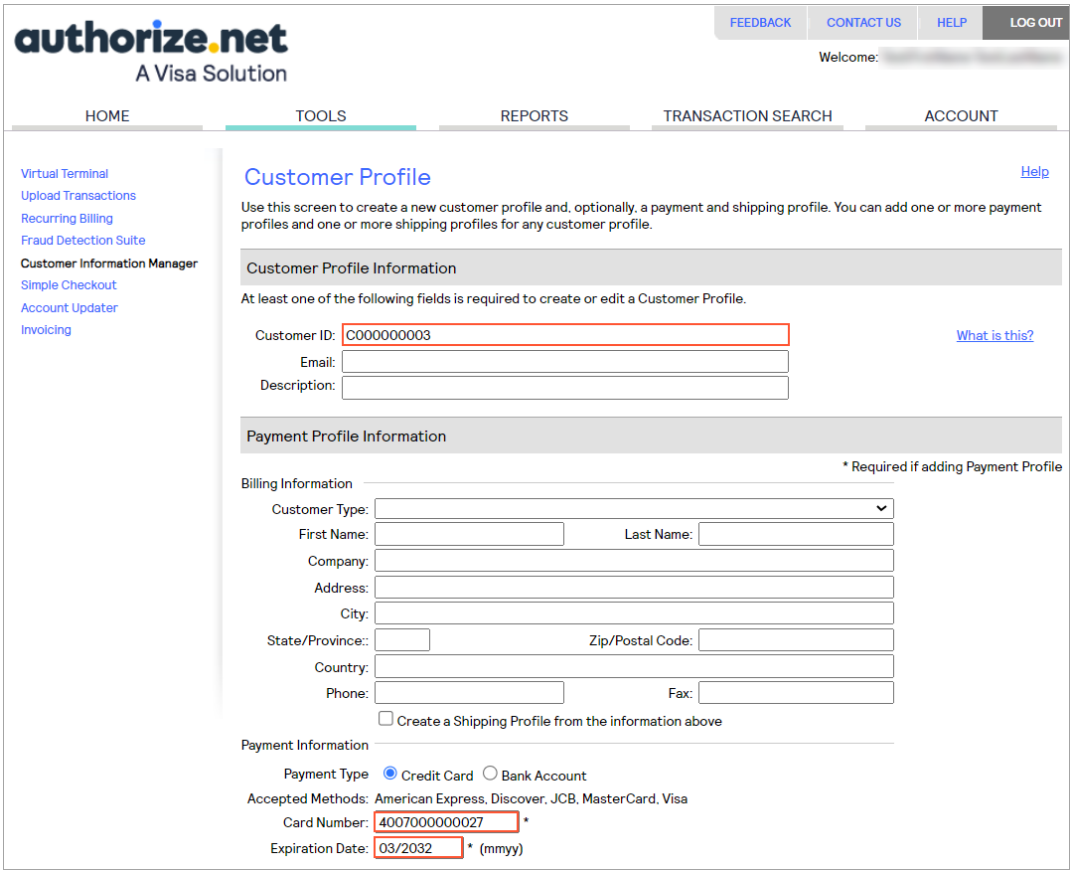

Figure: Customer Information Manager

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

Request

You can use the following request example to register a customer credit card through the contract-based REST API.

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

> **i** In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT /CustomerPaymentMethod?
    $select=CardAccountNbr,Details/Name&
    $expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001
Accept: application/json
Content-Type: application/json

{
```

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

Usage Notes

To capture a payment through the contract-based REST API, you need to use the `CaptureCreditCardPayment` action, which is available for the `Payment` entity of the *Default/24.200.001* endpoint. Because the capturing of a credit card payment is a long-running operation, you need to monitor the status of the long-running operation before retrieving the result of the capturing.

Related Links

- *Setup of Card Payment Processing (Authorize.Net)*
- *Create a Record*

DeductionBenefitCode

This chapter presents code examples showing requests that use the `DeductionBenefitCode` entity.

Create a Deduction Code

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a deduction code in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.

Request

You can use the following request example to create a deduction code through the contract-based REST API. The created deduction code will be an employee deduction and associated with employee settings.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/DeductionBenefitCode
Accept: application/json
Content-Type: application/json

{
    "DeductionBenefitCodeID": { "value": "TST" },
    "Description": { "value": "Test deduction" },
    "ContributionType": { "value": "DED" },
    "Active": { "value": false },
    "AssociatedWith": { "value": "Employee Settings" },
    "EmployeeDeduction": {
        "CalculationMethod": { "value": "GRS" },
        "Percent": { "value": 20 },
        "ApplicableEarnings": { "value": "TOT" }
    },
    "GLAccounts": {
        "DeductionLiabilityAccount": { "value": "20000" },
        "DeductionLiabilitySub": { "value": "000000" }
    }
}
```

EarningTypeCode

This chapter presents code examples showing requests that use the `EarningTypeCode` entity.

Create an Earning Type Code

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create an earning type code in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.

Request

You can use the following request example to create an earning type code through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/EarningTypeCode
Accept: application/json
Content-Type: application/json

{
    "EarningTypeCodeID": { "value": "TST" },
    "Description": { "value": "Test Code" },
    "Category": { "value": "Wage" },
    "AccrueTimeOff": { "value": true },
    "Active": { "value": false }
}
```

Employee

This chapter presents code examples showing requests that use the `Employee` entity.

Create an Employee

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create employees in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make

sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following HTTP request example to create an employee, Jane Doe.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Employee
Accept: application/json
Content-Type: application/json

{
    "Status": {"value": "Active"},
    "FinancialSettings":
    {
        "APAccount": {"value": "20000"},
        "APSubaccount": {"value": "000000"},
        "CashAccount": {"value": "10200WH"},
        "ExpenseAccount": {"value": "63000"},
        "ExpenseSubaccount": {"value": "000000"},
        "PaymentMethod": {"value": "CHECK"},
        "PrepaymentAccount": {"value": "20000"},
        "PrepaymentSubaccount": {"value": "000000"},
        "SalesAccount": {"value": "40000"},
        "SalesSubaccount": {"value": "000000"},
        "TaxZone": {},
        "Terms": {"value": "7D"}
    },
    "EmployeeSettings":
    {
        "BranchID": {"value": "HEADOFFICE"},
        "Calendar": {"value": "MAIN"},
        "CurrencyID": {"value": "USD"},
        "CurrencyRateTypeID": {"value": "SPOT"},
        "DepartmentID": {"value": "AFTERSALES"},
        "EmployeeClass": {"value": "EMPHOURLY"}
    },
    "ContactInfo":
    {
        "FirstName": {"value": "Jane"},
        "LastName": {"value": "Doe"}
    }
}
```

Retrieve Information about an Employee

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve information about employees from Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following HTTP request example to retrieve the data of the employee with the *EP00000001* ID.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$expand=ContactInfo,Delegates,EmployeeSettings,EmploymentHistory,
    FinancialSettings&$filter=EmployeeID eq 'EP00000001' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Employee
Accept: application/json
Content-Type: application/json

{}
```

EmployeePayrollClass

This chapter presents code examples showing requests that use the `EmployeePayrollClass` entity.

Create an Employee Payroll Class

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create an employee payroll class in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.
5. Update the work locations as follows:

   > If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

   a. On the *Work Locations* (PR101040) form, open the *BELLEVUE* work location.
   b. Select the **Active** check box, and click **Save** on the form toolbar. Now the *BELLEVUE* work location can be specified in an employee payroll class.

Request

You can use the following request example to create an employee payroll class through the contract-based REST API. For the class, you will specify the work location and paid time off settings.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=PayrollDefaults/WorkLocations,PTODefaults HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/EmployeePayrollClass
Accept: application/json
Content-Type: application/json

{
    "EmployeePayrollClassID": { "value": "TESTCLASS" },
    "Description": { "value": "Test CLASS" },
    "PayrollDefaults": {
        "EmployeeType": { "value": "Hourly" },
        "PayGroup": { "value": "WEEKLY" },
        "DefaultCalendar": { "value": "MAIN" },
        "WorkingWeeksPerYear": { "value": 51 },
        "OverrideHoursPerYearforCertProject": { "value": true },
        "CertifiedProjectHoursperYear": { "value": 2050 },
        "ExemptFromOvertimeRules": { "value": true },
        "NetPayMinimum": { "value": 150 },
        "MaximumPercentofNetPayforallGarnishments": { "value": 45 },
        "DefaultWCCCode": { "value": "5606" },
```

```
        "DefaultUnion": { "value": "NYS" },
        "ExemptFromCertifiedReporting": { "value": false },
        "WorkLocations": [
            {
                "LocationID": { "value": "BELLEVUE" },
                "DefaultWorkLocation": { "value": true }
            }
        ]
    },
    "PTODefaults": [
        {
            "PTOBank": { "value": "PTO" },
            "EffectiveDate": { "value": "01/01/2016" },
            "Active": { "value": "true" },
            "AccrualPercent": { "value": "7" },
            "AccrualLimit": { "value": "150" },
            "CarryoverType": { "value": "P" },
            "CarryoverAmount": { "value": "200" },
            "FrontLoadingAmount": { "value": "24" }
        }
    ]
}
```

EmployeePayrollSettings

This chapter presents code examples showing requests that use the `EmployeePayrollSettings` entity.

Specify the Employee Payroll Settings

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can specify or update the employee payroll settings in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.

Request

You can use the following HTTP request example to specify the payroll settings of the *EP00000004* employee.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/EmployeePayrollSettings
Accept: application/json
Content-Type: application/json

{
    "EmployeeID": { "value": "EP00000004" },
    "ClassID": { "value": "HOURLY" },
    "PaymentMethod": { "value": "CHECK" },
    "CashAccount": { "value": "10200WH" }
}
```

Update Work Locations in the Employee Payroll Settings

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can update the work locations in the employee payroll settings in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.
5. Update the work locations as follows:

   > If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

   a. On the *Work Locations* (PR101040) form, open the *BELLEVUE* work location.
   b. Select the **Active** check box, and click **Save** on the form toolbar. Now the *BELLEVUE* work location can be specified in an employee payroll class.
6. Execute the *Specify the Employee Payroll Settings* request.

Request

You can use the following HTTP request example to specify *BELLEVUE* as the work location in the payroll settings for the *EP00000004* employee.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=WorkLocations/WorkLocationDetails HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/EmployeePayrollSettings
Accept: application/json
Content-Type: application/json

{
    "EmployeeID": {
        "value": "EP00000004"
    },
    "WorkLocations": {
        "WorkLocationClassDefaults": {
            "value": false
        },
        "WorkLocationDetails": [
            {
                "LocationID": {
                    "value": "BELLEVUE"
                },
                "DefaultWorkLocation": {
                    "value": false
                }
            }
        ]
    }
}
```

Specify Employment Records in the Employee Payroll Settings

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can specify employment records in employee payroll settings in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make

sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.
5. Execute the *Specify the Employee Payroll Settings* request.

Request

You can use the following HTTP request example to specify an employment record in the payroll settings for the *EP00000004* employee.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=EmploymentRecords HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/EmployeePayrollSettings
Accept: application/json
Content-Type: application/json

{
    "EmployeeID": {
        "value": "EP00000004"
    },
    "EmploymentRecords": [
        {
            "StartDate": {
                "value": "2021-05-13"
            },
            "StartReason": {
                "value": "REH"
            },
            "Active": {
                "value": true
            },
            "Position": {
                "value": "ACCOUNTANT"
            }
        }
    ]
}
```

InventoryIssue

This chapter presents code examples showing requests that use the `InventoryIssue` entity.

Create an Inventory Issue

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create an inventory issue in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create an inventory issue for one unit of the *APJAM08* item through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/InventoryIssue
Accept: application/json
Content-Type: application/json

{
    "Date": { "value": "2024-12-02T00:00:00+03:00" },
    "Description": { "value": "Descr" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "ExtCost": { "value": 0 },
            "ExtPrice": { "value": 4.15 },
            "InventoryID": { "value": "APJAM08" },
            "LineNumber": { "value": 1 },
            "Location": { "value": "MAIN" },
            "Project": { "value": "X" },
            "Qty": { "value": 1 },
            "TranType": { "value": "Issue" },
            "UnitPrice": { "value": 0 },
            "UOM": { "value": "PIECE" },
            "Warehouse": { "value": "RETAIL" }
        }
    ],
    "PostPeriod": { "value": "122024" }
}
```

Release an Inventory Issue

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can release an inventory issue in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

You can use the following request example to invoke the release operation for the *000055* inventory issue through the contract-based REST API.

```
POST /ReleaseInventoryIssue HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/InventoryIssue
Accept: application/json
Content-Type: application/json

{
    "entity": {
        "ReferenceNbr": { "value": "000055" }
    }
}
```

If the *202 Accepted* status code is returned for this request, the release operation has been invoked successfully. The `Location` header of the response contains the URL that you can use to check the status of the release operation by using the `GET` HTTP method. When the `GET` HTTP method with this URL returns *204 No Content*, the operation has been completed. Following is an example of such request.

```
GET /ReleaseInventoryIssue/status/2dd24788-3f54-4673-a040-bf55b6d04a00 HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/InventoryIssue
Accept: application/json
Content-Type: application/json
```

Related Links

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

InventoryQuantityAvailable

This chapter presents code examples showing requests that use the `InventoryQuantityAvailable` entity.

Retrieve the Available Quantity of an Inventory Item

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve from Acumatica ERP the available quantity of an inventory item. This quantity is provided by the Available Quantity by Inventory Item (GI640590) generic inquiry, which is available for the external systems through the `InventoryQuantityAvailable` entity of the *Default/24.200.001* endpoint.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve the aggregated quantity of an inventory item through the contract-based REST API. You will retrieve the quantity of the *APJAM08* inventory item that was available on June 7, 2024.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Results HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/InventoryQuantityAvailable
Accept: application/json
Content-Type: application/json

{
    "InventoryID": { "value": "APJAM08" },
    "LastModifiedDateTime": { "value": "6/7/2024" }
}
```

Usage Notes

You use the following fields of the `InventoryQuantityAvailable` entity:

- `InventoryID`: The identifier of the inventory item for which the available quantity is retrieved
- `LastModifiedDateTime`: The date and time for which the available quantity of the inventory item is retrieved

InventorySummaryInquiry

This chapter presents code examples showing requests that use the `InventorySummaryInquiry` entity.

Get a Summary of an Inventory Item

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the summary information about an inventory item from *Inventory Summary* (IN401000) inquiry form.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

You can use the following request example to retrieve the summary information about the *SIMCARD* inventory item through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Results HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/InventorySummaryInquiry
Accept: application/json
Content-Type: application/json

{
   "InventoryID": { "value": "SIMCARD" }
```

```
}
```

Invoice

This chapter presents code examples showing requests that use the `Invoice` entity.

Create an Invoice with Tax Parameters Overridden

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create invoices with the tax parameters overridden.

System Preparation

Before you test the code below, do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

To further prepare the system, you need to create a tax and a tax category that uses this tax, and modify the existing tax zone so that it will use the new tax. You perform these tasks as follows:

If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

1. On the *Taxes* (TX205000) form, create a tax with the tax schedule as follows:
   a. Create a tax with the following parameters:
      - **Tax ID**: *PST*
      - **Description**: *Tax ID for testing*
      - **Tax Type**: *VAT*
      - **Include in VAT Taxable Total**: Selected
      - **Exclude from Tax on Tax Calculation**: Selected
      - **Tax Agency**: *CANADABC*
   b. On the **Tax Schedule** tab of the form, add two rows with the following data.

Column	First Row	Second Row
Start Data	*1/1/1900*	*1/1/1900*

Column	First Row	Second Row
Tax Rate	*5*	*5*
Min. Taxable Amount	*0*	*0*
Max. Taxable Amount	*0*	*0*
Reporting Group	*Input Tax (purchases)*	*Output Tax (sales)*
Group Type	*Input*	*Output*

c. Click **Save**.

2. On the *Tax Categories* (TX205500) form, create the *TEST* tax category with the *Tax category for test* description and add the *CAGST* and *PST* taxes to the tax category.
3. On the **Applicable Taxes** tab of the *Tax Zones* (TX206000) form, add the *PST* tax to the *CANADABC* tax zone.

Request

You can use the following example of an HTTP request to create an invoice with the tax parameters that are different from those registered in the system.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details,TaxDetails HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Invoice
Accept: application/json
Content-Type: application/json

{
    "CreatedDateTime": {
        "value": "1900-01-01T03:00:00+03:00"
    },
    "Customer": {
        "value": "AACUSTOMER"
    },
    "Date": {
        "value": "2022-01-23T00:00:00+03:00"
    },
    "Description": {
        "value": "Invoice with Taxes"
    },
    "Details": [
        {
            "Account": {
                "value": "40000"
            },
            "Amount": {
                "value": 1000
            },
```

```
            "Branch": {
                "value": "PRODWHOLE"
            },
            "ExtendedPrice": {
                "value": 1000
            },
            "InventoryID": {
                "value": "CONSULTING"
            },
            "LineNbr": {
                "value": 1
            },
            "ProjectTask": {},
            "Qty": {
                "value": 10
            },
            "TransactionDescription": {
                "value": "Project Consulting"
            },
            "UnitPrice": {
                "value": 100
            },
            "UOM": {
                "value": "HOUR"
            }
        }
    ],
    "Hold": {
        "value": false
    },
    "IsTaxValid": {
        "value": true
    },
    "LocationID": {
        "value": "MAIN"
    },
    "PostPeriod": {
        "value": "012022"
    },
    "TaxDetails": [
        {
            "TaxID": {
                "value": "CAGST"
            },
            "TaxableAmount": {
                "value": 1000.0
            },
            "TaxAmount": {
                "value": 150.0
            },
            "ExpenseAmount": {
                "value": 0.0
            }
        },
        {
            "TaxID": {
                "value": "PST"
```

```
            },
            "TaxableAmount": {
                "value": 1000.0
            },
            "TaxAmount": {
                "value": 150.0
            }
        }
    ],
    "Type": {
        "value": "Invoice"
    },
    "custom": {
        "CurrentDocument": {
            "TaxZoneID": {
                "type": "CustomStringField",
                "value": "CANADABC"
            }
        }
    }
}
```

In the request above, you have set the `IsTaxValid` value to *true* to use in the invoice the tax parameters specified in the `TaxDetails` field. If you do not specify this value, the tax parameters that are registered in the system will be used instead. In particular, the tax amounts equal to *50* will be set both for the *CAGST* and *PST* taxes in the created bill, and the *CABCPST* tax of the amount *70* will be applied as well.

Usage Notes

If the `IsTaxValid` field is specified in the request body, its value is set to *true*, and an external tax zone is specified, then the tax details from the request body are used without modification, and no other taxes are applied.

If the `IsTaxValid` field is specified in the request body, its value is set to *true*, and an internal tax zone is specified, then the tax details from the request body are compared to those calculated by the system. If the taxes specified in the request body are present among those calculated by the system, then the tax details from the request body are used without modification and no other taxes are applied.

If the `IsTaxValid` field is not specified in the request body or its value is not set to *true*, tax calculation is performed by the system.

Retrieve the List of Invoices

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the list of available invoices.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve the full list of invoices through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Invoice
Accept: application/json
Content-Type: application/json
```

Release an AR Invoice

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can release an AR invoice.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Invoices and Memos* (AR301000) form, create an invoice, save it, and click **Remove Hold**.
5. In the database, in the `ARInvoiceNbr` table, learn the `RefNoteID` value that corresponds to the reference number of the created invoice, which is stored in the `RefNbr` column.

Request

You can use the following request example to invoke the release operation for an AR invoice through the contract-based REST API. You specify the `RefNoteID` value of the invoice in the `id` field of the request body.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /ReleaseInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Invoice
Accept: application/json
Content-Type: application/json

{
    "entity": { "id": "8beb2af9-fa58-ec11-9e16-9828a61840c3" }
}
```

Related Links

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

Specify the Tax Zone for an Invoice

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can specify the tax zone for an invoice.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to set the tax zone of the *000097* invoice (whose `id` is *8deb6bf9-2072-eb11-b83e-00155d408001*) to *AVALARA* through the contract-based REST API. This request also clears the **Don't Print** box (in the **Print and Email Options** section of the **Financial** tab) of the *Invoices and Memos* (AR301000) form for the invoice.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$custom=CurrentDocument.TaxZoneID,CurrentDocument.DontPrint HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Invoice
Accept: application/json
Content-Type: application/json

{
    "id": "8deb6bf9-2072-eb11-b83e-00155d408001",
    "custom": {
        "CurrentDocument": {
```

```
            "TaxZoneID": {
                 "value": "AVALARA"
            },
             "DontPrint": {
                "value": false
            }
        }
    }
}
```

ItemWarehouse

This chapter presents code examples showing requests that use the `ItemWarehouse` entity.

Override Values in the Item–Warehouse Details

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can specify item–warehouse details, mark particular fields as overridden, and set new values for these fields.

System Preparation

Before you test the code below, you do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management*, *Inventory*, and *Multiple Warehouses* features are enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name that you specified when you created the instance and the *HEADOFFICE* branch.

Request

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

You can use the following example of an HTTP request to specify item–warehouse details for the *APPLES* inventory item and the *RETAIL* warehouse. This request marks the `MaxQty`, `ReorderPoint`, and `SafetyStock` fields as overridden and specifies their new values. The specification of the new values for the `MSRP`, `ReplenishmentMethod`, `ReplenishmentSource`, `Seasonality`, `PreferredLocation`, `PreferredVendor`, and `ServiceLevel` fields in the request body has no effect because these fields are not marked as overridden.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ItemWarehouse
Accept: application/json
Content-Type: application/json

{
    "InventoryID": { "value": "APPLES" },
    "OverrideMaxQty": { "value": true },
    "MaxQty": { "value": 2222 },
    "OverridePrice": { "value": false },
    "MSRP": { "value": 600 },
    "OverrideReorderPoint": { "value": true },
    "ReorderPoint": { "value": 14 },
    "ReplenishmentClass": { "value": "PURCHASE" },
    "OverrideReplenishmentSettings": { "value": false },
    "ReplenishmentMethod": { "value": "Min./Max." },
    "ReplenishmentSource": { "value": "Purchase" },
    "Seasonality": { "value": "NONE" },
    "OverrideSafetyStock": { "value": true },
    "SafetyStock": { "value": 244 },
    "PreferredLocation": { "value": "MAIN" },
    "PreferredVendor": { "value": "ALLFRUITS" },
    "ServiceLevel": { "value": 87 },
    "WarehouseID": { "value": "RETAIL" }
}
```

JournalTransaction

This chapter presents code examples showing requests that use the `JournalTransaction` entity.

Create a GL Transaction with a Project Code That Does Not Produce a Project Transaction

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can import to Acumatica ERP general ledger transactions with project codes that do not produce project transactions.

To create a general ledger transaction with a project code that does not produce a project transaction, you set the `IsNonPM` field of the `JournalTransaction` entity to *true*.

This setting could be used in the following user scenario: A construction company has built an integrated solution of Acumatica ERP with an external payroll system. The external payroll system calculates payoffs, including benefits, additions, deductions, and taxes. Once a week, the construction company needs to import general ledger transactions with project information from this payroll system to Acumatica ERP, where they are verified and released. The construction company doesn't want to update the project subledger in Acumatica ERP with the information from general ledger transactions (for example, if the standard labor costs have already been posted to the project subledger from time entries).

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a general ledger transaction through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/JournalTransaction
Accept: application/json
Content-Type: application/json

{
  "Module" : {"value" : "GL"},
  "TransactionDate" : {"value" : "2024-02-15T00:00:00"},
  "Description" : {"value" : "Transaction description"},
  "BranchID" : {"value" : "HEADOFFICE"},
  "Details" : [
   {
    "BranchID" : {"value" : "HEADOFFICE"},
    "Account" : {"value" : "10200"},
    "CostCode" : {"value" : "00000"},
    "IsNonPM" : {"value" : true}
   }
  ]
}
```

Lead

This chapter presents code examples showing requests that use the `Lead` entity.

Create a Lead

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a lead in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a lead through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Lead
Accept: application/json
Content-Type: application/json

{
    "FirstName": {"value": "Brent"},
    "LastName": {"value": "Edds"},
    "Email": {"value": "brent.edds.test27@avantehs.com"}
}
```

Ledger

This chapter presents code examples showing requests that use the `Ledger` entity.

Retrieve the List of Ledgers

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve a list of ledgers. For details about the management of ledgers, see *Managing Ledgers*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.

2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

You can use the following HTTP request example to retrieve a list of the available ledgers. This request also retrieves information about the companies associated with each ledger and the branches that can post to each ledger.

```
GET ?$expand=Branches,Companies&$select=LedgerID,Branches/BranchID,Branches/BranchName,
    Branches/CompanyName,Companies/Company,Companies/CompanyName HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Ledger
Accept: application/json
Content-Type: application/json
```

Opportunity

This chapter presents code examples showing requests that use the `Opportunity` entity.

Create a Sales Order from an Opportunity

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a sales order from an opportunity in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a sales order from the *000011* opportunity through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /CreateOpportunitySalesOrder HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Opportunity
Accept: application/json
Content-Type: application/json

{
    "entity":{
        "OpportunityID": {"value":"000011"}
    },
    "parameters":{
        "OrdereType": {"value":"SO"},
        "RecalculatePricesAndDiscounts": {"value":"false"}
    }
}
```

Create a Business Account from an Opportunity

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a business account from an opportunity in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a business account from the *000002* opportunity through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /CreateAccountFromOpportunity HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Opportunity
Accept: application/json
Content-Type: application/json

{
    "entity": { "OpportunityID": {"value":"000002"},
        "custom": {
            "AccountInfo": {
                "BAccountID": {  "value": "11111" },
                "AccountName": { "value": "asdqwe" }
            },
            "ContactInfo": {
                "LastName": { "value": "test" }
            }
        }
    }
}
```

PayGroup

This chapter presents code examples showing requests that use the `PayGroup` entity.

Create a Pay Group

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a pay group in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.

Request

You can use the following request example to create a pay group through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PayGroup
Accept: application/json
Content-Type: application/json

{
    "PayGroupID": { "value": "NewGroup" },
    "PayGroupName": { "value": "New Group Name" },
    "BenefitExpenseAccount": { "value": "69600" },
    "BenefitExpenseSub": { "value": "000000" },
    "BenefitLiabilityAccount": { "value": "20300" },
    "BenefitLiabilitySub": { "value": "000000" },
    "DeductionLiabilityAccount": { "value": "20300" },
    "DeductionLiabilitySub": { "value": "000000" },
    "EarningsAccount": { "value": "51000" },
    "EarningsSub": { "value": "000000" },
    "TaxExpenseAccount": { "value": "65100" },
    "TaxExpenseSub": { "value": "000000" },
    "TaxLiabilityAccount": { "value": "20300" },
    "TaxLiabilitySub": { "value": "000000" },
    "IsDefault": { "value": false }
}
```

Payment

This chapter presents code examples showing requests that use the `Payment` entity.

Create a Payment for an Invoice and a Sales Order

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create an accounts receivable payment for an invoice and sales order in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create an AR payment for the *000002* invoice and *000036* sales order through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
Content-Type: application/json

{
    "Branch": { "value": "HEADOFFICE" },
    "CashAccount": { "value": "10200WH" },
    "CurrencyID": { "value": "USD" },
    "CustomerID": { "value": "GOODFOOD" },
    "Description": { "value": "Creating Payment for different doc types" },
    "PaymentMethod": { "value": "CHECK" },
    "Type": { "value": "Payment" },
    "DocumentsToApply": [
        {
            "DocType": { "value": "INV" },
            "DocLineNbr": { "value": "1" },
            "ReferenceNbr": { "value": "000002" }
        }
    ],
    "OrdersToApply": [
        {
            "OrderType": { "value": "SO" },
            "OrderNbr": { "value": "000036" }
        }
    ]
}
```

Create a Payment with a Credit Card Transaction Imported from Another System

You can use the contract-based REST API to make an external system create an accounts receivable payment in Acumatica ERP. This AR payment can include a credit card transaction imported from another system. You can view the created payments on the *Payments and Applications* (AR302000) form.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create an AR payment with a credit card transaction imported from another system through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT /Payment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {
        "value": "HMBAKERY"
    },
    "PaymentAmount": {
        "value": 123.0000
    },
    "PaymentMethod": {
        "value": "VISA"
    },
    "CreditCardTransactionInfo": [
        {
            "TranNbr": {
                "value": "112224543839510314532923"
            },
            "TranType": {
                "value": "Authorize Only"
            },
            "NeedsValidation": {
                "value": true
            },
            "AuthNbr": {
                "value": "ebd4c8"
            }
        }
```

```
    ]
}
```

Usage Notes

You specify the following information about a credit card transaction in the request body:

- `TranNbr`: The transaction ID from the credit card processing center.
- `TranType`: The credit card transaction type.
- `NeedsValidation`: An indicator of whether validation of the credit card transaction should be deferred. If the value is *true*, the payment will receive the *Pending Processing* status. During further transaction processing, the system will send a request to the processing center for verification of the data received through the API.
- `AuthNbr`: The authorization number from the credit card processing center.

Related Links

- *Processing Credit Card Payments*

Release a Payment

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can release an accounts receivable payment in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following:

1. Create a payment for an invoice and sales order, as described in *Create a Payment for an Invoice and a Sales Order*.
2. On the *Payments and Applications* (AR302000) form, open the created payment. (Its reference number is assumed to be *000077* in this example.)
3. On the form toolbar, click **Remove Hold**, and specify the payment amount in the **Payment Amount** box.
4. Save the document.

Request

You can use the following request example to invoke the release process for the *000077* AR payment through the contract-based REST API. The reference number of the AR payment is specified in the `ReferenceNbr` field.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /Release HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
Content-Type: application/json

{
    "entity": {
```

```
        "Type": { "value": "Payment" },
        "ReferenceNbr": { "value": "000077" }
    }
}
```

Related Links

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

Retrieve Payments One by One

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve payments one by one from Acumatica ERP. In this case, the key fields—namely, the payment type and reference number—are specified in a request. When a single payment is retrieved by using the contract-based REST API, the following information can be retrieved as well: documents to apply, orders to apply, application history, credit card processing information, and credit card transaction information.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

You can use the following request example to retrieve the *000001* payment, along with detailed information about it, through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET Payment/000001?$select=ReferenceNbr,Type,Status,ApplicationDate,
ApplicationHistory/DisplayDocType,ApplicationHistory/DisplayRefNbr
&$expand=ApplicationHistory HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
Content-Type: application/json
```

PaymentMethod

This chapter presents code examples showing requests that use the `PaymentMethod` entity.

Create a Payment Method

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a payment method in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a payment method that uses cash or checks through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PaymentMethod
Accept: application/json
Content-Type: application/json

{
    "Active": { "value": false },
    "Description": { "value": "Test Method" },
    "IntegratedProcessing": { "value": false },
    "MeansOfPayment": { "value": "Cash/Check" },
    "PaymentMethodID": { "value": "TST" },
    "SettingsForPR": {
        "PRProcessing": { "value": "Print Checks" },
        "Report": { "value": "PR641010" }
    },
    "UseInPR": { "value": true },
    "UseInAP": { "value": false },
    "UseInAR": { "value": false },
    "RequireRemittanceInformationforCashAccount": { "value": false },
    "AllowedCashAccounts": [
        {
            "CashAccount": { "value": "10200WH" },
            "Description": { "value": "Company Checking Account" },
```

```
            "Branch": { "value": "HEADOFFICE" },
            "APDefault": { "value": false },
            "APLastRefNbr": { "value": "3202" },
            "APSuggestNextNbr": { "value": false },
            "UseInPR": { "value": true }
        }
    ]
}
```

PayPeriod

This chapter presents code examples showing requests that use the `PayPeriod` entity.

Create a Pay Period Schedule

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a pay period schedule in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.

Request

You can use the following request example to create a pay period schedule through the contract-based REST API. This pay period schedule is for 2025 and has 52 weekly pay periods.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PayPeriod
Accept: application/json
Content-Type: application/json

{
    "NumberOfPeriods": { "value": 53 },
```

```
    "Override": { "value": true },
    "PayGroup": { "value": "WEEKLY" },
    "Year": { "value": "2025" }
}
```

PayrollBatch

This chapter presents code examples showing requests that use the `PayrollBatch` entity.

Create a Payroll Batch

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a payroll batch in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.
5. On the *Pay Groups* (PR205000) form, make sure that the *MONTHLY* pay group exists.
6. On the *Pay Periods* (PR201000) form, create the *03-2024* pay period, or make sure that it already exists.

Request

You can use the following request example to create a payroll batch for the *MONTHLY* pay group and *03-2023* pay period through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PayrollBatch
Accept: application/json
Content-Type: application/json

{
    "PayrollType": { "value": "Regular" },
    "PayGroup": { "value": "MONTHLY" },
```

```
    "Description": { "value": "Test New Batch" },
    "Hold": { "value": false },
    "PayPeriod": { "value": "032024" }
}
```

PayrollUnionLocal

This chapter presents code examples showing requests that use the `PayrollUnionLocal` entity.

Create a Union

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a union in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.

Request

You can use the following request example to create a union associated with the *ALLFRUITS* vendor through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PayrollUnionLocal
Accept: application/json
Content-Type: application/json

{
    "Active": { "value": false },
    "Description": { "value": "Test Union" },
    "PayrollUnionLocalID": { "value": "TST" },
    "Vendor": { "value": "ALLFRUITS" }
```

```
}
```

PayrollWCCCode

This chapter presents code examples showing requests that use the `PayrollWCCCode` entity.

Create a Workers' Compensation Class Code

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a workers' compensation class code in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.

Request

You can use the following request example to create a workers' compensation class code through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PayrollWCCCode
Accept: application/json
Content-Type: application/json

{
    "Country": { "value": "US" },
    "WCCCodes": [
        {
            "Active": { "value": true },
            "Description": { "value": "Test Code" },
            "WCCCode": { "value": "2222" }
        }
    ]
```

```
}
```

ProFormaInvoice

This chapter presents code examples showing requests that use the `ProFormaInvoice` entity.

Send a Pro Forma Invoice by Email

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create pro forma invoices and send them by email. For details about pro forma invoices, see *Pro Forma Invoices: General Information*.

System Preparation

Sending a pro forma invoice by email is a final stage of a more common task of creating and sending a pro forma invoice by email.

If you were performing the overall process of creating and sending a pro forma invoice by email, you (or another employee) would perform the following general steps:

1. *Create a Project from a Project Template*
2. *Make a Project Active*
3. *Specify the Next Billing Date for a Project*
4. *Retrieve a Project Task*
5. *Activate a Project Task*
6. *Specify the Progress of a Project Task*
7. *Invoke Project Billing*
8. *Retrieve the List of Pro Forma Invoices of a Project*
9. *Send a Pro Forma Invoice by Email*

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Requests

You can use the following sequence of examples of HTTP requests to send a pro forma invoice by email through the contract-based REST API. You will use the `RefNbr` value of the pro forma invoice that is retrieved in *Retrieve the List of Pro Forma Invoices of a Project*.

> In the request examples below, <Acumatica ERP instance URL> is the URL of the Acumatica ERP instance (such as https://my.acumatica.com/MyInstance). You can omit the instance name in the URL (such as https://my.acumatica.com) if the instance is installed in the root of the website.

Do the following:

1. Send the pro forma invoice by email.

```
POST /EmailProFormaInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProFormaInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
    "RefNbr": {
        "value": "000019"
    }
  }
}

HTTP/1.1 202 Accepted
Location: [/<Acumatica ERP instance URL>]/entity/Default/24.200.001/
    ProFormaInvoice/EmailProFormaInvoice/status/a4caa455-0eed-4c11-a5a9-2a8333e53db1
```

2. Monitor the status of the operation until the system returns the *204 No Content* code.

```
GET /EmailProFormaInvoice/status/a4caa455-0eed-4c11-a5a9-2a8333e53db1 HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProFormaInvoice
Accept: application/json
Content-Type: application/json

HTTP/1.1 204 No Content
```

Usage Notes

For a pro forma invoice to be created from the project, the project must have the `Customer`, `BillingRule`, `BillingPeriod`, and `NextBillingDate` values specified, and must have the *Active* status. Because of data validation in the project, `NextBillingDate` cannot be specified for a project with the *In Planning* status, and you cannot change the customer in a project with the *Active* status. Therefore, these settings can be specified only in multiple API calls.

Related Links

- *Pro Forma Invoices: General Information*

Project

This chapter presents code examples showing requests that use the `Project` entity.

Create a Project from a Project Template

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a project from a project template. This step can be part of the larger process of creating and sending a pro forma invoice by email, which is described in *Send a Pro Forma Invoice by Email*.

For details about project templates, see *Project Templates and Common Tasks: General Information*.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following example of an HTTP request to create a project from the *PROGRESS* project template and specify the `Customer`, `BillingRule`, and `BillingPeriod` settings of the project through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

{
    "ProjectID" : {"value" : "TESTPR3"},
    "ProjectTemplateID" : {"value" : "PROGRESS"},
    "Customer" : {"value" : "COFFEESHOP"},
    "BillingAndAllocationSettings" :
    {
        "BillingRule" : {"value" : "PROGRESS"},
        "BillingPeriod" : {"value" : "Month"},
    }
}
```

Related Links

- *Pro Forma Invoices: General Information*

- *Project Billing PreparationBilling with a Direct AR Invoice: General Information*

Make a Project Active

For a pro forma invoice to be created from a project, the project must have `Customer`, `BillingRule`, `BillingPeriod`, and `NextBillingDate` specified, and must have the *Active* status. If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create make a project active. This step can be part of the larger process of creating and sending a pro forma invoice by email, which is described in *Send a Pro Forma Invoice by Email*.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
5. Implement the *Create a Project from a Project Template* example as a prerequisite to the current example. The project must exist before you can make it active.

Request

You can use the following example of an HTTP request to make the *TESTPR3* project active through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

{
    "ProjectID" : {"value" : "TESTPR3"},
    "Hold" : {"value" : false}
}
```

Specify the Next Billing Date for a Project

For a pro forma invoice to be created from a project, the project must have `Customer`, `BillingRule`, `BillingPeriod`, and `NextBillingDate` specified, and must have the *Active* status. If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can specify various settings for a project, such as the next billing date. This step can be part of the larger process of creating and sending a pro forma invoice by email, which is described in *Send a Pro Forma Invoice by Email*.

The next billing date of a project is specified on the **Summary** tab (the **Billing and Allocation Settings** section) of the *Projects* (PM301000) form.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
5. Implement the *Make a Project Active* example as a prerequisite to the current example. Because of data validation in the project, `NextBillingDate` is specified for the project after it is assigned the *Active* status.

Request

You can use the following example of an HTTP request to specify the next billing date for a project.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=BillingAndAllocationSettings HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

{
    "ProjectID" : {"value" : "TESTPR3"},
    "BillingAndAllocationSettings" :
    {
        "NextBillingDate" : {"value" : "2024-06-15"}
    }
}
```

Related Links

- *Project Billing PreparationBilling with a Direct AR Invoice: General Information*

Invoke Project Billing

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can invoke project billing. This step can be part of the larger process of creating and sending a pro forma invoice by email, which is described in *Send a Pro Forma Invoice by Email*.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name that you specified when you created the instance and the *HEADOFFICE* branch.
5. Implement the *Specify the Progress of a Project Task* example as a prerequisite to the current example.

Requests

You can use the following sequence of examples of HTTP requests to run project billing and check the completeness of this operation through the contract-based REST API:

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

1. Invoke project billing to create a pro forma invoice.

```
POST /RunProjectBilling HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

{
  "entity" : {
    "ProjectID": {
        "value": "TESTPR3"
    }
  }
}

HTTP/1.1 202 Accepted
Location: [/<Acumatica ERP instance URL>]/entity/Default/24.200.001/
```

```
    Project/RunProjectBilling/status/6952c6d1-04be-4330-a26e-c6b855ba332c
```

2. Monitor the status of the operation until the system returns the *204 No Content* code.

```
GET /RunProjectBilling/status/6952c6d1-04be-4330-a26e-c6b855ba332c HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

HTTP/1.1 204 No Content
```

Related Links

- *Pro Forma Invoices: General Information*
- *Project Billing PreparationBilling with a Direct AR Invoice: General Information*

Retrieve the List of Pro Forma Invoices of a Project

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the list of the pro forma invoices of a project. This step can be part of the larger process of creating and sending a pro forma invoice by email, which is described in *Send a Pro Forma Invoice by Email*.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
5. Implement the *Invoke Project Billing* example as a prerequisite to the current example, so that the *TESTPR3* project will be created with a pro forma invoice.

Request

You can use the following example of an HTTP request to retrieve the list of pro forma invoices of a project through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /TESTPR3?$expand=Invoices HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
```

```
Content-Type: application/json
```

To obtain the list of pro forma invoices of the project with the *TESTPR3* project ID, you can also use the `GET` request for the `ProFormaInvoice` entity with the `$filter=ProjectID eq 'TESTPR3'` parameter.

ProjectBudget

This chapter presents code examples showing requests that use the `ProjectBudget` entity.

Specify the Progress of a Project Task

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can specify the progress of a project task. This step can be part of the larger process of creating and sending a pro forma invoice by email, which is described in *Send a Pro Forma Invoice by Email*.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
5. Implement the *Make a Project Active* example as a prerequisite to the current example. In this prerequisite example, you will prepare the project task for which the progress will be specified.

Request

You can use the following example of an HTTP request to specify the progress of a project task through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProjectBudget
Accept: application/json
Content-Type: application/json

{
  "ProjectTaskID" : {"value" : "PHASE1"},
```

```
  "ProjectID" : {"value" : "TESTPR3"},
  "InventoryID" : {"value" : "<N/A>"},
  "Completed" : {"value" : 25},
  "PendingInvoiceAmount" : {"value" : 725}
}
```

ProjectTask

This chapter presents code examples showing requests that use the `ProjectTask` entity.

Retrieve a Project Task

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve a project task that meets some conditions. This retrieval step can be part of a larger process of creating and sending a pro forma invoice by email, which is described in *Send a Pro Forma Invoice by Email*.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following example of an HTTP request to retrieve a project task with the *TESTPR3* project ID and the *PHASE1* task ID through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$filter=ProjectID%20eq%20'TESTPR3'%20and%20ProjectTaskID%20eq%20'PHASE1' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProjectTask
Accept: application/json
Content-Type: application/json
```

Activate a Project Task

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can activate a project task. This activation task can be part of a more common task of creating and sending a pro forma invoice by email, which is described in *Send a Pro Forma Invoice by Email*.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
5. Perform the following requests to prepare the data for testing:
   a. *Create a Project from a Project Template*
   b. *Make a Project Active*
   c. *Specify the Next Billing Date for a Project*
   d. *Retrieve a Project Task*

Request

You can use the following example of an HTTP request to activate a project task through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /Activate HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProjectTask
Accept: application/json
Content-Type: application/json

{
    "entity":
    {
        "ProjectID": {
            "value": "TESTPR3"
        },
        "ProjectTaskID": {
            "value": "PHASE1"
        }
```

```
    },
    "parameters": {}
}
```

Related Links

- *Pro Forma Invoices: General Information*
- *Project Billing PreparationBilling with a Direct AR Invoice: General Information*

PTOBank

This chapter presents code examples showing requests that use the `PTOBank` entity.

Create a PTO Bank

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a paid time off (PTO) bank in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* and *US Payroll* features are enabled.
5. Modify the earning type code as follows:

   > If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

   a. On the *Earning Type Codes* (PR102000) form, open the *HL* earning type code (which represents public holidays).
   b. Change the settings of the earning type code as follows:
      - **Earning Type Category**: *Time Off*
      - **Regular Time Type Code**: *RG*
   c. Save your changes.

Request

You can use the following request example to create a holiday PTO bank through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=PayrollDefaults/WorkLocations,PTODefaults HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PTOBank
Accept: application/json
Content-Type: application/json

{
    "PTOBankID": { "value": "TST" },
    "Description": { "value": "Test PTO Bank" },
    "Active": { "value": "false" },
    "BankStartDate": { "value": "01/01/2019" },
    "DisbursingEarningCode": { "value": "HL" },
    "CreateFinTransactions": { "value": "true" }
}
```

PurchaseOrder

This chapter presents code examples showing requests that use the `PurchaseOrder` entity.

Create a Purchase Order

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create purchase orders. For details about the management of purchase documents, see *Managing Purchase Documents*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

You can use the following example of an HTTP request to create a purchase order and release it from hold at once.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseOrder
Accept: application/json
Content-Type: application/json

{
    "VendorID": { "value": "GOODFRUITS" },
    "Location": { "value": "MAIN" },
    "Details": [
        {
            "BranchID": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APPLES" },
            "OrderQty": { "value": 1 },
            "WarehouseID": { "value": "WHOLESALE" },
            "UOM": { "value": "LB" }
        }
    ],
    "Hold": { "value": false }
}
```

Create a Purchase Order with Tax Parameters Overridden

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create purchase orders. In the orders, these systems can override the tax details that are calculated by the system.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

You can use the following example of an HTTP request to create a purchase order and set the amount of the *NYSTATETAX* tax to *0*.

```
PUT ?$expand=Details,TaxDetails HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseOrder
Accept: application/json
Content-Type: application/json

{
    "Branch": { "value": "HEADOFFICE" },
    "Details": [
        {
            "BranchID": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APPLES" },
            "LineType": { "value": "Goods for IN" },
            "OrderQty": { "value": 20 },
            "OrderType": { "value": "RO" },
            "TaxCategory": { "value": "TAXABLE" },
            "UnitCost": { "value": 2.29 },
            "UOM": { "value": "LB" },
            "WarehouseID": { "value": "WHOLESALE" }
        }
    ],
    "Hold": { "value": true },
    "IsTaxValid":  { "value": true },
    "Location": { "value": "MAIN" },
    "TaxDetails": [
        {
            "TaxableAmount": { "value": 45.8 },
            "TaxAmount": { "value": 0 },
            "TaxID": { "value": "NYSTATETAX" },
            "TaxRate": { "value": 0 }
        }
    ],
    "Terms": { "value": "30D" },
    "Type": { "value": "Normal" },
    "VendorID": { "value": "ALLFRUITS" },
    "VendorTaxZone": { "value": "NYZONE" }
}
```

Usage Notes

If the `IsTaxValid` field is specified in the request body, its value is set to *true*, and an external tax zone is specified, then the tax details from the request body are used without modification, and no other taxes are applied.

If the `IsTaxValid` field is specified in the request body, its value is set to *true*, and an internal tax zone is specified, then the tax details from the request body are compared to those calculated by the system. If the taxes

specified in the request body are present among those calculated by the system, then the tax details from the request body are used without modification and no other taxes are applied.

If the `IsTaxValid` field is not specified in the request body or its value is not set to *true*, tax calculation is performed by the system.

PurchaseReceipt

This chapter presents code examples showing requests that use the `PurchaseReceipt` entity.

Create a Purchase Receipt

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create purchase receipts. For details about the management of purchase documents, see *Managing Purchase Documents*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

You can use the following example of HTTP requests to create a purchase receipt from the purchase order. The creation of a purchase order is described in *Create a Purchase Order*.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "VendorID": { "value": "GOODFRUITS" },
    "Location": { "value": "MAIN" },
    "Details": [
```

```
        {
            "POOrderNbr": { "value": "000030" },
            "POOrderType": { "value": "Normal" }
        }
    ]
}
```

Insert Lines with Allocations (with Location) in a PO Receipt

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create purchase receipts and specify the locations of items in them. For details about the management of purchase documents, see *Managing Purchase Documents*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following example of an HTTP request to create a purchase receipt and specify the locations of particular items.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details,Details/Allocations HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "Type": {"value": "Receipt"},
    "VendorID": {"value": "AAVENDOR"},
    "CreateBill": {"value": "False"},
    "Description": {"value": "Test receipt with allocations"},
    "Details": [
        {
            "InventoryID": {"value": "CONBABY1"},
```

```
            "ReceiptQty": {"value": "11"},
            "Allocations": [
                {
                    "Location": {"value": "DOCK"},
                    "Qty": {"value": "5"}
                },
                {
                    "Location": {"value": "R1S1"},
                    "Qty": {"value": "6"}
                }
            ]
        },
        {
            "InventoryID": {"value": "CONBOTTLE1"},
            "ReceiptQty": {"value": "15"},
            "Allocations": [
                {
                    "Location": {"value": "DROPSHIP"},
                    "Qty": {"value": "7"}
                },
                {
                    "Location": {"value": "RECEIVING"},
                    "Qty": {"value": "8"}
                }
            ]
        }
    ]
}
```

Insert Lines with Allocations (with Expiration Dates) in a PO Receipt

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create purchase receipts and specify the expiration dates of the items in them. For details about the management of purchase documents, see *Managing Purchase Documents*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

Request

You can use the following example of an HTTP request to create a purchase receipt and specify the expiration dates of particular goods.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details,Details/Allocations HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "Type": {"value": "Receipt"},
    "VendorID": {"value": "GOODFRUITS"},
    "CreateBill": {"value": "False"},
    "Description": {"value": "Test receipt with Expiration Date in Allocations"},
    "Details": [
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "ORANGES"},
            "ReceiptQty": {"value": 2},
            "Warehouse": {"value": "WHOLESALE"},
            "LotSerialNbr": {"value": ""},
            "ExpirationDate": {"value": ""},
            "Allocations": [
                {
                    "Location": {"value": "MAIN"},
                    "Qty": {"value": 1},
                    "LotSerialNbr": {"value": "a"},
                    "ExpirationDate": {"value": "2024-04-25"}
                },
                {
                    "Location": {"value": "MAIN"},
                    "Qty": {"value": 1},
                    "LotSerialNbr": {"value": "b"},
                    "ExpirationDate": {"value": "2024-04-27"}
                }
            ]
        }
    ]
}
```

Create a Purchase Return from a Purchase Receipt Record

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create purchase returns from purchase receipt records. For details about the management of purchase documents, see *Managing Purchase Documents*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following example of an HTTP request to create a purchase return by using the purchase receipt number (the `POReceiptNbr` field) and the purchase receipt line number (the `POReceiptLineNbr` field). Thus, a specific line of a purchase receipt will be added to the purchase return being created.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "Type": {"value": "Return"},
    "VendorID": {"value": "GOODFRUITS"},
    "CreateBill": {"value": "False"},
    "Description": {"value": "Test return (one PR line)"},
    "ProcessReturnWithOriginalCost": {"value": "True"},
    "Details": [
        {
            "POReceiptNbr": {"value": "000016"},
            "POReceiptLineNbr": {"value": "1"}
        }
    ]
}
```

Create a Purchase Return from a Purchase Receipt

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create purchase returns from purchase receipts. For details about the management of purchase documents, see *Managing Purchase Documents*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following example of an HTTP request to create a purchase return by using the purchase receipt number (the `POReceiptNbr` field). Thus, the whole purchase receipt will be used in the purchase return being created.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "Type": {"value": "Return"},
    "VendorID": {"value": "GOODFRUITS"},
    "CreateBill": {"value": "False"},
    "Description": {"value": "Test return (all PR lines)"},
    "ProcessReturnWithOriginalCost": {"value": "True"},
    "Details": [
        {
            "POReceiptNbr": {"value": "000013"}
        }
    ]
}
```

Create a Purchase Return for Particular Items

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create purchase returns by specifying items to return. For details about the management of purchase documents, see *Managing Purchase Documents*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following example of an HTTP request to create a purchase return by specifying items to return. In the request, no information about the original purchase receipts is specified.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "Type": {"value": "Return"},
    "VendorID": {"value": "GOODFRUITS"},
    "CreateBill": {"value": "False"},
    "Description": {"value": "Test return"},
    "ProcessReturnWithOriginalCost": {"value": "True"},
    "Details": [
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APPLES"},
            "Warehouse": {"value": "WHOLESALE"},
            "Location": {"value": "MAIN"},
            "ReceiptQty": {"value": "1"},
            "ExtendedCost": {"value": "3"}
        }
    ]
}
```

Create a Purchase Receipt in a Non-Base Currency

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a purchase receipt in a non-base currency. For details about the management of purchase documents, see *Managing Purchase Documents*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* and *Multicurrency Accounting* features are enabled.
5. Modify the *GOODFRUITS* vendor as follows:

   > If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

   a. On the *Vendors* (AP303000) form, select the *GOODFRUITS* vendor.
   b. On the **Financial** tab, select the following check boxes: **Enable Currency Override** and **Enable Rate Override**.
   c. In the **Curr. Rate Type** box, select *SPOT*.
   d. Save your changes.

Request

You can use the following example of an HTTP request to create a purchase receipt in a non-base currency by using the `CurrencyID` and `CurrencyRate` fields. In the example, you will override cost-related values (such as `UnitCost` or `ExtendedCost`).

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json
```

```
{
    "Type": {"value": "Receipt"},
    "VendorID": {"value": "GOODFRUITS"},
    "CreateBill": {"value": "False"},
    "CurrencyID": {"value": "EUR"},
    "CurrencyRate": {"value": "1.2"},
    "Description": {"value": "Test receipt in non-base currency and with new cost fields"},
    "Details": [
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APPLES"},
            "Warehouse": {"value": "WHOLESALE"},
            "ReceiptQty": {"value": 1.0},
            "ExpirationDate": {"value": "2021-04-25"},
            "UnitCost": {"value": "111"},
            "ExtendedCost": {"value": "333"}
        }
    ]
}
```

Create a Transfer Receipt with Allocations for a Transfer Order

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create transfer receipts with allocations in Acumatica ERP in a single API call. For details about the creation of transfer receipts, see *Two-Step Transfers: General Information*.

For a detailed description of a user scenario when transfer receipts with allocations can be created, see *Sales from Multiple Warehouses: General Information*.

System Preparation

Before you test the code below, you do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management*, *Inventory*, and *Multiple Warehouses* features are enabled.

To further prepare the system, you need to create a sales order with items allocated in different warehouses; to fulfill this sales order, you need to create a transfer order and transfer receipt. You perform these tasks as follows:

1. On the *Warehouses* (IN204000) form, open the *RETAIL* warehouse. On the **Locations** tab, clear the **Receiving Location** box. In the warning dialog box that is displayed, click **Yes**; click **Save**.
2. On the *Sales Orders* (SO301000) form, create a sales order with items as follows:
    a. Create a sales order of the *SO* order type for the *GOODFOOD* customer and *MAIN* location.
    b. On the **Details** tab of the form, click **Add Items** on the table toolbar.

c. In the **Inventory Lookup** dialog box, which opens, select the unlabeled check box in the line containing the *APJAM08* inventory ID and the *RETAIL* warehouse, and specify `12` in the **Qty. Selected** column. Click **Add & Close**.

d. On the **Details** tab, click the line the system has added to the table, and click **Line Details** on the table toolbar.

e. In the **Line Details** dialog box, which opens, in the only line, select the check box in the **Allocated** column. In the automatically added line, select the check box in the **Allocated** column, and specify *WHOLESALE* in the **Alloc. Warehouse** column; leave `1` in the **Quantity** column. Click **OK**.

f. Save the sales order.

3. On the More menu (under **Replenishment**), click **Create Transfer Order**.
4. On the *Create Transfer Orders* (SO509000) form, which opens, in the table, select the unlabeled check box for the transfer request that the system just created, and click **Process** on the form toolbar.
5. On the *Sales Orders* (SO301000) form, which opens for the created transfer order, click **Create Shipment** on the form toolbar. In the **Specify Shipment Parameters** dialog box, leave the parameter values as they are, and click **OK**.
6. On the *Shipments* (SO302000) form (which opens for the created shipment), on the form toolbar, click **Confirm Shipment** and then click **Update IN**.

Request

You can use the following example of an HTTP request to create a transfer receipt with allocations for the transfer order that was just created. Half of the *APJAM08* inventory item (0.5 units) will be moved to the *JS1* location, and the other half of the *APJAM08* inventory item will be moved to the *JS2* location.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details,Details/Allocations HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "Type": {
        "value": "Transfer Receipt"
    },
    "Warehouse": {
        "value": "RETAIL"
    },
    "Details": [
        {
            "TransferOrderType": {
                "value": "TR"
            },
            "TransferOrderNbr": {
                "value": "000064"
            },
            "TransferOrderLineNbr": {
                "value": "1"
            },
```

```
            "TransferShipmentNbr": {
                "value": "000059"
            },
            "ReceiptQty": {
                "value": "1"
            },
            "Allocations": [
                {
                    "Location": {
                        "value": "JS1"
                    },
                    "Qty": {
                        "value": 0.5
                    }
                },
                {
                    "Location": {
                        "value": "JS2"
                    },
                    "Qty": {
                        "value": 0.5
                    }
                }
            ]
        }
    ]
}
```

Release a Purchase Receipt

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can release a purchase receipt in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following request example to invoke the release process for the purchase receipt through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /ReleasePurchaseReceipt HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "entity":{ "id": {{id}} }
}
```

Usage Notes

You can learn the value of the `id` field for the needed purchase receipt by viewing the `POReceipt` database table.

SalesInvoice

This chapter presents code examples showing requests that use the `SalesInvoice` entity.

Remove a Sales Invoice from Hold

If you are using the REST API to integrate Acumatica ERP with an external system, this external system can remove a sales invoice from hold as a part of processing the sales invoice. For example, the external system can remove a sales invoice from hold in Acumatica ERP when a user reviews the sales invoice before confirming it in the external system.

In Acumatica ERP, the execution of the request described in this topic changes the value in the **Status** box on the *Invoices* (SO303000) form to *Balanced*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.
4. Clear the **Validate Document Totals on Entry** check box on the *Accounts Receivable Preferences* (AR101000) form. With this check box cleared, the invoice amount does not have to be entered in the **Amount** box on the *Invoices* (SO303000) form. (When the check box is selected, this box can be used to validate data during manual document entry.)
5. On the *Invoices* form, make sure that the invoice with the *INV000046* reference number exists and has the *On Hold* status.

Request

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

You can use the following sample HTTP request to remove the *INV000046* sales invoice from hold.

```
PUT /entity/Default/24.200.001/SalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

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

Invoke Release of an Invoice

If you are using the REST API to integrate Acumatica ERP with an external system, this external system can release a sales invoice. For example, the external system can release a sales invoice in Acumatica ERP when a user confirms the sales invoice in the external system.

In Acumatica ERP, the successful execution of the request described in this topic changes the value in the **Status** box on the *Invoices* (SO303000) form to *Open*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

4. Remove the *INV000046* sales invoice from hold by completing the *Remove a Sales Invoice from Hold* activity.

Request

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

You can use the following sample HTTP request to invoke the release operation for the *INV000046* sales invoice.

```
POST /entity/Default/24.200.001/SalesInvoice/ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

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

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

A response to the `POST` request with the `202 Accepted` status has the `Location` header, which contains a URL that you should use to check the status of the operation by using the `GET` HTTP method. When the `GET` HTTP method with this URL returns `204 No Content`, the operation is completed.

Related Links

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

Retrieve the Status of the Release Operation

If you are using the REST API to integrate Acumatica ERP with an external system, this external system can release a sales invoice. After the release operation has been invoked through the REST API, you need to check the status of the operation, as described in this topic.

In Acumatica ERP, after the release operation has been invoked on the *Invoices* (SO303000) form, the system displays the processing status on the form.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make

sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.
4. Remove the *INV000046* sales invoice from hold by completing the *Remove a Sales Invoice from Hold* activity.
5. Invoke the release operation by completing the *Invoke Release of an Invoice* activity.

Request

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Suppose that the `Location` header of the response for the request that invoked the release operation contains the following URL: *<Acumatica ERP instance URL>/entity/Default/24.200.001/SalesInvoice/ReleaseSalesInvoice/status/f84addec-dddd-4ec7-be5d-07a943263351*. You can use the following sample HTTP request to check the status of the release operation.

While the status returned by the request is `202 Accepted`, the operation is in progress. You should have a delay between the checks of the status of the operation so that the performance of the application is not impaired. The operation is completed when the `GET` request to this URL returns *204 No Content*.

```
GET /entity/Default/24.200.001/SalesInvoice
/ReleaseSalesInvoice/status/f84addec-dddd-4ec7-be5d-07a943263351 HTTP/1.1
Host: <Acumatica ERP instance URL>
Accept: application/json
```

Check the Status of a Sales Invoice

If you are using the REST API to integrate Acumatica ERP with an external system, this external system may need to check the status of a sales invoice. For example, the external system may need to check the status of a sales invoice after the release operation has been performed for it.

In Acumatica ERP, after the release operation has been completed on the *Invoices* (SO303000) form, the sales invoice has the *Open* status, which is specified in the **Status** box.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.
4. Remove the *INV000046* sales invoice from hold by completing *Remove a Sales Invoice from Hold*.
5. Invoke the release operation by completing *Invoke Release of an Invoice*.
6. Make sure the operation has completed successfully by completing *Retrieve the Status of the Release Operation*.

Request

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

You can use the following sample HTTP request to check the status of the sales invoice after the release operation.

```
GET /entity/Default/24.200.001/SalesInvoice/Invoice/INV000046
    ?$select=ReferenceNbr,Type,Status HTTP/1.1
Host: <Acumatica ERP instance URL>
Accept: application/json
Content-Type: application/json
```

Create a Credit Memo

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a credit memo in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following request example to create a credit memo through the contract-based REST API. This credit memo will be created to return all units of the item listed in the *000005* invoice, all units of the item listed in line *1* of the *000008* invoice, and 50 units of the item listed in line *2* of the *000012* invoice.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "GOODFOOD" },
    "Details": [
        {
            "OrigInvNbr": { "value": "000005" },
            "OrigInvType": { "value": "Invoice" }
        },
        {
            "OrigInvNbr": { "value": "000008" },
            "OrigInvType": { "value": "Invoice" },
            "OrigInvLineNbr": { "value": 1 }
        },
        {
            "OrigInvNbr": { "value": "000012" },
            "OrigInvType": { "value": "Invoice" },
            "OrigInvLineNbr": { "value": 2 },
            "Qty": { "value": 50 }
        }
    ],
    "LocationID": { "value": "MAIN" },
    "Project": { "value": "X" },
    "Type": { "value": "Credit Memo" }
}
```

Create a Direct Sales Invoice

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a direct sales invoice in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

   > If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

5. Perform Step 1, 2, and 3 in *Entry of a Direct Sales Invoice*.

Request

You can use the following request example to create an invoice for the *FRUITICO* customer through the contract-based REST API. This invoice will contain sold stock items and ordered services. Also, billing settings will be specified, and a payment will be applied to the invoice.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=ApplicationsInvoice,Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "Type": {"value": "Invoice"},
    "Hold": {"value": false},
    "Details":
    [
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "CHERJAM32"},
            "Qty": {"value": 1},
            "UOM": {"value": "PIECE"},
            "LotSerialNbr": {"value": "JM2301290000"}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APJAM32"},
            "Qty": {"value": 2},
            "UOM": {"value": "BOX"}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "ADVERT"},
            "Qty": {"value": 1},
            "UnitPrice": {"value": 1000},
            "UOM": {"value": "DAY"}
        }
    ],
    "BillingSettings":
    {
        "BillToAddressOverride": {"value": true},
        "BillToAddress":
```

```
        {
            "AddressLine1": {"value": "Fillmore Str"},
            "City": {"value": "San Francisco"},
            "State": {"value": "CA"}
        }
    },
    "ApplicationsInvoice":
    [
        {
            "DocType": {"value": "Payment"},
            "AdjustingDocReferenceNbr": {"value": "000076"},
            "AmountPaid": {"value": 1235.27}
        }
    ]
}
```

Create a Sales Invoice in a Non-Base Currency

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a sales invoice in a non-base currency in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.
3. Enable currency overriding and rate overriding for the *FRUITICO* customer as described in *Enable Currency Overriding and Rate Overriding for a Customer*. Instead of executing the request that is provided in *Enable Currency Overriding and Rate Overriding for a Customer*, you can configure the *FRUITICO* customer as follows:
   a. On the *Customers* (AR303000) form, open the *FRUITICO* customer.
   b. In the **Financial Settings** section of the **Financial** tab, select the **Enable Currency Override** check box, select the **Enable Rate Override** check box, and specify *SPOT* in the **Curr. Rate Type** box.
   c. Save your changes.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
5. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following request example to create an invoice in euros for the *FRUITICO* customer through the contract-based REST API. This invoice contains the following details:

- Two *APJAM96* items that are being sold to the customer
- An *APJAM08* item that the customer is returning

- A refund of one hour of the *CLEANING* service, which had been paid for and is no longer needed

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "Type": {"value": "Invoice"},
    "Currency": {"value": "EUR"},
    "Hold": {"value": false},
    "Details":
    [
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APJAM96"},
            "WarehouseID": {"value": "WHOLESALE"},
            "Qty": {"value": 2},
            "UOM": {"value": "JBOX"},
            "Location": {"value": "L3R1S2"}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APJAM08"},
            "WarehouseID": {"value": "WHOLESALE"},
            "Location": {"value": "L1R1S2"},
            "Qty": {"value": -1},
            "UOM": {"value": "PIECE"}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "CLEANING"},
            "Qty": {"value": -1},
            "UOM": {"value": "HOUR"}
        }
    ]
}
```

SalesOrder

This chapter presents code examples showing requests that use the `SalesOrder` entity.

Retrieve a List of Sales Orders with Details and Related Shipments

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve a list of sales orders along with their details and related shipments.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

You can use the following request example to retrieve the sales orders of the *C000000003* customer (along with their details and related shipments) through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$expand=Details,Shipments&$filter=CustomerID eq 'C000000003'&
    $select=OrderNbr,OrderType,CustomerID,CustomerOrder,Details/InventoryID,
    Details/OrderQty,Details/UnitPrice,Date,OrderedQty,OrderTotal,
    Shipments/InvoiceNbr,Shipments/ShipmentNbr HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Retrieve a List of Sales Orders in Multiple Batches

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve a list of sales orders in batches. This way of retrieving data can be chosen for better performance: If there is a large amount of data, only part of it can be retrieved at a time. For more information, see *$top Parameter* and *$skip Parameter*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

You can use the following request example to retrieve the sixth to tenth sales orders (in order of creation) of the *C000000003* customer through the contract-based REST API. (In fact, this request will retrieve the sixth to ninth sales orders because only nine orders belong to the *C000000003* customer.)

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$filter=CustomerID eq 'C000000003'&
    $select=OrderNbr,OrderType,CustomerID,OrderTotal&$top=5&$skip=5 HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Create a Sales Order with the Unit of Measure Specified

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create sales orders and specify the units of measure for the sales order items.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following example of an HTTP request to create a sales order of two small jars of jam. The unit of measure for the sales order item is *PIECE*.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details&$select=CustomerID,Details/Branch,Details/InventoryID,
    Details/OrderQty,OrderNbr HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "GOODFOOD" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APJAM08" },
            "OrderQty": { "value": 2 },
            "UOM": { "value": "PIECE" },
            "WarehouseID": { "value": "WHOLESALE" }
        }
    ]
}
```

Create a Sales Order with a Credit Card Payment

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create sales orders with credit card payments in a single API call. For details about the creation of sales orders, see *Reserving Payments for Sales Orders*.

> A sales order is created with a payment in one transaction. If a payment cannot be created, the sales order will not be created either.

System Preparation

Before you test the code below, do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make

sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Request

You can use the following example of an HTTP request to create a sales order with a credit card payment. The status of the successfully created payment for the sales order is *Pending Processing* and the payment reference is empty because no credit card payment has been authorized or captured.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details,Payments HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "WIDGETCC"},
    "Date": {"value": "2024-05-10T00:00:00"},
    "Description": {"value": "Internal CC Payment"},
    "Details": [
        {
            "Branch": {"value": "PRODWHOLE"},
            "DiscountAmount": {"value": 10.00},
            "ExtendedPrice": {"value": 500.00},
            "FreeItem": {"value": false},
            "InventoryID": {"value": "AACOMPUT01"},
            "OrderQty": {"value": 1.00},
            "UnitPrice": {"value": 500.00},
            "UOM": {"value": "EA"},
            "WarehouseID": {"value": "WHOLESALE"}
        },
        {
            "Branch": {"value": "PRODWHOLE"},
            "DiscountAmount": {"value": 10.00},
            "ExtendedPrice": {"value": 500.00},
            "FreeItem": {"value": false},
            "InventoryID": {"value": "AALEGO500"},
            "OrderQty": {"value": 50.00},
            "UnitPrice": {"value": 50.00},
            "UOM": {"value": "EA"},
            "WarehouseID": {"value": "WHOLESALE"}
        }
    ],
    "Hold": {"value": false},
    "LocationID": {"value": "MAIN"},
    "OrderType": {"value": "SO"},
    "Payments": [
        {
            "ApplicationDate": {"value": "2024-08-11T00:00:00+03:00"},
```

```
            "AppliedToOrder": {"value": 480.00},
            "CashAccount": {"value": "10600"},
            "PaymentAmount": {"value": 980.00},
            "PaymentMethod": {"value": "ACUPAYCC"}
        }
    ],
    "RequestedOn": {"value": "2024-05-10T00:00:00"}
}
```

Create a Sales Order with a Captured Credit Card Payment

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a sales order with payments in a single API call. For details about the creation of sales orders, see *Reserving Payments for Sales Orders*.

System Preparation

Before you test the code below, do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Request

You can use the following example of an HTTP request to create in one call a sales order with an internal credit card payment that will be captured. To capture a payment, you set the `Capture` parameter of the payment to *true*. The operation you create by this call is asynchronous.

The status of the successfully created payment for the sales order is *Open* because not full amount of the payment in the example is captured. The `PaymentRef` field of the payment contains the payment reference of the captured credit card payment.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Payments HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
```

```
    "CustomerID": {"value": "WIDGETCC"},
    "Date": {"value": "2023-05-10T00:00:00"},
    "Description": {"value": "Internal CC Payment (Capture)"},
    "Details": [
        {
            "Branch": {"value": "PRODWHOLE"},
            "DiscountAmount": {"value": 10.00},
            "ExtendedPrice": {"value": 500.00},
            "FreeItem": {"value": false},
            "InventoryID": {"value": "AACOMPUT01"},
            "OrderQty": {"value": 1.00},
            "UnitPrice": {"value": 500.00},
            "UOM": {"value": "EA"},
            "WarehouseID": {"value": "WHOLESALE"}
        },
        {
            "Branch": {"value": "PRODWHOLE"},
            "DiscountAmount": {"value": 10.00},
            "ExtendedPrice": {"value": 500.00},
            "FreeItem": {"value": false},
            "InventoryID": {"value": "AALEGO500"},
            "OrderQty": {"value": 50.00},
            "UnitPrice": {"value": 50.00},
            "UOM": {"value": "EA"},
            "WarehouseID": {"value": "WHOLESALE"}
        }
    ],
    "Hold": {"value": false},
    "LocationID": {"value": "MAIN"},
    "OrderType": {"value": "SO"},
    "Payments": [
        {
            "ApplicationDate": {"value": "2023-08-11T00:00:00+03:00"},
            "AppliedToOrder": {"value": 480.00},
            "CashAccount": {"value": "10600"},
            "PaymentAmount": {"value": 980.00},
            "PaymentMethod": {"value": "ACUPAYCC"},
            "Capture": {"value": true}
        }
    ],
    "RequestedOn": {"value": "2023-05-10T00:00:00"}
}
```

Create a Sales Order with an Authorized Credit Card Payment

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a sales order with payments in a single API call. For details about the creation of sales orders, see *Reserving Payments for Sales Orders*.

System Preparation

Before you test the code below, do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Request

You can use the following example of an HTTP request to create in one call a sales order with an internal credit card payment that will be authorized. To authorize a payment, you set the `Authorize` parameter of the payment to *true*. The operation you create by using this call is asynchronous.

The status of the successfully created payment for the sales order is *Pending Processing* because no credit card payment has been captured. The `PaymentRef` field of the payment contains the payment reference of the authorized credit card payment.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Payments HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "WIDGETCC"},
    "Date": {"value": "2023-05-10T00:00:00"},
    "Description": {"value": "Internal CC Payment (Capture)"},
    "Details": [
        {
            "Branch": {"value": "PRODWHOLE"},
            "DiscountAmount": {"value": 10.00},
            "ExtendedPrice": {"value": 500.00},
            "FreeItem": {"value": false},
            "InventoryID": {"value": "AACOMPUT01"},
            "OrderQty": {"value": 1.00},
            "UnitPrice": {"value": 500.00},
            "UOM": {"value": "EA"},
            "WarehouseID": {"value": "WHOLESALE"}
        },
        {
            "Branch": {"value": "PRODWHOLE"},
            "DiscountAmount": {"value": 10.00},
            "ExtendedPrice": {"value": 500.00},
            "FreeItem": {"value": false},
            "InventoryID": {"value": "AALEGO500"},
            "OrderQty": {"value": 50.00},
            "UnitPrice": {"value": 50.00},
```

```
            "UOM": {"value": "EA"},
            "WarehouseID": {"value": "WHOLESALE"}
        }
    ],
    "Hold": {"value": false},
    "LocationID": {"value": "MAIN"},
    "OrderType": {"value": "SO"},
    "Payments": [
        {
            "ApplicationDate": {"value": "2023-08-11T00:00:00+03:00"},
            "AppliedToOrder": {"value": 480.00},
            "CashAccount": {"value": "10600"},
            "PaymentAmount": {"value": 980.00},
            "PaymentMethod": {"value": "ACUPAYCC"},
            "Authorize": {"value": true}
        }
    ],
    "RequestedOn": {"value": "2023-05-10T00:00:00"}
}
```

Create a Sales Order with an External Credit Card Payment

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a sales order with payments in a single API call. For details about the creation of sales orders, see *Reserving Payments for Sales Orders*.

System Preparation

Before you test the code below, do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Request

You can use the following example of an HTTP request to create in one call a sales order with an external credit card payment. You specify the external credit card payment parameters in the `CreditCardTransactionInfo` structure.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "WIDGETCC"},
    "Date": {"value": "2023-05-10T00:00:00"},
    "Description": {"value": "External CC payment"},
    "Details": [
        {
            "Branch": {"value": "PRODWHOLE"},
            "DiscountAmount": {"value": 10.00},
            "ExtendedPrice": {"value": 500.00},
            "FreeItem": {"value": false},
            "InventoryID": {"value": "AACOMPUT01"},
            "OrderQty": {"value": 1.00},
            "UnitPrice": {"value": 500.00},
            "UOM": {"value": "EA"},
            "WarehouseID": {"value": "WHOLESALE"}
        },
        {
            "Branch": {"value": "PRODWHOLE"},
            "DiscountAmount": {"value": 10.00},
            "ExtendedPrice": {"value": 500.00},
            "FreeItem": {"value": false},
            "InventoryID": {"value": "AALEGO500"},
            "OrderQty": {"value": 50.00},
            "UnitPrice": {"value": 50.00},
            "UOM": {"value": "EA"},
            "WarehouseID": {"value": "WHOLESALE"}
        }
    ],
    "Hold": {"value": false},
    "LocationID": {"value": "MAIN"},
    "OrderType": {"value": "SO"},
    "Payments": [
        {
            "ApplicationDate": {"value": "2023-08-11T00:00:00+03:00"},
            "AppliedToOrder": {"value": 480.00},
            "CashAccount": {"value": "10600"},
            "CreditCardTransactionInfo": [
                {
                    "NeedValidation": {"value": true},
                    "TranDate": {"value": "2023-08-11T00:00:00+03:00"},
                    "TranNbr": {"value": "40050474170"},
                    "TranType": {"value": "AUT"}
                }
            ],
            "PaymentAmount": {"value": 980.00},
            "PaymentMethod": {"value": "ACUPAYCC"}        }
```

```
    ],
    "RequestedOn": {"value": "2023-05-10T00:00:00"}
}
```

Apply Discounts to a Sales Order

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can apply the available discounts to a document (such as a sales order or AR invoice) in a single API call. For details about the configuration and application of discounts, see *Configuring and Applying Customer Discounts*.

A user scenario involving the need to apply discounts can be the following: Through an external system, a manager of the company needs to import sales orders or other documents to Acumatica ERP and apply the available discounts to them. For details about the preparation of data for the import, creation, and running of import scenarios, see *Preparing Data for Import and Export by Using Scenarios*, *Configuring Import Scenarios*, and *Data Import*.

Although discounts can be applied to other types of documents, in this example, sales orders will be imported.

System Preparation

Before you test the code below, you do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management*, *Inventory*, and *Customer Discounts* features are enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

To continue preparing the system, you need to create a discount code and a discount based on this discount code, and then import a sales order to which the discount is applicable and another sales order to which the discount is not applicable. You perform these tasks as follows:

1. On the *Discount Codes* (AR209000) form, add a row, and create a discount code of the *Line* discount type that is applicable to *Customer and Item*. In the row, leave the four check boxes cleared for the discount code. It is especially important that the check box in the **Manual** column be cleared so that the system will calculate the discount automatically. Click **Save** on the form toolbar.
2. On the *Discounts* (AR209500) form, select the discount code created in the previous instruction; then in the **Sequence** box, type the name and description for a new sequence, and click **Save** on the form toolbar.
3. Specify the following settings in the Summary area:
   - **Discount By**: *Percent*
   - **Break By**: *Amount*
   - **Active**: Selected
   - **Promotional**: Selected
   - **Effective Date**: Today's date

- **Expiration Date**: Any future date

4. On the **Discount Breakpoints** tab, add a row to the table, and specify the following settings in the row:
   - **Break Amount**: `500`
   - **Discount Percent**: `5`
5. On the **Items** tab, add a row to the table, and in the **Inventory ID** column of the row, select *APJAM32*.
6. On the **Customers** tab, add a row to the table, and in the **Customer** column, select *GOODFOOD*.
7. On the form toolbar, click **Save**.
8. Create a CSV file with the following contents.

```
ORDER NBR;ORDER TYPE;CUSTOMER;LOCATION;BRANCH;INVENTORY ID;QUANTITY
SO0001;SO;GOODFOOD;MAIN;HEADOFFICE;APJAM32;10
SO0002;SO;CANDYY;MAIN;HEADOFFICE;APJAM32;10
```

9. On the *Data Providers* (SM206015) form, create a data provider as follows:
   a. In the **Name** box, specify the name to be used for the data provider.
   b. In the **Provider Type** box, select *CSV Provider*.
   c. Save the data provider.
   d. Drag the CSV file that you have created onto the form.
   e. On the **Parameters** tab, set the value of the `Delimiter` parameter to *;*.
   f. On the **Schema** tab, fill the schema of the data provider as follows:
      a. On the left pane toolbar, click **Fill Schema Objects**.
      b. In the **Active** column of the **Source Objects** table, select the check box in every row.
      c. On the right pane toolbar, click **Fill Schema Fields**.
   g. On the form toolbar, click **Save**.
10. On the *Import Scenarios* (SM206025) form, create an import scenario. In the Summary area, specify the following settings for it (other settings in the area should remain unchanged):
    - **Screen Name**: *Sales Orders* (the **Screen ID** is *SO.30.10.00*)
    - **Provider**: The data provider you created
    - **Provider Object**: The CSV file used as the source for the data provider
11. Click **Save** on the form toolbar.
12. For the created import scenario, on the **Mapping** tab, add rows with the settings shown in the following table (leaving the **Active** check box selected in each row).

Target Object	Field / Action Name	Source Field / Value
Order Summary	*Key: OrderType*	*=[Document.OrderType]*
Order Summary	*Key: OrderNbr*	*=[Document.OrderNbr]*
Order Summary	*Action: Cancel*	
Order Summary	*Order Nbr.*	*ORDER NBR*
Order Summary	*Action: Cancel*	
Order Summary	*Order Type*	*ORDER TYPE*

Target Object	Field / Action Name	Source Field / Value
Order Summary	*Customer*	*CUSTOMER*
Order Summary	*Location*	*LOCATION*
Details	*<Line Number>*	*=-1*
Details	*Branch*	*BRANCH*
Details	*Inventory ID*	*INVENTORY ID*
Details	*Quantity*	*QUANTITY*
Order Summary	*Action: Save*	

13. Click **Save** on the form toolbar.
14. On the *Import by Scenario* (SM206036) form, select the import scenario you have created, click **Prepare**, make sure that the table on the **Prepared Data** tab contains the correct data of two sales orders, and click **Import**.

Request

You can use the following example of an HTTP request to apply available discounts to both imported sales orders. To affect both sales orders, you need to call the HTTP request twice, each time specifying the proper order number as the value of the `entity/OrderNbr` field.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder/
AutoRecalculateDiscounts
Accept: application/json
Content-Type: application/json

{
    "entity" :
    {
        "OrderType" : {"value" : "SO"},
        "OrderNbr" : {"value" : "000065"}
    },
    "parameters": { }
}
```

Check the imported sales orders. Notice that the discount you created affected the imported sales order whose `OrderNbr` is specified in the request, but the discount did not affect the other imported sales order.

Create a Return for Credit Without Validation of the Card Refund Against the Original Transaction

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create refunds of credit card payments without validation against the original transactions.

You can verify whether the original transaction number specified for a refund is related to the payment made. You can do this in one of the following ways:

- On the *Processing Centers* (CA205000) form, select the **Allow Unlinked Refunds** check box for the payment method's default processing center.
- On the *Order Types* (SO201000) form, clear the **Validate Card Refunds Against Original Transactions** check box for the sales order type that is used to create a refund.
- In a contract-based REST API request, set the `ValidateCCRefundOrigTransaction` value of the `SalesOrderPayment` entity to *false*.

System Preparation

Before you test the code below, you do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *SalesDemo* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *PRODWHOLE* branch.

Request

You can use the following example of an HTTP request to make a refund. In this request, you will do the following:

- Set `Date` to today's date
- Set `Hold` to *false*
- In `Details`, specify the inventory items for which the refund must be made
- Set `Payments.Refund` to *true*
- Specify the `Payments.OrigTransactionNbr`, `Payments.CardAccountNbr`, and `Payments.CashAccount` values from the information of a payment that is not related to the inventory items from `Details`
- Set `Payments.DocType` to *Refund*

If you run the following example of an HTTP request with `Payments.ValidateCCRefundOrigTransaction` set to *true*, the system will generate an error that indicates that the return for credit to be created has no items that were paid with the transaction specified as the original.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details,Payments HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CurrencyID": {
        "value": "USD"
    },
    "CustomerID": {
        "value": "WIDGETCC"
    },
    "Date": {
        "value": "2023-08-19T00:00:00+03:00"
    },
    "Hold": {
                "value": false
            },
    "Details": [
        {
            "Branch": {
                "value": "PRODWHOLE"
            },
            "InventoryID": {
                "value": "AACOMPUT01"
            },
            "OrderQty": {
                "value": 5.000000
            },
            "UnitPrice": {
                "value": 500.000000
            },
            "UOM": {
                "value": "EA"
            },
            "WarehouseID": {
                "value": "WHOLESALE"
            }
         }
    ],
    "OrderType": {
        "value": "RC"
    },
    "PaymentMethod": {
        "value": "ACUPAYCC"
    },
    "Payments": [
        {
            "ApplicationDate": {
                "value": "2023-08-19T00:00:00+03:00"
            },
```

```
            "AppliedToOrder": {
                "value": 40.0000
            },
            "CardAccountNbr": {
                "value": "VISA:****-****-****-1111"
            },
            "CashAccount": {
                "value": "10250"
            },
            "Currency": {
                "value": "USD"
            },
            "DocType": {
                "value": "Refund"
            },
            "Hold": {
                "value": false
            },
            "OrigTransactionNbr": {
                "value": "60165234282"
            },
            "PaymentAmount": {
                "value": 40.0000
            },
            "PaymentMethod": {
                "value": "ACUPAYCC"
            },
            "PaymentRef": {
                "value": "601704533664"
            },
            "ProcessingCenterID": {
                "value": "ACUPAYMENT"
            },
            "Refund": {
                "value": true
            },
            "ValidateCCRefundOrigTransaction": {
                "value": false
            }
        }
    ]
}
```


If the `Payments/ValidateCCRefundOrigTransaction` value is *false*, the request is executed successfully.

Retrieve a Sales Order by Using the Values of Specific Fields

If you are using the REST API to integrate Acumatica ERP with an external system, this external system can retrieve a sales order by using the values of specific fields of the sales order. For example, before submitting a sales order for processing, a customer of the online store can select the needed order by using this customer order number.

The customer order number of a sales order is entered in the **Customer Order Nbr.** box of the Summary area of the *Sales Orders* (SO301000) form.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

You can use the following sample HTTP request to retrieve the sales order whose customer order number is *SO248-563-06* along with its detail lines through the contract-based REST API.

```
GET /entity/Default/24.200.001/SalesOrder?
    $expand=Details&
    $select=OrderNbr,OrderType,Details/InventoryID,Details/WarehouseID&
    $filter=OrderType eq 'SO' and CustomerOrder eq 'SO248-563-06' HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
```

Update the Detail Lines of a Sales Order

If you are using the REST API to integrate Acumatica ERP with an external system, this external system can update the detail lines of a sales order. For example, a customer of the online store can edit the order and submit the updated order. The updated sales order can be viewed on the *Sales Orders* (SO301000) form.

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

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

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

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.
4. Execute the request that is described in *Retrieve a Sales Order by Using the Values of Specific Fields* to obtain the IDs of the detail lines hat should be updated.

Request

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

You can use the following request example to delete the line for the *CONGRILL* item (which has the *08731a64-7381-e511-80c0-00155d012302* identifier in the session of this request), and update the quantity of the *AALEGO500* item (which has the *f4a7e172-7381-e511-80c0-00155d012302* session identifier).

```
PUT /entity/Default/24.200.001/SalesOrder?
    $expand=Details&
    $select=OrderType,OrderNbr,Details/OrderQty,Details/InventoryID,
    OrderedQty,OrderTotal HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

{
 "OrderType": {"value": "SO"},
 "OrderNbr": {"value": "000003"},
 "Hold": {"value": false},
 "Details":[
  {
   "id": "08731a64-7381-e511-80c0-00155d012302",
   "delete": true
  },
  {
   "id": "f4a7e172-7381-e511-80c0-00155d012302",
   "OrderQty": {"value": 5}
  }
 ]
}
```

Create a Shipment from a Sales Order

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a shipment from a sales order in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a shipment from the *000029* sales order (whose ID is *42bb9a17-a402-e911-b818-00155d408001*) through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /SalesOrderCreateShipment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "id": "42bb9a17-a402-e911-b818-00155d408001"
  },
  "parameters": {
      "ShipmentDate": { "value": "2025-08-20T00:00:00+03:00" },
      "WarehouseID": { "value": "RETAIL" }
  }
}
```

Create a Return for Credit

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a return for credit (that is, a return order of the *RC* type) for any number of invoices in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following request example to create a return order through the contract-based REST API. This return order will be created to refuse the *INSTALL* service included in the *000062* invoice and return 50 units of the *ORANGES* product included in the *000030* invoice.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "GOODFOOD" },
    "Description": { "value": "Return Order for Invoices" },
    "Details": [
        {
            "InvoiceType": { "value": "Invoice" },
            "InvoiceNbr": { "value": "000062" },
            "InventoryID": { "value": "INSTALL" }
        },
        {
            "InvoiceType": { "value": "Invoice" },
            "InvoiceNbr": { "value": "000030" },
            "InventoryID": { "value": "ORANGES" },
            "OrderQty": { "value": "50" }
        }
    ],
    "LocationID": { "value": "MAIN" },
    "OrderType": { "value": "RC" }
}
```

Create a Sales Order with Tax Parameters Overridden

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a sales order. The external system can override the tax details that are calculated by the system in the order.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

You can use the following example of an HTTP request to create a sales order and set the amount of the *NYSTATETAX* tax to *0.5*.

```
PUT ?$expand=Details,TaxDetails HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "GOODFOOD" },
    "IsTaxValid": { "value": true },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APJAM08" },
            "OrderQty": { "value": 2 },
            "UOM": { "value": "PIECE" },
            "WarehouseID": { "value": "WHOLESALE" }
        }
    ],
    "TaxDetails": [
        {
```

```
            "TaxID": { "value": "NYSTATETAX" },
            "TaxAmount": { "value": 0.5 }
        }
    ]
}
```

Usage Notes

If the `IsTaxValid` field is specified in the request body, its value is set to *true*, and an external tax zone is specified, then the tax details from the request body are used without modification, and no other taxes are applied.

If the `IsTaxValid` field is specified in the request body, its value is set to *true*, and an internal tax zone is specified, then the tax details from the request body are compared to those calculated by the system. If the taxes specified in the request body are present among those calculated by the system, then the tax details from the request body are used without modification and no other taxes are applied.

If the `IsTaxValid` field is not specified in the request body or its value is not set to *true*, tax calculation is performed by the system.

Create a Sales Order with Allocations

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a sales order with allocations in a single API call.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

Request

You can use the following example of an HTTP request to create a sales order with allocations in one call.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details,Details/Allocations HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
```

```
Content-Type: application/json

{
    "CustomerID": {"value": "COFFEESHOP"},
    "Description": {"value": "Sales Order with Allocations"},
    "Details": [
        {
            "InventoryID": {"value": "APJAM08"},
            "Allocations": [
                {
                    "Allocated": { "value": true },
                    "AllocWarehouseID": { "value": "WHOLESALE" },
                    "InventoryID": { "value": "APJAM08" },
                    "LotSerialNbr": { "value": "116046" },
                    "Qty": { "value": 1 }
                }
            ]
        }
    ],
    "Hold": {"value": true},
    "LocationID": {"value": "MAIN"},
    "OrderType": {"value": "SO"}
}
```

Usage Notes

If you specify the values of the `Details/Allocations/Qty` fields of a detail line, do not specify the value of the `Details/OrderQty` field of the same line, so that no empty allocations appear in the created sales order.

Create an RMA Order for a Return

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a return merchandise authorization (RMA) order—that is, a return order of the *RM* type.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* and *Advanced SO Invoices* features are enabled.
5. Prepare a sales order, shipment, and an invoice as follows:

> If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

a. On the *Sales Orders* (SO301000) form, create a sales order to sell one *SWB-32OZ-GBT* item from the *Retail* warehouse to the *FRUITICO* customer.
b. On the form toolbar, click **Create Shipment** to create a shipment for the sales order.
c. In the **Specify Shipment Parameters** dialog box, which opens, specify today's date as the shipment date and *Retail* as the warehouse, and click **OK**.
d. On the *Shipments* (SO302000) form, which opens, click **Confirm Shipment** and then click **Prepare Invoice**.
e. On the *Invoices* (SO303000) form, which opens, click **Release**. In this example, the invoice number is supposed to be *000126*.

Request

You can use the following example of an HTTP request to create a return order of the *RM* type for the *FRUITICO* customer. According to the order, an *SWB-32OZ-GBT* item will be returned, and a *PEARJAM96* item will be sold to the customer.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "FRUITICO" },
    "OrderType": { "value": "RM" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InvoiceNbr": { "value": "000126" },
            "Operation": { "value": "Receipt" },
            "InventoryID": { "value": "SWB-32OZ-GBT" },
            "OrderQty": { "value": -1 },
            "UOM": { "value": "EA" },
            "WarehouseID": { "value": "WHOLESALE" },
            "AutoCreateIssue": { "value": false }
        },
        {
            "Branch": { "value": "HEADOFFICE" },
            "Operation": { "value": "Issue" },
            "InventoryID": { "value": "PEARJAM96" },
            "OrderQty": { "value": 1 },
            "UOM": { "value": "PIECE" },
            "WarehouseID": { "value": "RETAIL" },
            "AutoCreateIssue": { "value": false }
        }
```

```
    ]
}
```

Create a Shipment with the Receipt Operation for an RMA Order for a Return

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a return merchandise authorization (RMA) order—that is, a return order of the *RM* type—in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
5. Execute the *Create an RMA Order for a Return* request.

Request

You can use the following request example to create a shipment with the *Receipt* operation from the *000127* return order of the *RM* type through the contract-based REST API. In the `WarehouseID` parameter, you specify the destination warehouse for the items that will be returned.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
POST /SalesOrderCreateReceipt HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "OrderNbr": {"value": "000127"},
    "OrderType":  {"value": "RM"}
  },
  "parameters": {
    "ShipmentDate":  {"value": "2023-02-11T00:00:00+03:00"},
    "WarehouseID":  {"value": "WHOLESALE"}
  }
```

```
}
```

ServiceOrder

This chapter presents code examples showing requests that use the `ServiceOrder` entity.

Retrieve a Service Order

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve service orders from Acumatica ERP.

System Preparation

Before you test the code below, do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Service Management* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve the service order with the *000019* ID through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$filter=ServiceOrderNbr%20eq%20'000019' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ServiceOrder
Accept: application/json
Content-Type: application/json
```

Shipment

This chapter presents code examples showing requests that use the `Shipment` entity.

Create a Shipment for Sales Orders

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a shipment for multiple sales orders in a single API call.

System Preparation

Before you test the request below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.
4. On the *Sales Orders* (SO301000) form, find the sales orders with the *000004* and *000006* order numbers. These are the preconfigured sales orders of the customer with the *C000000003* customer ID. Make sure that these sales orders have the *Open* status.

Request

You can use the following example of an HTTP request to create a shipment for two sales orders.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

If you do not use the Postman collection provided with this course, to execute this request, you need to create a new request in a Postman collection and specify the settings as follows:

1. Select the `PUT` HTTP method.
2. Specify the *<Acumatica ERP instance URL>/entity/Default/24.200.001/Shipment* URL, such as *https://localhost/MyStoreInstance/entity/Default/24.200.001/Shipment*.
3. On the **Params** tabs, specify the values of the *$expand* and *$select* parameters.
4. Make sure that the authorization is configured correctly.
5. On the **Headers** tab, add the values of the `Accept` and `Content-Type` headers.
6. On the **Body** tab, add the body of the request.

```
PUT /entity/Default/24.200.001/Shipment?
    $select=Type,ShipmentNbr,Status,Details/InventoryID&
    $expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json
```

```
{
 "Type":{"value":"Shipment"},
 "CustomerID":{"value":"C000000003"},
 "WarehouseID":{"value":"MAIN"},
      "ShipmentDate":{"value":"2024-11-01"},
 "Details":[
  {
   "OrderType":{"value":"SO"},
   "OrderNbr":{"value":"000004"}
  },
  {
   "OrderType":{"value":"SO"},
   "OrderNbr":{"value":"000006"}
  }
 ]
}
```

Usage Notes

Note the following about creation of a shipment:

- Although you are creating a shipment with multiple detail lines, you use one request for the creation of the shipment. (That is, you do not need to submit each detail of the shipment in a separate request.) For optimal performance of the application, it is important to use the minimum number of requests.
- If you need to include in a shipment particular items (rather than all items) from a sales order, you need to retrieve the sales orders with the included items from Acumatica ERP by using the key fields, and then include the needed items in the shipment (by specifying the inventory ID, the warehouse ID, and the type and number of the sales order in which the item is included in the details of the shipment).
- If you intend to use the `Picked` field of the `Shipment` entity, note that its value may be incorrect in scenarios other than batch picking or wave picking.

Create a Shipment for Two Sales Orders with Allocations and Package Specifications

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a shipment with allocations and package specifications in a single API call.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

Also, in this example, a shipment will be created for two sales orders. Before you create the shipment, you need to create two sales orders. You can do this on the *Sales Orders* (SO301000) form or through the contract-based REST API, as described in *Create a Sales Order with the Unit of Measure Specified*. The sales orders must have the following settings.

If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

Element	Sales Order 1	Sales Order 2
Order Type	*SO*	*SO*
Customer	*GOODFOOD*	*GOODFOOD*
Branch column (in the only row on the **Details** tab)	*HEADOFFICE*	*HEADOFFICE*
Inventory ID column (in the only row on the **Details** tab)	*APJAM08*	*APJAM32*
Quantity column (in the only row on the **Details** tab)	2	1
UOM column (in the only row on the **Details** tab)	*PIECE*	*BOX*
Warehouse column (in the only row on the **Details** tab)	*WHOLESALE*	*WHOLESALE*

Request

You can use the following example of an HTTP request to create a shipment for the two sales orders that you created. For the first sales order (in this example, its ID is *000071*; if the sales order you added has a different number, it should be used in the code), you take one jar of jam from the *L2R3S1* location and another jar of jam from the *L3R2S1* location. For the second sales order (in this example, its ID is *000072*; if the sales order you added has a different number, it should be used in the code), you take jars of jam from the *L1R3S2* location. Also, you indicate that all jars of jam of both sales orders should be packed into one large box.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details,Details/Allocations,Packages,Packages/PackageContents HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "GOODFOOD" },
```

```
    "Details": [
        {
            "OrderNbr": { "value": "000071" },
            "OrderType": { "value": "SO" },
            "OrderLineNbr": { "value": 1 },
            "Allocations": [
                {
                    "InventoryID": { "value": "APJAM08" },
                    "LocationID": { "value": "L2R3S1" },
                    "Qty": { "value": 1 },
                    "UOM": { "value": "PIECE" }
                },
                {
                    "InventoryID": { "value": "APJAM08" },
                    "LocationID": { "value": "L3R2S1" },
                    "Qty": { "value": 1 },
                    "UOM": { "value": "PIECE" }
                }
            ]
        },
        {
            "OrderNbr": { "value": "000072" },
            "OrderType": { "value": "SO" },
            "OrderLineNbr": { "value": 1 },
            "Allocations": [
                {
                    "InventoryID": { "value": "APJAM32" },
                    "LocationID": { "value": "L1R3S2" },
                    "Qty": { "value": 6 },
                    "UOM": { "value": "PIECE" }
                }
            ]
        }
    ],
    "LocationID": { "value": "MAIN" },
    "Operation": { "value": "Issue" },
    "Packages": [
        {
            "BoxID": { "value": "LARGE" },
            "UOM": { "value": "KG" },
            "Weight": { "value": 15 },
            "PackageContents": [
                {
                    "InventoryID": { "value": "APJAM08" },
                    "LotSerialNbr": { "value": "" },
                    "Quantity": { "value": 1 },
                    "UOM": { "value": "PIECE" },
                    "OrigOrderType": { "value": "SO" },
                    "OrigOrderNbr": { "value": "000071" }
                },
                {
                    "InventoryID": { "value": "APJAM08" },
                    "LotSerialNbr": { "value": "" },
                    "Quantity": { "value": 1 },
                    "UOM": { "value": "PIECE" },
                    "OrigOrderType": { "value": "SO" },
                    "OrigOrderNbr": { "value": "000071" }
```

```
                },
                {
                    "InventoryID": { "value": "APJAM32" },
                    "LotSerialNbr": { "value": "" },
                    "Quantity": { "value": 6 },
                    "UOM": { "value": "PIECE" },
                    "OrigOrderType": { "value": "SO" },
                    "OrigOrderNbr": { "value": "000072" }
                }
            ]
        }
    ],
    "WarehouseID": { "value": "WHOLESALE" }
}
```

Usage Notes

If you intend to use the `Picked` field of the `Shipment` entity, note that its value may be incorrect in scenarios other than batch picking or wave picking.

Read the Tracking Number from a Shipment

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve a shipment from Acumatica ERP and read its tracking number.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve the *000001* shipment, along with the tracking number of its package, through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$expand=Packages&$select=Packages/TrackingNbr
    &$filter=ShipmentNbr%20eq%20'000001' HTTP/1.1
```

```
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json
```

Write the Tracking Number to a Shipment

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can set the tracking number of a package of a shipment in Acumatica ERP.

System Preparation

Before you test the code below, do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. On the *Shipments* (SO302000) form, for the shipment with the *000058* number, on the **Packages** tab, add a package with the *SMALL* box ID.

Request

You can use the following request example to specify the tracking number of the package with the *ec062915-9061-ec11-9e19-9828a61840c3* ID, which is shipped in the shipment with the *5d79d031-8763-ea11-b82d-00155d408001* ID through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Packages&$select=Packages/TrackingNbr HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
    "id": "5d79d031-8763-ea11-b82d-00155d408001",
    "Packages": [
        {
            "id": "ec062915-9061-ec11-9e19-9828a61840c3",
            "TrackingNbr": { "value": "398305336619" }
        }
    ]
}
```

Usage Notes

You can learn the value of the `id` field of the shipment in the `SOShipment` database table, and the value of the `id` field of the package in the `SOPackageDetail` database table.

Update the Freight Cost or Price

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can update the freight cost or freight price of a shipment in Acumatica ERP.

System Preparation

Before you test the code below, do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
5. Configure the shipment settings as follows:

   > If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

   a. On the **Details** tab of the *Ship via Codes* (CS207500) form, for the *LOCAL* ship via code, select the *Manual* calculation method, and save your changes.

   b. On the **Shipping** tab of the *Shipments* (SO302000) form, for the *000058* shipment, specify *LOCAL* in the **Ship Via** box and save your changes.

Request

You can use the following request example to specify the freight cost and freight price of the *000058* shipment through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json
```

```
{
    "FreightPrice": { "value": 2.0000 },
    "FreightCost": { "value": 1.0000 },
    "OverrideFreightPrice": { "value": true },
    "ShipmentNbr": { "value": "000058" }
}
```

Usage Notes

To update the freight cost or freight price of a shipment in Acumatica ERP, you use the following fields of the `Shipment` entity:

- `FreightPrice`: The freight price
- `FreightCost`: The freight cost
- `OverrideFreightPrice`: The field that makes the `FreightPrice` field either read-only (when its value is *false*) or available for editing (when its value is *true*)

Create Separate Shipments for Each Sales Order

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create separate shipments for multiple sales orders in a single API call.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Also, in this example, two shipments will be created for two sales orders. Before you create shipments, you need to create two sales orders. You can do this on the *Sales Orders* (SO301000) form or through the contract-based REST API, as described in *Create a Sales Order with the Unit of Measure Specified*. The sales orders must have the following settings.

If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

Element	Sales Order 1	Sales Order 2
Order Type	*SO*	*SO*

Element	Sales Order 1	Sales Order 2
Customer	*GOODFOOD*	*GOODFOOD*
Branch column (in the only row on the **Details** tab)	*HEADOFFICE*	*HEADOFFICE*
Inventory ID column (in the only row on the **Details** tab)	*APJAM08*	*APJAM32*
Quantity column (in the only row on the **Details** tab)	20	3
UOM column (in the only row on the **Details** tab)	*PIECE*	*BOX*
Warehouse column (in the only row on the **Details** tab)	*WHOLESALE*	*WHOLESALE*

Request

You can use the following example of an HTTP request to create two separate shipments for the two sales orders that you created. In this example, the number of the first sales order is *000063*, and the number of the second sales order is *000064*; if the sales order you added has a different number, it should be used in the code.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "GOODFOOD" },
    "Details": [
        {
            "OrderNbr": { "value": "000063" },
            "OrderType": { "value": "SO" }
        },
        {
            "OrderNbr": { "value": "000064" },
            "OrderType": { "value": "SO" }
        }
    ],
    "LocationID": { "value": "MAIN" },
    "Operation": { "value": "Issue" },
    "WarehouseID": { "value": "WHOLESALE" },
    "CreateNewShipmentForEveryOrder": { "value": true}
}
```

Usage Notes

You set the `CreateNewShipmentForEveryOrder` field of the `Shipment` entity to *true* to specify that a separate shipment must be created for every sales order listed in `Details`. You set the `CreateNewShipmentForEveryOrder` field to *false* to specify that a single shipment must be created for all sales order listed in `Details`. The default value is *false*.

StockItem

This chapter presents code examples showing requests that use the `StockItem` entity.

Retrieve the List of Modified Stock Items

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the list of stock items that have been modified within a particular period.

You will use the `GET` HTTP method and the `StockItem` entity of the *Default/24.200.001* endpoint to list the stock items. The `StockItem` entity is mapped to the *Stock Items* (IN202500) form.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Stock Items* (IN202500) form, modify these inventory items as follows:
   - *AACOMPUT01*: Change the status of this inventory item to *Inactive* and save the record.
   - *AALEGO500*: Change the description of this inventory item and save the record.

   Now you have at least two inventory items that have been modified within the past month, and one of them has the *Active* status.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

You can use the following request example to retrieve the active stock item that was changed today through the contract-based REST API. (You should specify today's date instead of `2024-11-14`.) The only stock item that the response will contain will be *AALEGO500*, which you have just changed.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /entity/Default/24.200.001/StockItem?
    $expand=WarehouseDetails&
    $select=InventoryID,Description,WarehouseDetails/WarehouseID,
        WarehouseDetails/QtyOnHand,ItemClass,BaseUOM&
    $filter=ItemStatus eq 'Active' and
        LastModified gt datetimeoffset'2024-11-14T00:00:00.000' HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json
```

Usage Notes

You use the `$filter` parameter to specify the search conditions for the fields.

Because the database can contain thousands of stock item records, to achieve the best performance of the application, you need to specify the fields of the stock item records that should be returned. You use the `$select` parameter to specify the fields whose values should be retrieved from Acumatica ERP for each stock item record.

The fields that are specified in the `$filter` parameter are returned by default.

You use the `$expand` parameter to specify the nested entity, such as `WarehouseDetails`, to be returned.

Retrieve Stock Items with Attributes

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve stock items along with their attributes from Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve the *APL-16OZ-GBT* stock item, along with its attributes through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /APL-16OZ-GBT?$expand=Attributes HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
```

```
Accept: application/json
Content-Type: application/json
```

Retrieve Unit Conversion Rules from a Stock Item

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve unit conversion rules from a stock item in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve unit conversion rules from the *PLUMJAM96* stock item through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /PLUMJAM96?$expand=UOMConversions HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json
```

Retrieve Stock Items with Prices and Quantities by Warehouse

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve stock items with details by using the identifier of the warehouse specified as the default for the item from Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.

2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve stock items whose default warehouse is *WHOLESALE*, along with their default prices and quantities, through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$filter=DefaultWarehouseID%20eq%20'WHOLESALE'&$expand=WarehouseDetails
    &$select=DefaultPrice,WarehouseDetails/QtyOnHand,
    WarehouseDetails/WarehouseID HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json
```

Retrieve the List of Attachments of a Stock Item

If you are using the REST API to integrate Acumatica ERP with an external system, this external system can retrieve the attachments of a stock item. For example, in an online store, you may need to display an image of each item that is sold in the store. Images for the items can be stored in Acumatica ERP as attachments to the *Stock Items* (IN202500) form. To display an image of a stock item in the online store, you should retrieve the list of images that are attached to the stock item and then export the needed image as described in *Retrieve the File Attached to a Stock Item*.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Requests

> In the request examples below, <Acumatica ERP instance URL> is the URL of the Acumatica ERP instance (such as https://my.acumatica.com/MyInstance). You can omit the instance name in the URL (such as https://my.acumatica.com) if the instance is installed in the root of the website.

You can use the following request example to retrieve the URL of the attachment of the *AAMACHINE1* stock item through the REST API.

```
GET /AAMACHINE1?$select=InventoryID,files&$expand=files HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/StockItem
Accept: application/json
Content-Type: application/json
```

An example of a response is shown below.

```
{
    "id": "2acd2eed-1614-e511-9b82-c86000dddf0b",
    "rowNumber": 1,
    "note": {
        "value": ""
    },
    "InventoryID": {
        "value": "AAMACHINE1"
    },
    "custom": {},
    "_links": {
        "self": ...,
        "files:put": ...
    },
    "files": [
        {
            "id": "9be45eb7-f97d-400b-96a5-1c4cf82faa96",
            "filename": "Stock Items (AAMACHINE1)\\T2MCRO.jpg",
            "href": "/<Acumatica ERP instance name>/entity/Default/
24.200.001/files/9be45eb7-f97d-400b-96a5-1c4cf82faa96"
        }
    ]
}
```

Retrieve the File Attached to a Stock Item

If you are using the REST API to integrate Acumatica ERP with an external system, this external system can retrieve the attachments of a stock item. For example, in an online store, you may need to display an image of each item that is sold in the store. Images for the items can be stored in Acumatica ERP as attachments to the *Stock Items* (IN202500) form. To display an image of a stock item in the online store, you should retrieve the list of images that are attached to the stock item as described in *Retrieve the List of Attachments of a Stock Item* and then export the needed image as described in this topic.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.
4. To find out the URL for the retrieval of the attachment, execute the request that is described in *Retrieve the List of Attachments of a Stock Item*.

Requests

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

To retrieve the file attached to the *AAMACHINE1* stock item through the REST API, you can use the URL specified in the `files/href` field of the response of the request that is described in *Retrieve the List of Attachments of a Stock Item*. An example of a request is shown below. The response will contain the contents of the `T2MCRO.jpg` file.

```
GET /9be45eb7-f97d-400b-96a5-1c4cf82faa96 HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/files
Accept: application/json
Content-Type: application/json
```

Create a Stock Item with Attributes

In this example, by using the contract-based REST API, you will create in Acumatica ERP a stock item that has attributes specified. An *attribute* is a property of an object in the system that specifies additional information that is not defined by the standard properties of the object (that is, those supported by the standard UI elements).

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

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.

2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

You can use the following request example to create a stock item record with attributes through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT /entity/Default/24.200.001/StockItem?
    $select=InventoryID,Attributes/AttributeDescription,Attributes/Value&
    $expand=Attributes HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

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

Add a Note to a Stock Item

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can add a note to a stock item.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.

2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

You can use the following request example to add the *My note* note to the *AALEGO500* stock item through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT /entity/Default/24.200.001/StockItem?
    $select=InventoryID HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
Content-Type: application/json

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

Usage Notes

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

Obtain the URL for Attaching a File

This example shows how you can obtain the URL you will use for attaching a file to a particular stock item record by using the contract-based REST API.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.

Request

You can use the following sample request to obtain the URL you will use for attaching a file to the *AALEGO500* stock item record through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET /entity/Default/24.200.001/StockItem/AALEGO500?
    $select=InventoryID HTTP/1.1
Host: [<Acumatica ERP instance URL>]
Accept: application/json
```

Attach a File to a Stock Item

In this example, you attach a file to a stock item record by using the REST API.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *T100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *MYSTORE* branch.
4. Perform the following prerequisite activity: *Obtain the URL for Attaching a File*.

Request

You can use the following sample request to attach the *`T2MCRO.jpg`* file, which is provided in the Acumatica GitHub repository, to the *AALEGO500* stock item through the contract-based REST API.

> In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT /entity/Default/24.200.001/files/PX.Objects.IN.InventoryItemMaint/
    Item/cae53ce0-1614-e511-9b82-c86000dddf0b/T2MCRO.jpg HTTP/1.1
Host: [<Acumatica ERP instance URL>]
```

```
Accept: application/json
Content-Type: application/octet-stream

"<The T2MCRO.jpg file in binary format>"
```

TaxCategory

This chapter presents code examples showing requests that use the `TaxCategory` entity.

Update a Tax Category

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can update tax categories in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to update the *TAXABLE* tax category through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/TaxCategory
Accept: application/json
Content-Type: application/json

{
    "Active": { "value": true },
    "Description": { "value": "Taxable Goods and Services v2" },
    "ExcludeListedTaxes": { "value": true },
    "TaxCategoryID": { "value": "TAXABLE" }
}
```

TimeEntry

This chapter presents code examples showing requests that use the `TimeEntry` entity.

Read Employee Time Activities

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can read employee time activities in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve all available time activities through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/TimeEntry
Accept: application/json
Content-Type: application/json
```

Write Employee Time Activities

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can write the time spent on the tasks of projects to Acumatica ERP. For more information about tracking time on projects, see *Employee Time Billing: General Information*.

System Preparation

Before you test the code below, you configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects Accounting* and *Time Management* features are enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. Sign in as *gibbs* with the *123* password to the instance in the client application by using the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following request example to track time through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/TimeEntry
Accept: application/json
Content-Type: application/json

{
  "Summary" : {"value" : "Time entry summary"},
  "Date" : {"value" : "2022-08-17T05:50:43.233" },
  "Employee" : {"value" : "EP00000026" },
  "ProjectID" : {"value" : "TOMYUM1" },
  "ProjectTaskID" : {"value" : "PHASE1" },
  "CostCode" : {"value" : "00-000" },
  "EarningType" : {"value" : "RG" },
  "TimeSpent" : {"value" : "01:30" },
  "BillableTime" : {"value" : "00:30" },
}
```

Usage Notes

The `TimeSpent`, `BillableTime`, and `BillableOvertime` fields of the `TimeEntry` entity have the `StringValue` type. These fields accept values in the following format: *hh:mm*, where *hh* is the number of hours, and *mm* is the number of minutes.

The `TimeEntryID` field has the `GuidValue` type; however, its value is a sequentially generated string that looks like a GUID. Therefore, the global uniqueness of the values is not guaranteed.

Search for Time Entries by Date

Through the contract-based REST API, the time spent on project tasks can be retrieved from Acumatica ERP to an external system. For more information about tracking time on projects, see *Employee Time Billing: General Information*.

System Preparation

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to obtain time entries through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$filter=cf.DateTime(f='Items.Date')%20ge%20datetimeoffset'2022-08-17'
          %20and%20cf.DateTime(f='Items.Date')%20le%20
          datetimeoffset'2022-08-18T23%3A59%3A59.999%2B04%3A00'&
          $select=Date,ProjectID,TimeSpent,BillableTime HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/TimeEntry
Accept: application/json
Content-Type: application/json
```

Usage Notes

To filter time entries by date, you cannot use the `TimeEntry.Date` field of the system endpoint. Instead of this field, you should use the custom `Items.Date` field, as shown in this example.

Vendor

This chapter presents code examples showing requests that use the `Vendor` entity.

Retrieve the List of Vendors

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can retrieve the list of the vendors that are available in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to retrieve vendors that meet a condition (the `CashAccount` field must be equal to *10200TG*) through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
GET ?$filter=CashAccount%20eq%20'10200TG' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Vendor
Accept: application/json
Content-Type: application/json
```

Create a Vendor

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a vendor in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a vendor with the *TESTVENDOR* ID through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Vendor
Accept: application/json
Content-Type: application/json

{
    "APAccount": { "value": "20000" },
    "APSubaccount": { "value": "000000" },
    "CashAccount": { "value": "10200WH" },
    "CurrencyID": { "value": "USD" },
    "CurrencyRateType": { "value": "SPOT" },
    "EnableCurrencyOverride": { "value": true },
    "EnableRateOverride": { "value": false },
    "F1099Vendor": { "value": false },
    "ForeignEntity": { "value": false },
    "LandedCostVendor": { "value": false },
    "LastModifiedDateTime": { "value": "2020-08-11T10:47:41.17-04:00" },
    "LocationName": { "value": "Primary Location" },
    "MaxReceipt": { "value": 100.000000 },
    "MinReceipt": { "value": 0.000000 },
    "PaymentBy": { "value": "Due Date" },
    "PaymentLeadTimedays": { "value": 0 },
    "PaymentMethod": { "value": "CHECK" },
    "PaySeparately": { "value": false },
    "PrintOrders": { "value": false },
    "ReceiptAction": { "value": "Accept but Warn" },
    "RemittanceAddressOverride": { "value": false },
    "RemittanceContactOverride": { "value": false },
    "SendOrdersbyEmail": { "value": false },
    "ShippingAddressOverride": { "value": false },
    "ShippingContactOverride": { "value": false },
    "Status": { "value": "Active" },
    "TaxCalculationMode": { "value": "Tax Settings" },
    "Terms": { "value": "30D" },
    "ThresholdReceipt": { "value": 100.000000 },
    "VendorClass": { "value": "SUBCON" },
    "VendorID": { "value": "TESTVENDOR" },
    "VendorIsLaborUnion": { "value": false },
    "VendorIsTaxAgency": { "value": false  },
    "VendorName": { "value": "TESTVENDOR" }
```

```
}
```

WorkCalendar

This chapter presents code examples showing requests that use the `WorkCalendar` entity.

Create a Work Calendar

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a work calendar in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

Request

You can use the following request example to create a work calendar through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/WorkCalendar
Accept: application/json
Content-Type: application/json

{
    "WorkCalendarID": {
        "value": "TST"
    },
    "Description": {
        "value": "Test Calendar"
    },
    "TimeZone": {
        "value": "GMTM0800A"
    }
```

```
}
```

WorkLocation

This chapter presents code examples showing requests that use the `WorkLocation` entity.

Create a Work Location

If you are using the contract-based REST API to integrate Acumatica ERP with an external system, this external system can create a work location in Acumatica ERP.

System Preparation

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Payroll* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

Request

You can use the following request example to create a work location through the contract-based REST API.

In the request example below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as *https://my.acumatica.com/MyInstance*). You can omit the instance name in the URL (that is, you can use *https://my.acumatica.com*) if the instance is installed in the root of the website.

```
PUT ?$expand=AddressInfo HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/WorkLocation
Accept: application/json
Content-Type: application/json

{
    "Active": { "value": true },
    "WorkLocationID": { "value": "DELLEVUE" },
    "WorkLocationName": { "value": "New address location" },
    "AddressInfo": {
        "AddressLine1": { "value": "1st Browny St." },
        "AddressLine2": { "value": "Suite 200" },
        "City": { "value": "Bellevue" },
        "Country": { "value": "US" },
        "PostalCode": { "value": "98004" },
```

```
        "State": { "value": "WA" }
    }
}
```

Scenarios

This chapter describes various Acumatica ERP contract-based API usage scenarios related to particular functional areas.

Inventory and Order Management

You can use the contract-based REST API for the integration of inventory and order management in Acumatica ERP with external systems. For details on inventory and order management, see *Inventory Management* and *Order Management*.

Creation of a Purchase Receipt from a Purchase Order

If you are using the contract-based REST API to integrate Acumatica ERP with external systems, these external systems can create a purchase order and a purchase receipt (from the purchase order). For details about the management of purchase documents, see *Managing Purchase Documents*.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* feature is enabled.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create a Purchase Order

By using the following code, you create a purchase order and release it from hold at once.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseOrder
```

```
Accept: application/json
Content-Type: application/json

{
    "VendorID": { "value": "GOODFRUITS" },
    "Location": { "value": "MAIN" },
    "Details": [
        {
            "BranchID": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APPLES" },
            "OrderQty": { "value": 1 },
            "WarehouseID": { "value": "WHOLESALE" },
            "UOM": { "value": "LB" }
        }
    ],
    "Hold": { "value": false }
}
```

Step 2: Create a Purchase Receipt from the Purchase Order

You create a purchase receipt from the purchase order that you just created by using the following code.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "VendorID": { "value": "GOODFRUITS" },
    "Location": { "value": "MAIN" },
    "Details": [
        {
            "POOrderNbr": { "value": "000030" },
            "POOrderType": { "value": "Normal" }
        }
    ]
}
```

Creation of a Shipment with Allocations and Package Specifications

If you are using the contract-based REST API to integrate Acumatica ERP with external systems, these external systems can create a shipment with allocations and package specifications.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create the Sales Orders That Will Be Added to a Shipment

First, you need to create the sales orders that you will add to a shipment. In this example, you will create two sales orders.

You can use the following example of an HTTP request to create a sales order of two small jars of jam.

```
PUT ?$expand=Details&$select=CustomerID,Details/Branch,Details/InventoryID,
    Details/OrderQty,OrderNbr HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "GOODFOOD" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APJAM08" },
            "OrderQty": { "value": 2 },
            "UOM": { "value": "PIECE" },
            "WarehouseID": { "value": "WHOLESALE" }
        }
    ]
}
```

You can also use the following example of an HTTP request to create a sales order of a box containing six bigger jars of jam.

```
PUT ?$expand=Details&$select=CustomerID,Details/Branch,Details/InventoryID,
    Details/OrderQty,OrderNbr HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "GOODFOOD" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APJAM32" },
            "OrderQty": { "value": 1 },
            "UOM": { "value": "BOX" },
            "WarehouseID": { "value": "WHOLESALE" }
```

```
        }
    ]
}
```

Step 2: Create a Shipment for the Two Sales Orders with Allocations and a Package

You can use the following example of an HTTP request to create in one call a shipment for the two created sales orders. For the first sales order, you take one jar of jam from the `L2R3S1` location and another jar of jam from the `L3R2S1` location. Also, you indicate that all jars of jam of both sales orders should be packed into one large box.

If you intend to use the `Picked` field of the `Shipment` entity, note that its value may be incorrect in scenarios other than batch picking or wave picking.

```
PUT ?$expand=Details,Details/Allocations,Packages,Packages/PackageContents HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "GOODFOOD" },
    "Details": [
        {
            "OrderNbr": { "value": "000071" },
            "OrderType": { "value": "SO" },
            "OrderLineNbr": { "value": 1 },
            "Allocations": [
                {
                    "InventoryID": { "value": "APJAM08" },
                    "LocationID": { "value": "L2R3S1" },
                    "Qty": { "value": 1 },
                    "UOM": { "value": "PIECE" }
                },
                {
                    "InventoryID": { "value": "APJAM08" },
                    "LocationID": { "value": "L3R2S1" },
                    "Qty": { "value": 1 },
                    "UOM": { "value": "PIECE" }
                }
            ]
        },
        {
            "OrderNbr": { "value": "000072" },
            "OrderType": { "value": "SO" },
            "OrderLineNbr": { "value": 1 },
            "Allocations": [
                {
                    "InventoryID": { "value": "APJAM32" },
                    "LocationID": { "value": "L1R3S2" },
                    "Qty": { "value": 6 },
                    "UOM": { "value": "PIECE" }
                }
            ]
        }
    ],
    "LocationID": { "value": "MAIN" },
    "Operation": { "value": "Issue" },
```

```
    "Packages": [
        {
            "BoxID": { "value": "LARGE" },
            "UOM": { "value": "KG" },
            "Weight": { "value": 15 },
            "PackageContents": [
                {
                    "InventoryID": { "value": "APJAM08" },
                    "LotSerialNbr": { "value": "" },
                    "Quantity": { "value": 1 },
                    "UOM": { "value": "PIECE" },
                    "OrigOrderType": { "value": "SO" },
                    "OrigOrderNbr": { "value": "000071" }
                },
                {
                    "InventoryID": { "value": "APJAM08" },
                    "LotSerialNbr": { "value": "" },
                    "Quantity": { "value": 1 },
                    "UOM": { "value": "PIECE" },
                    "OrigOrderType": { "value": "SO" },
                    "OrigOrderNbr": { "value": "000071" }
                },
                {
                    "InventoryID": { "value": "APJAM32" },
                    "LotSerialNbr": { "value": "" },
                    "Quantity": { "value": 6 },
                    "UOM": { "value": "PIECE" },
                    "OrigOrderType": { "value": "SO" },
                    "OrigOrderNbr": { "value": "000072" }
                }
            ]
        }
    ],
    "WarehouseID": { "value": "WHOLESALE" }
}
```

Creation of a Purchase Receipt with Allocations

If you are using the contract-based REST API to integrate Acumatica ERP with external systems, these external systems can create a purchase receipt with allocations in Acumatica ERP in a single API call. For details about the creation of purchase receipts, see *Purchases for Sale: General Information*.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make

sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create a Purchase Order

You use the following code to create a purchase order and open it.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseOrder
Accept: application/json
Content-Type: application/json

{
    "VendorID": { "value": "GLORYFRUIT" },
    "Type": { "value": "Normal" },
    "Date": { "value": "2018-01-30" },
    "Description": { "value": "Purchase of apples, 20 lb" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APPLES" },
            "WarehouseID": { "value": "WHOLESALE" },
            "OrderQty": { "value": 20 },
            "UnitCost": { "value": 9.95 }
        }
    ],
    "Hold": { "value": false }
}
```

Step 2: Create a Purchase Receipt with Allocations

Next, you create a purchase receipt with allocations for the order as follows.

```
PUT ?$expand=Details,Details/Allocations HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/PurchaseReceipt
Accept: application/json
Content-Type: application/json

{
    "Type": { "value": "Receipt" },
    "VendorID": { "value": "GLORYFRUIT" },
    "Details": [
        {
            "InventoryID": { "value": "APPLES" },
            "LotSerialNbr": { "value": "" },
            "Warehouse": { "value": "WHOLESALE" },
            "Allocations": [
                {
```

```
                    "ExpirationDate": { "value": "2021-05-29T00:00:00+03:00" },
                    "Qty": { "value": 10 },
                    "LotSerialNbr": { "value": "FRT000862" }
                },
                {
                    "ExpirationDate": { "value": "2021-05-29T00:00:00+03:00" },
                    "Qty": { "value": 10 },
                    "LotSerialNbr": { "value": "FRT000877" }
                }
            ]
        }
    ]
}
```

Project Accounting

You can use the contract-based REST API for the integration of Acumatica ERP projects with external systems. For details on projects and project accounting, see *Projects*.

Creation of a Pro Forma Invoice

If you are using the contract-based REST API to integrate Acumatica ERP with external systems, these external systems can create pro forma invoices and send them by email. For details about pro forma invoices, see *Pro Forma Invoices: General Information*.

For a pro forma invoice to be created from a project, the project must have `Customer`, `BillingRule`, `BillingPeriod`, and `NextBillingDate` specified, and must have the *Active* status. Because of data validation in the project, `NextBillingDate` cannot be specified for a project with the *In Planning* status, and you cannot change the customer in a project with the *Active* status. Therefore, these settings can be specified only in multiple API calls, as shown in the code examples below.

A `ProFormaInvoice` entity can be created through the invocation of the `RunProjectBilling` action of the `Project` entity. Because email settings are not mapped to any fields of the `Project` entity, you have to prepare a project template with the specified email settings on the *Project Templates* (PM208000) form and then use this template for the creation of the project through the API. The project template can also contain preconfigured project tasks, as is the case with the *PROGRESS* template, which is preconfigured in the *U100* dataset and used in this example. For details about project templates, see *Project Templates and Common Tasks: General Information*.

Testing of the Requests

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

> In the request examples below, <Acumatica ERP instance URL> is the URL of the Acumatica ERP instance (such as https://my.acumatica.com/MyInstance). You can omit the instance name in the URL (such as https://my.acumatica.com) if the instance is installed in the root of the website.

Step 1: Create a Project

You first need to create a project from the project template and specify the Customer, BillingRule, and BillingPeriod settings of the project as follows.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

{
    "ProjectID" : {"value" : "TESTPR3"},
    "ProjectTemplateID" : {"value" : "PROGRESS"},
    "Customer" : {"value" : "COFFEESHOP"},
    "BillingAndAllocationSettings" :
    {
        "BillingRule" : {"value" : "PROGRESS"},
        "BillingPeriod" : {"value" : "Month"},
    }
}
```

Step 2: Make the Project Active

You make the project active, as shown in the following code.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

{
    "ProjectID" : {"value" : "TESTPR3"},
    "Hold" : {"value" : false}
}
```

Step 3: Specify the Next Billing Date for the Project

You specify NextBillingDate for the project, as shown in the following code.

```
PUT ?$expand=BillingAndAllocationSettings HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

{
    "ProjectID" : {"value" : "TESTPR3"},
    "BillingAndAllocationSettings" :
    {
        "NextBillingDate" : {"value" : "2024-06-15"}
```

```
    }
}
```

Step 4: Retrieve a Project Task

You retrieve the *PHASE1* project task associated with the created project as follows.

```
GET ?$filter=ProjectID%20eq%20'TESTPR3'%20and%20ProjectTaskID%20eq%20'PHASE1' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProjectTask
Accept: application/json
Content-Type: application/json
```

Step 5: Activate a Project Task

You activate the project task as follows.

```
POST /Activate HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProjectTask
Accept: application/json
Content-Type: application/json

{
    "entity":
    {
        "ProjectID": {
            "value": "TESTPR3"
        },
        "ProjectTaskID": {
            "value": "PHASE1"
        }
    },
    "parameters": {}
}
```

Step 6: Specify the Progress of the Project Task

You use the following code to specify the progress of the project task.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProjectBudget
Accept: application/json
Content-Type: application/json

{
  "ProjectTaskID" : {"value" : "PHASE1"},
  "ProjectID" : {"value" : "TESTPR3"},
  "InventoryID" : {"value" : "<N/A>"},
  "Completed" : {"value" : 25},
  "PendingInvoiceAmount" : {"value" : 725}
}
```

Step 7: Invoke Project Billing

You invoke project billing to create a pro forma invoice as follows.

```
POST /RunProjectBilling HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

{
  "entity" : {
    "ProjectID": {
        "value": "TESTPR3"
    }
  }
}

HTTP/1.1 202 Accepted
Location: [/<Acumatica ERP instance URL>]/entity/Default/24.200.001/
    Project/RunProjectBilling/status/6952c6d1-04be-4330-a26e-c6b855ba332c
```

Step 8: Monitor the Operation Status

You use the following code to monitor the status of the operation until the system returns the *204 No Content* code.

```
GET /RunProjectBilling/status/6952c6d1-04be-4330-a26e-c6b855ba332c HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json

HTTP/1.1 204 No Content
```

Step 9: Retrieve the List of Pro Forma Invoices

You obtain the list of pro forma invoices of the project (which currently contains only one pro forma invoice) by adding *$expand=Invoices* to the endpoint address, as shown in the following code. For details about parameters, see *$expand Parameter*.

```
GET /TESTPR3?$expand=Invoices HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Project
Accept: application/json
Content-Type: application/json
```

Step 10: Send the Pro Forma Invoice by Email

You use the following code to send the pro forma invoice by email.

```
POST /EmailProFormaInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProFormaInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
    "RefNbr": {
        "value": "000019"
    }
  }
}
```

```
HTTP/1.1 202 Accepted
Location: [/<Acumatica ERP instance URL>]/entity/Default/24.200.001/
    ProFormaInvoice/EmailProFormaInvoice/status/a4caa455-0eed-4c11-a5a9-2a8333e53db1
```

Step 11: Monitor the Sending Operation Status

You use the following code to monitor the status of the operation.

```
GET /EmailProFormaInvoice/status/a4caa455-0eed-4c11-a5a9-2a8333e53db1 HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/ProFormaInvoice
Accept: application/json
Content-Type: application/json

HTTP/1.1 204 No Content
```

Management of Account Groups

If you are using the contract-based REST API to integrate Acumatica ERP with external systems, these external systems can create, modify, and remove account groups. For more information about account groups, see *Account Groups: General Information*.

The `AccountGroup` entity supports the creation, retrieval, update, and removal of the entity itself; however, you cannot modify the list of accounts of a particular account group by using the `AccountGroup` entity. Instead, you have to use the `AccountGroup` property in the `Account` entity. You can use the `DefaultAccountID` property of the `AccountGroup` entity to specify the default account for the group.

The removal of the default account from the group does not cause the `DefaultAccountID` property to be updated automatically. If you remove the default account from the group, you have to update the `DefaultAccountID` property.

Testing of the Requests

Before you test the code below, you need to configure your client application and the Acumatica ERP instance to be used as follows:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Projects* feature is enabled.
3. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
4. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create an Account Group

You start by creating an account group with the *ACCG02* identifier, as shown in the following code.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/AccountGroup
Accept: application/json
Content-Type: application/json

{
  "AccountGroupID" : {"value" : "ACCG02"},
  "Description" : {"value" : "Test Account Group"}
}
```

Step 2: Add Accounts to the Account Group

Now you need to add accounts to the account group. You add the *40000* account to the *ACCG02* account group as follows.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Account
Accept: application/json
Content-Type: application/json

{
  "AccountCD" : {"value" : "40000"},
  "AccountGroup" : {"value" : "ACCG02"}
}
```

Next, you add the *40010* account to the *ACCG02* account group, as shown in the following code.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Account
Accept: application/json
Content-Type: application/json

{
    "AccountCD" : {"value" : "40010"},
    "AccountGroup" : {"value" : "ACCG02"}
}
```

Step 3: Specify the Default Account of the Account Group

You need to specify the *40000* account as the default account of the *ACCG02* account group, as shown in the following code example.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/AccountGroup
Accept: application/json
Content-Type: application/json

{
  "DefaultAccountID" : {"value" : "40000"},
  "AccountGroupID" : {"value" : "ACCG02"}
```

```
}
```

Step 4: Retrieve the List of Accounts of the Account Group

You need to obtain the list of accounts of the *ACCG02* account group as follows.

```
GET ?$filter=AccountGroup%20eq%20'ACCG02'&$select=AccountCD HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Account
Accept: application/json
Content-Type: application/json
```

Step 5: Remove an Account from the Group

Finally, you remove the *40010* account from the *ACCG02* account group, as shown in the following code.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Account
Accept: application/json
Content-Type: application/json

{
  "AccountCD" : {"value" : "40010"},
  "AccountGroup" : {"value" : null}
}
```

Related Links

- *Account Groups: General Information*

POS Systems

A point-of-sale (POS) system is an electronic system that is used to record the sales, payment, and return transactions of a retail store. The POS system can be operated by a cashier, or it can be a self-service terminal where customers perform all operations by themselves.

Examples of the integration of Acumatica ERP with POS systems include the following:

- *Entry of a Direct Sales Invoice*
- *Entry of a Direct Sales Invoice Along with a Return*
- *Entry of a Credit Memo with Positive and Negative Lines*
- *Entry of a Direct Sales Invoice in a Non-Default Currency*
- *Entry of a Direct Sales Invoice for a Shipped Order and Return*
- *Entry of a Direct Sales Invoice for an Unshipped Sales Order*
- *Entry of a Direct Sales Invoice for a Partially Shipped Sales Order*
- *Entry of a Credit Memo for an Unshipped RMA Order*

Entry of a Direct Sales Invoice

A point-of-sale (POS) system can create and process direct sales invoices—that is, sales invoices created on the *Invoices* (SO303000) form for which neither a sales order nor a shipment has been created. The POS system creates

the direct sales invoice on the *Invoices* (SO303000) form and the payment on the *Payments and Applications* (AR302000) form for it. It then applies this payment to the invoice and releases the invoice.

In this topic, you will implement HTTP requests for the following user scenario. A customer comes to the store, picks up a number of items (including a jar of cherry jam, which has a serial number assigned), and orders billboard advertising. The customer would like to buy the items and pay for the order. In the POS system, one invoice is created for this operation.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Assign a Tax Zone to a Customer

For the payment amount that is used in this example, the *FRUITICO* customer is assigned the *NYSTATE* tax zone. You use the following example of an HTTP request to assign the tax zone to the customer.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Customer
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "TaxZone": {"value": "NYSTATE"}
}
```

Step 2: Create a Payment

You use the following example of an HTTP request to create a payment that is sufficient to pay the direct sales invoice, which has not yet been created, and to remove the payment from hold. In this example, you create a payment before you create a direct sales order in order to show how to use one call to create a sales order and apply a payment to it.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
```

```
Content-Type: application/json

{
    "CashAccount": {"value": "10250ST"},
    "CustomerID": {"value": "FRUITICO"},
    "Hold": {"value": false},
    "PaymentAmount": {"value": 1235.27},
    "Type": {"value": "Payment"}
}
```

Step 3: Release the Payment

You use the following example of an HTTP request to release the payment that was created in the previous step (whose reference number is assumed to be *000076*).

```
POST /ReleasePayment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000076"},
      "Type": {"value": "Payment"}
      },
  "parameters" : {}
}
```

At this point, you would check the status of this operation (as well as the status of another operation in this topic) by performing actions similar to those described in *Execute an Action That Is Present in an Endpoint*. For simplicity, the checking requests are not provided here.

Step 4: Create a Direct Sales Invoice

You use the following example of an HTTP request to create a direct sales invoice containing the customer's purchased items and ordered service. In the request, you will also specify the billing settings, apply the previously released payment to the invoice, and remove the invoice from hold.

```
PUT ?$expand=ApplicationsInvoice,Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "Type": {"value": "Invoice"},
    "Hold": {"value": false},
    "Details":
    [
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "CHERJAM32"},
            "Qty": {"value": 1},
            "UOM": {"value": "PIECE"},
            "LotSerialNbr": {"value": "JM2301290000"}
        },
```

```
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APJAM32"},
            "Qty": {"value": 2},
            "UOM": {"value": "BOX"}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "ADVERT"},
            "Qty": {"value": 1},
            "UnitPrice": {"value": 1000},
            "UOM": {"value": "DAY"}
        }
    ],
    "BillingSettings":
    {
        "BillToAddressOverride": {"value": true},
        "BillToAddress":
        {
            "AddressLine1": {"value": "Fillmore Str"},
            "City": {"value": "San Francisco"},
            "State": {"value": "CA"}
        }
    },
    "ApplicationsInvoice":
    [
        {
            "DocType": {"value": "Payment"},
            "AdjustingDocReferenceNbr": {"value": "000076"},
            "AmountPaid": {"value": 1235.27}
        }
    ]
}
```

Step 5: Release the Direct Sales Invoice

You use the following example of an HTTP request to release the direct sales invoice that you created in the previous step (whose reference number is assumed to be *000114*).

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000114"},
      "Type": {"value": "Invoice"}
      },
  "parameters" : {}
}
```

Entry of a Direct Sales Invoice Along with a Return

A point-of-sale (POS) system can create and process direct sales invoices (that is, invoices for which neither a sales order nor a shipment has been created) and include in these invoices lines for returned items that were previously sold. The POS system creates the direct sales invoice on the *Invoices* (SO303000) form and releases the invoice.

In this topic, you will implement HTTP requests for the following user scenario. A customer comes to the store and picks up a number of items (including items that are tracked by their serial numbers). The customer would like to buy these items and to return a previously purchased box of pens. In the POS system, one invoice is created for this operation. The customer pays the difference between the sale and the return.

For simplicity, this example does not include the creation of the payment or the application of the payment to the invoice. You can find an example showing how to create and apply a payment related to a direct sales invoice in *Entry of a Direct Sales Invoice*.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create a Direct Sales Invoice

You use the following example of an HTTP request to create a direct sales invoice containing the picked items and the items to be returned, and release the invoice from hold.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "Type": {"value": "Invoice"},
    "Hold": {"value": false},
    "Details":
    [
```

```
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "CHERJAM32"},
            "Qty": {"value": 1},
            "UOM": {"value": "PIECE"},
            "LotSerialNbr": {"value": "JM2302010003"}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "BANANAS"},
            "WarehouseID": {"value": "WHOLESALE"},
            "Location": {"value": "F2S2"},
            "Qty": {"value": 5},
            "UOM": {"value": "LB"},
            "UnitPrice": {"value": 1},
            "LotSerialNbr": {"value": "FR200384"}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "PEN"},
            "Location": {"value": "MAIN"},
            "Qty": {"value": -1},
            "UnitPrice": {"value": 2},
            "UOM": {"value": "PIECE"}
        }
    ]
}
```

Step 2: Release the Invoice

You use the following example of an HTTP request to release the invoice that you created in the previous step (whose reference number is assumed to be *000116*).

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000116"},
      "Type": {"value": "Invoice"}
      },
  "parameters" : {}
}
```

You would then check the status of this operation by performing similar actions to those described in *Execute an Action That Is Present in an Endpoint*. For simplicity, the checking requests are not provided here.

Entry of a Credit Memo with Positive and Negative Lines

A point-of-sale (POS) system can create and process credit memos with positive lines (for the returned items) and negative lines (for the purchased items). A credit memo is created on the *Invoices* (SO303000) form instead of a direct sales invoice if the payment amount of the returned items is greater than the payment amount of the newly

purchased items. The POS system creates the credit memo, links the sales invoice with the returned lines to the credit memo, and releases the credit memo.

In this topic, you will implement HTTP requests for the following user scenario. A customer comes to the store and picks up a jar of apple jam. The customer would like to buy it and to return 10 previously purchased jars of cherry jam. The price of the returned items is greater than the price of the purchased item. In the POS system, one invoice is created for the whole operation. The customer is given the difference between the returned items and the sold items.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.
3. On the *Invoices* (SO303000) form, create a direct sales invoice for selling 10 *CHERJAM32* items and other goods to the *FRUITICO* customer. The 10 *CHERJAM32* items must be in line *1* of the invoice. Make a note of the invoice number. In the example, the invoice number is assumed to be *000114*; you should instead use the number of the sales invoice you create.
4. On the form toolbar, click **Release** to release the invoice.
5. On the More menu (under **Processing**), click **Pay** to create a payment.
6. On the *Payments and Applications* (AR302000) form, which opens, save the payment. On the form toolbar, click **Remove Hold** and then click **Release**.
7. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
8. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create a Credit Memo

You use the following example of an HTTP request to create a credit memo and release it from hold. With the credit memo, the following is done:

- The items in line *1* of the *000114* invoice are returned.

  > Replace *000114* with the number of the invoice you have created.

- One item of the *APJAM08* goods is purchased.
- The *000114* invoice is attached to the credit memo.
- A total of $80 is credited from that invoice.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
```

```
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "Type": {"value": "Credit Memo"},
    "Hold": {"value": false},
    "Details":
    [
        {
            "OrigInvNbr": { "value": "000114" },
            "OrigInvType": { "value": "Invoice" },
            "OrigInvLineNbr": { "value": 1 },
            "Qty": { "value": 10 }
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APJAM08"},
            "WarehouseID": {"value": "WHOLESALE"},
            "Location": {"value": "MAIN"},
            "Qty": {"value": -1},
            "UOM": {"value": "PIECE"}
        }
    ],
    "ApplicationsCreditMemo":
    [
        {
            "Customer": {"value": "FRUITICO"},
            "DocType": {"value": "Invoice"},
            "ReferenceNbr": {"value": "000114"},
            "AmountPaid": {"value": 80}
        }
    ]
}
```

Step 2: Release the Credit Memo

You use the following example of an HTTP request to release the credit memo that you created in the previous step (whose reference number is assumed to be *000116*).

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000116"},
      "Type": {"value": "Credit Memo"}
      },
  "parameters" : {}
}
```

You need to check the status of this operation by performing similar actions to those described in *Execute an Action That Is Present in an Endpoint*. For simplicity, the checking requests are not provided here.

Entry of a Direct Sales Invoice in a Non-Default Currency

A point-of-sale (POS) system can create and process direct sales invoices—that is, invoices for which neither a sales order nor a shipment has been created—in a currency that differs from the default currency of the customer account. The POS system creates the direct sales invoice in the needed currency and releases the invoice.

For simplicity, this example does not include the creation of the payment or the application of the payment to the invoice. You can find an example showing how to create and apply a payment in *Entry of a Direct Sales Invoice*.

In this topic, you will implement HTTP requests for the following user scenario. A customer comes to the U.S. store and picks up two boxes of large jars of apple jam. The customer would like to buy them, return a previously purchased small jar of apple jam, cancel the order of the cleaning service, and pay in euros. In the POS system, one invoice is created for this operation. The customer pays the difference between the sale and return.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Multicurrency Accounting*, and *Advanced SO Invoices* features are enabled.
5. Configure multicurrency support as follows:
   a. On the *Currencies* (CM202000) form, open *EUR*, select the **Active** and **Use for Accounting** boxes, and specify the following values in the other boxes:
      - **Realized Gain Account**: *83100*
      - **Realized Loss Account**: *83100*
      - **Unrealized Gain Account**: *84000*
      - **Unrealized Loss Account**: *84000*
      - **Revaluation Gain Account**: *83200*
      - **Revaluation Loss Account**: *83200*
      - **Rounding Gain Account**: *83100*
      - **Rounding Loss Account**: *83100*

      Save your changes.
   b. On the *Currency Management Preferences* (CM101000) form, click **Save**.
   c. On the **Currency Rate Entry** tab of the *Currency Rates* (CM301000) form, add a row with the following settings:
      - **From Currency**: *EUR*
      - **Currency Rate Type**: *SPOT*
      - **Currency Effective Date**: Today
      - **Currency Rate**: 1.1

- **Mult./Div.**: *Multiply*

Save your changes.

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Configure the Customer

You use the following example of an HTTP request to enable the currency overriding and the rate overriding for the *FRUITICO* customer and to assign the *SPOT* currency rate type to the customer.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Customer
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "EnableCurrencyOverride": {"value": true},
    "EnableRateOverride": {"value": true},
    "CurrencyRateType": {"value": "SPOT"}
}
```

Step 2: Create an Invoice

You use the following example of an HTTP request to create an invoice containing the items the customer is buying and those the customer is returning, and release the invoice from hold.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "Type": {"value": "Invoice"},
    "Currency": {"value": "EUR"},
    "Hold": {"value": false},
    "Details":
    [
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APJAM96"},
            "WarehouseID": {"value": "WHOLESALE"},
            "Qty": {"value": 2},
            "UOM": {"value": "JBOX"},
            "Location": {"value": "L3R1S2"}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APJAM08"},
            "WarehouseID": {"value": "WHOLESALE"},
            "Location": {"value": "L1R1S2"},
```

```
            "Qty": {"value": -1},
            "UOM": {"value": "PIECE"}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "CLEANING"},
            "Qty": {"value": -1},
            "UOM": {"value": "HOUR"}
        }
    ]
}
```

Step 3: Release the Invoice

You use the following example of an HTTP request to release the invoice that you created at the previous step (whose reference number is assumed to be *000118*).

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000118"},
      "Type": {"value": "Invoice"}
      },
  "parameters" : {}
}
```

At this point, you would check the status of this operation by performing similar actions to those described in *Execute an Action That Is Present in an Endpoint*. For simplicity, the checking requests are not provided here.

Entry of a Direct Sales Invoice for a Shipped Order and Return

A point-of-sale (POS) system can create and process direct sales invoices that contain both of the following types of lines:

- Lines that have been linked to a return merchandise authorization (RMA) order—that is, a return order of the *RM* type created on the *Sales Orders* (SO301000) form
- Lines that have not been linked to any order

Either type of line can reflect a newly sold item or a returned item. To process these invoices, the POS system performs the following steps:

1. For the returned items that are linked to a sales order and shipment, creates a new RMA order (return order of the *RM* type).
2. In the RMA order, adds the previously issued sales invoice that contains the returned items.
3. In the RMA order, adds the newly ordered items.
4. Creates shipments for the newly ordered items and confirms these shipments.
5. Creates shipments with the *Receipt* operation (incoming shipments) for the returned items and confirms these shipments.
6. Creates a sales invoice and releases the invoice.

For simplicity, this example does not include the creation of the customer's payment or the application of the payment to the sales invoice. You can find an example showing how to create and apply a payment in *Entry of a Direct Sales Invoice*.

In this topic, you will implement HTTP requests for the following user scenario. In the online shop, a customer buys a 32-ounce glass bottle, but then the customer decides to return the bottle. In the online shop, the customer creates an order that includes both the returned item and a 16-ounce glass bottle that the customer wants to buy. Thus, the customer would like to buy a smaller bottle and to return a larger one. In a POS system, one invoice is created for the whole operation.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management* and *Advanced SO Invoices* features are enabled.
5. Prepare a sales order, shipment, and an invoice as follows:

   > If you use the `IntegrationDevelopmentGuide.postman_collection.json` Postman collection for testing, you do not need to perform the following configuration steps because they are performed in the pre-request script.

   a. On the *Sales Orders* (SO301000) form, create a sales order to sell one *SWB-32OZ-GBT* item from the *Retail* warehouse to the *FRUITICO* customer.
   b. On the form toolbar, click **Create Shipment** to create a shipment for the sales order.
   c. In the **Specify Shipment Parameters** dialog box, which opens, specify today's date as the shipment date and *Retail* as the warehouse, and click **OK**.
   d. On the *Shipments* (SO302000) form, which opens, click **Confirm Shipment** and then click **Prepare Invoice**.
   e. On the *Invoices* (SO303000) form, which opens, click **Release**. In this example, the invoice number is supposed to be *000126*.

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create an RMA Order

You use the following example of an HTTP request to create an RMA order (an order of the *RM* type) for the return of the *SWB-32OZ-GBT* item that the *FRUITICO* customer bought earlier and the sale of a *PEARJAM96* item to the customer.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "FRUITICO" },
    "OrderType": { "value": "RM" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InvoiceNbr": { "value": "000126" },
            "Operation": { "value": "Receipt" },
            "InventoryID": { "value": "SWB-32OZ-GBT" },
            "OrderQty": { "value": -1 },
            "UOM": { "value": "EA" },
            "WarehouseID": { "value": "WHOLESALE" },
            "AutoCreateIssue": { "value": false }
        },
        {
            "Branch": { "value": "HEADOFFICE" },
            "Operation": { "value": "Issue" },
            "InventoryID": { "value": "PEARJAM96" },
            "OrderQty": { "value": 1 },
            "UOM": { "value": "PIECE" },
            "WarehouseID": { "value": "RETAIL" },
            "AutoCreateIssue": { "value": false }
        }
    ]
}
```

Step 2: Correct the Order of the RM Type

You use the following example of an HTTP request to set the value of the `AutoCreateIssue` field of the `SalesOrder` object to *false* to avoid the creation of an additional detail line in the sales order of the *RM* type. You use the `id` fields to identify both the order and the detail line you are amending. You received the values of the `id` fields in the HTTP response in the previous step.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "id": "4593e6b2-8ba9-ed11-9e84-9828a61840c3",
    "Details": [
        {
            "id": "4b93e6b2-8ba9-ed11-9e84-9828a61840c3",
            "AutoCreateIssue": {"value": false}
        }
    ]
}
```

Step 3: Create a Shipment with the Receipt Operation for the Sales Order of the RM Type

You use the following example of an HTTP request to create a shipment with the *Receipt* operation to confirm that the *SWB-32OZ-GBT* item has been received from the *FRUITICO* customer. In the `entity/OrderNbr` field, you specify the number of the sales order of the *RM* type that was created in Step 1. In the `parameters/WarehouseID` field, you specify the warehouse to which the *SWB-32OZ-GBT* item is returned.

```
POST /SalesOrderCreateReceipt HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "OrderNbr": {"value": "000127"},
    "OrderType":  {"value": "RM"}
  },
  "parameters": {
    "ShipmentDate":  {"value": "2023-02-11T00:00:00+03:00"},
    "WarehouseID":  {"value": "WHOLESALE"}
  }
}
```

You would then check the status of this operation (as well as the statuses of other operations in this topic) by performing similar actions to those described in *Execute an Action That Is Present in an Endpoint*. For simplicity, the checking requests are not provided here.

Step 4: Create a Shipment for the RMA Order

You use the following example of an HTTP request to create a shipment for the RMA order (the order of the *RM* type). In the `entity/OrderNbr` field, you specify the number of the RMA order created in Step 1. In the `parameters/WarehouseID` field, you specify the warehouse from which the *PEARJAM96* item is taken (see Step 1).

```
POST /SalesOrderCreateShipment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "OrderNbr": {"value": "000127"},
    "OrderType":  {"value": "RM"}
  },
  "parameters": {
    "ShipmentDate":  {"value": "2023-02-11T00:00:00+03:00"},
    "WarehouseID":  {"value": "RETAIL"}
  }
}
```

Step 5: Retrieve the Shipments' Numbers of the RMA Order

You use the following example of an HTTP request to retrieve the numbers of the shipments of the RMA order. In the URL, the number of the order of the *RM* type is specified (which in this example is *000127*).

```
GET /RM/000127?$expand=Shipments&
    $select=OrderNbr,OrderType,Shipments/ShipmentNbr HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Following is an example of the HTTP response. In the `Shipments` array, the first element is the shipment with the *Receipt* operation that was created in Step 3, and the second element is the shipment that was created in Step 4.

```
{
    ...
    "Shipments": [
        {
            "id": "1a7cc873-cf08-43c8-9330-a4f27b36d3a1",
            "rowNumber": 1,
            "note": null,
            "ShipmentNbr": { "value": "000069" },
            "custom": {}
        },
        {
            "id": "8f4e7437-ad2f-408e-99f5-5d9f3388b60f",
            "rowNumber": 2,
            "note": null,
            "ShipmentNbr": { "value": "000070" },
            "custom": {}
        }
    ],
    ...
}
```

Step 6: Confirm the Shipment with the Receipt Operation of the RMA Order

You use the following example of an HTTP request to confirm the shipment with the *Receipt* operation that was created for the RMA order. In the `entity/ShipmentNbr` field, you specify the number of the shipment that you learned in the previous step.

```
POST /ConfirmShipment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "ShipmentNbr": {"value": "000069"},
    "ShipmentType":  {"value": "Shipment"}
  },
  "parameters": {
  }
}
```

Step 7: Prepare a Credit Memo for the Shipment with the Receipt Operation

You use the following example of an HTTP request to prepare a credit memo for the shipment with the *Receipt* operation that was confirmed in the previous step. The `ShipmentNbr` field contains the same value as in the previous step.

```
POST /PrepareInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "ShipmentNbr": {"value": "000069"},
    "ShipmentType":  {"value": "Shipment"}
  },
  "parameters": {
  }
}
```

Step 8: Retrieve the Number of the Credit Memo

You use the following example of an HTTP request to retrieve the number of the credit memo that was prepared in the previous step.

```
GET ?$expand=Orders&$filter=ShipmentNbr eq '000069'&
    $select=ShipmentNbr,Orders/InvoiceNbr,Orders/InvoiceType HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json
```

Step 9: Release the Credit Memo

You use the following example of an HTTP request to release the credit memo that was prepared in Step 7. The HTTP response received in the previous step contains the reference number of the credit memo in the `Orders/InvoiceNbr` field.

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000128"},
      "Type": {"value": "Credit Memo"}
      },
  "parameters" : {}
}
```

Step 10: Confirm the Shipment of the RMA Order

You use the following example of an HTTP request to confirm the shipment that is created for the RMA order. In the `entity/ShipmentNbr` field, you specify the number of the shipment that was learned in Step 5.

```
POST /ConfirmShipment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
```

```
  "entity":{
    "ShipmentNbr": {"value": "000070"},
    "ShipmentType":  {"value": "Shipment"}
  },
  "parameters": {
  }
}
```

Step 11: Prepare a Sales Invoice for the Shipment

You use the following example of an HTTP request to prepare a sales invoice for the shipment that was confirmed in the previous step. The `ShipmentNbr` field contains the same value as in the previous step.

```
POST /PrepareInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "ShipmentNbr": {"value": "000069"},
    "ShipmentType":  {"value": "Shipment"}
  },
  "parameters": {
  }
}
```

Step 12: Retrieve the Number of the Prepared Sales Invoice

You use the following example of an HTTP request to retrieve the number of the sales invoice that was prepared in the previous step.

```
GET ?$expand=Orders&$filter=ShipmentNbr eq '000069'&
    $select=ShipmentNbr,Orders/InvoiceNbr,Orders/InvoiceType HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json
```

Step 13: Release the Sales Invoice

You use the following example of an HTTP request to release the sales invoice that was prepared in Step 11. The HTTP response received in the previous step contains the reference number of the credit memo in the `Orders/InvoiceNbr` field.

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000128"},
      "Type": {"value": "Credit Memo"}
      },
  "parameters" : {}
```

```
}
```

Entry of a Direct Sales Invoice for an Unshipped Sales Order

A point-of-sale (POS) system can create and process sales invoices on the *Invoices* (SO303000) form that contain both of the following types of lines:

- Lines that have not been linked to any sales order or shipment
- Lines for which a sales order has been created and a shipment has not been created

Either type of line can include information about newly bought items or returned items. To process these invoices, the POS system performs the following steps:

1. Creates a new return merchandise authorization (RMA) order—that is, a return order of the *RM* type created on the *Sales Orders* (SO301000) form—with the new items and returned items.
2. Creates a payment.
3. Creates a sales invoice and adds detail lines for both the new items and the returned items.
4. Applies the payment to the invoice.
5. Releases the invoice. As a result of this operation, the sales order gets the *Completed* status in Acumatica ERP. The SO invoice is added to the **Shipments** tab of the *Sales Orders* (SO301000) form and is treated by the system as a shipment (that is, the invoice updates shipped quantity in the sales order lines and updates inventory).

The sales order may not be completed if it was closed by the SO invoice partially (that is, if some lines of the sales order are not shipped or billed).

In this topic, you will implement HTTP requests for the following user scenario. A customer comes to the store and through a self-service terminal creates a sales order to buy a 32-ounce glass bottle and to return a 16-ounce glass bottle that the customer previously purchased in the store (this item is specified in the initial sales order). In a POS system, one invoice is created for the whole operation.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data* in the Installation Guide.
2. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.
3. On the *Sales Orders* (SO301000) form, create a sales order to sell an *ORG-16OZ-GBT* item from the *Retail* warehouse to the *FRUITICO* customer.
4. On the *Invoices* (SO303000) form, create a new sales invoice (its type must be *Invoice*) for the *FRUITICO* customer, and on the **Details** tab toolbar, click **Add Order** and add the sales order that was just created. In this example, the invoice number is assumed to be *000130*.
5. On the form toolbar, click **Release** to release the created invoice.
6. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.

7. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.

> In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create an RMA Order

You use the following example of an HTTP request to create an RMA order that contains the *ORG-16OZ-GBT* item (from the sales order that is created in the *Testing of the Requests* section) that the customer wants to return and an *ORG-32OZ-GBT* item that the customer wants to buy.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "FRUITICO" },
    "OrderType": { "value": "RM" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InvoiceNbr": { "value": "000130" },
            "Operation": { "value": "Receipt" },
            "InventoryID": { "value": "ORG-16OZ-GBT" },
            "OrderQty": { "value": -1 },
            "UOM": { "value": "EA" },
            "WarehouseID": { "value": "RETAIL" },
            "AutoCreateIssue": { "value": false }
        },
        {
            "Branch": { "value": "HEADOFFICE" },
            "Operation": { "value": "Issue" },
            "InventoryID": { "value": "ORG-32OZ-GBT" },
            "OrderQty": { "value": 1 },
            "UOM": { "value": "EA" },
            "WarehouseID": { "value": "RETAIL" },
            "AutoCreateIssue": { "value": false }
        }
    ]
}
```

Step 2: Correct the RMA Order

You use the following example of an HTTP request to set the value of the `AutoCreateIssue` field of the `SalesOrder` object to *false*to avoid the creation of an additional detail line in the sales order of the *RM* type. You use the `id` fields to identify both the sales order and the detail line you are amending. You received the values of the `id` fields in the HTTP response in the previous step.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
```

```
Content-Type: application/json

{
    "id": "bbbc7c53-e8ab-ed11-9e85-9828a61840c3",
    "Details": [
        {
            "id": "c1bc7c53-e8ab-ed11-9e85-9828a61840c3",
            "AutoCreateIssue": {"value": false}
        }
    ]
}
```

Step 3: Create a Payment

You use the following example of an HTTP request to create a payment with the paid amount that is equal to the total amount of the RMA order that was created in Step 1.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
Content-Type: application/json

{
    "CashAccount": {"value": "10250ST"},
    "CustomerID": {"value": "FRUITICO"},
    "Hold": {"value": false},
    "PaymentAmount": {"value": 8.32},
    "Type": {"value": "Payment"}
}
```

Step 4: Release the Payment

You use the following example of an HTTP request to release the payment that you created in the previous step. The reference number of the payment is assumed to be *000076*.

```
POST /ReleasePayment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000076"},
      "Type": {"value": "Payment"}
      },
  "parameters" : {}
}
```

You need to check the status of this operation (as well as the status of another operation in this topic) by performing instructions similar to those described in *Execute an Action That Is Present in an Endpoint*. For simplicity, the checking requests are not provided here.

Step 5: Create a Sales Invoice for the RMA Order

You use the following example of an HTTP request to create a sales invoice that contains both lines of the RMA order (whose order number is assumed to be *000131*) and apply the payment to the invoice.

```
PUT ?$expand=ApplicationsInvoice,Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "Type": {"value": "Invoice"},
    "Hold": {"value": false},
    "Details":
    [
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "ORG-16OZ-GBT"},
            "Location": {"value": "MAIN"},
            "Qty": {"value": -1},
            "UOM": {"value": "EA"},
            "OrderType": {"value": "RM"},
            "OrderNbr": {"value": "000131"},
            "OrderLineNbr": {"value": 1}
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "ORG-32OZ-GBT"},
            "Location": {"value": "MAIN"},
            "Qty": {"value": 1},
            "UOM": {"value": "EA"},
            "OrderType": {"value": "RM"},
            "OrderNbr": {"value": "000131"},
            "OrderLineNbr": {"value": 3}
        }
    ],
    "ApplicationsInvoice":
    [
        {
            "DocType": {"value": "Payment"},
            "AdjustingDocReferenceNbr": {"value": "000076"},
            "AmountPaid": {"value": 8.32}
        }
    ]
}
```

Step 6: Release the Sales Invoice

You use the following example of an HTTP request to release the invoice that you created in the previous step. The reference number of the invoice is assumed to be *000132*.

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
```

```
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000132"},
      "Type": {"value": "Invoice"}
      },
  "parameters" : {}
}
```

Entry of a Direct Sales Invoice for a Partially Shipped Sales Order

A point-of-sale (POS) system can create and process a direct sales invoice that contains both of the following types of lines:

- Lines for which a sales order has been created and a shipment has been created (but the shipment does not include all items of the sales order
- Lines that have not been linked to any sales order or shipment

To process an invoice of this type, the POS system performs the following steps:

1. Creates a new sales order of the *SO* type with multiple items.
2. Creates a payment.
3. Creates a sales invoice and adds detail lines for some (but not all) items from the sales order and the new items.
4. Applies the payment to the invoice.
5. Releases the invoice. At this moment, not all lines of the sales order are shipped. The sales order has the *Open* status in Acumatica ERP. On the **Shipments** tab of the *Sales Orders* (SO301000) form, for the sales order lines added to the invoice, a dummy shipment is created with the link to the invoice.
6. Creates the second payment.
7. Applies the second payment to the sales order.
8. Creates a shipment or multiple shipments for the remaining (not shipped) lines of the sales order and confirms each shipment.
9. Creates a sales invoice for each shipment and releases each invoice. As a result of the release of the invoice or invoices, the sales order gets the *Completed* status in Acumatica ERP.

The sales order may not be completed if it was closed by the sales invoice partially (that is, if some lines of the sales order are not shipped or billed).

In this topic, you will implement HTTP requests for the following user scenario. A customer comes to the store and uses a self-service terminal to create a sales order to buy a small jar of apple jam and 10 boxes of larger jars of apple jam. Then the customer picks up a large jar of cherry jam from the store shelves. The customer would like to buy the jar of apple jam (from the sales order) and the selected jar of cherry jam, and to have the 10 boxes shipped to the customer's home. In a POS system, one sales invoice is created for the purchase in the store and another invoice is created for the boxes to be shipped.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create a Sales Order

You use the following example of an HTTP request to create a sales order to sell a small jar of apple jam and 10 boxes of larger jars of apple jam to the *FRUITICO* customer.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "FRUITICO" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APJAM08" },
            "OrderQty": { "value": 1 },
            "UOM": { "value": "PIECE" },
            "WarehouseID": { "value": "WHOLESALE" }
        },
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APJAM32" },
            "OrderQty": { "value": 10 },
            "UOM": { "value": "BOX" },
            "WarehouseID": { "value": "WHOLESALE" }
        }
    ]
}
```

Step 2: Create a Payment

You use the following example of an HTTP request to create a payment for the purchase of a jar of apple jam (from the sales order) and a jar of cherry jam.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
```

```
Content-Type: application/json

{
    "CashAccount": {"value": "10250ST"},
    "CustomerID": {"value": "FRUITICO"},
    "Hold": {"value": false},
    "PaymentAmount": {"value": 22.91},
    "Type": {"value": "Payment"}
}
```

Step 3: Release the Payment

You use the following example of an HTTP request to release the payment that you created in the previous step (whose number is assumed to be *000078*).

```
POST /ReleasePayment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000078"},
      "Type": {"value": "Payment"}
      },
  "parameters" : {}
}
```

You would then check the status of this operation (as well as the statuses of other operations in this topic) by performing actions similar to those described in *Execute an Action That Is Present in an Endpoint*. For simplicity, the checking requests are not provided here.

Step 4: Create a Sales Invoice

You use the following example of an HTTP request to create a sales invoice containing the items for which the payment was created in Step 2. The payment will be applied to the invoice by this request.

```
PUT ?$expand=ApplicationsInvoice,Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "FRUITICO" },
    "Type": {"value": "Invoice"},
    "Hold": {"value": false},
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "OrderNbr": { "value": "000073" },
            "OrderType": { "value": "SO" },
            "OrderLineNbr": { "value": 1 },
            "Location": { "value": "MAIN" }
        },
        {
            "Branch": { "value": "HEADOFFICE" },
```

```
            "InventoryID": { "value": "CHERJAM32" },
            "UOM": { "value": "PIECE" },
            "Qty": { "value": 1 },
            "LotSerialNbr": { "value": "JM2302150000" },
            "Location": { "value": "MAIN" }
        }
    ],
    "ApplicationsInvoice":
    [
        {
            "DocType": {"value": "Payment"},
            "AdjustingDocReferenceNbr": {"value": "000078"},
            "AmountPaid": {"value": 22.91}
        }
    ]
}
```

Step 5: Release the Sales Invoice

You use the following example of an HTTP request to release the sales invoice that you created in the previous step.

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000133"},
      "Type": {"value": "Invoice"}
      },
  "parameters" : {}
}
```

Step 6: Create a Second Payment

You use the following example of an HTTP request to create a payment for the purchase of boxes of apple jam from the sales order that you created in Step 1.

```
PUT / HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
Content-Type: application/json

{
    "CashAccount": {"value": "10250ST"},
    "CustomerID": {"value": "FRUITICO"},
    "Hold": {"value": false},
    "PaymentAmount": {"value": 1084.4},
    "Type": {"value": "Payment"}
}
```

Step 7: Release the Payment

You use the following example of an HTTP request to release the payment that you created in the previous step (whose reference number is assumed to be *000079*).

```
POST /ReleasePayment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Payment
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000079"},
      "Type": {"value": "Payment"}
      },
  "parameters" : {}
}
```

Step 8: Apply the Second Payment to the Sales Order

You use the following example of an HTTP request to apply the payment that you created in Step 6 to the sales order.

```
PUT ?$expand=Payments HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "OrderNbr": { "value": "000073" },
    "OrderType": { "value": "SO" },
    "Payments": [
        {
            "DocType": { "value": "Payment" },
            "ReferenceNbr": { "value": "000079" },
            "AppliedToOrder": { "value": 1084.4 }
        }
    ]
}
```

Step 9: Create a Shipment for the Sales Order

You use the following example of an HTTP request to create a shipment for the sales order.

```
POST /SalesOrderCreateShipment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "OrderNbr": {"value": "000073"},
    "OrderType":  {"value": "SO"}
  },
```

```
  "parameters": {
    "ShipmentDate":  {"value": "2023-02-15T00:00:00+03:00"},
    "WarehouseID":  {"value": "WHOLESALE"}
  }
}
```

Step 10: Retrieve the Shipment Number

You use the following example of an HTTP request to learn the number of the shipment that you created for the sales order.

```
GET /SO/000073?$expand=Shipments&$select=OrderNbr,Shipments/ShipmentNbr HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Step 11: Confirm the Shipment

You use the following example of an HTTP request to confirm the shipment that you created in Step 9.

```
POST /ConfirmShipment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "ShipmentNbr": {"value": "000071"},
    "ShipmentType":  {"value": "Shipment"}
  },
  "parameters": { }
}
```

Step 12: Prepare a Sales Invoice for the Shipment

You use the following example of an HTTP request to prepare a sales invoice for the shipment that you created in Step 9.

```
POST /PrepareInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "ShipmentNbr": {"value": "000071"},
    "ShipmentType":  {"value": "Shipment"}
  },
  "parameters": { }
}
```

Step 13: Retrieve the Sales Invoice Number That Is Referenced in the Shipment

You use the following example of an HTTP request to learn the sales invoice number referenced in the shipment that you created in Step 9.

```
GET ?$expand=Orders&$select=ShipmentNbr,Orders/InvoiceNbr&
    $filter=ShipmentNbr eq '000071' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json
```

Step 14: Release the Sales Invoice

You use the following example of an HTTP request to release the sales invoice that you prepared in Step 12.

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000126"},
      "Type": {"value": "Invoice"}
      },
  "parameters" : {}
}
```

Entry of a Credit Memo for an Unshipped RMA Order

A point-of-sale (POS) system can create and process credit memos on the *Invoices* (SO303000) form that contain both of the following types of lines:

- Lines that have been linked to a return merchandise authorization (RMA) order—that is, a return order of the *RM* type created on the *Sales Orders* (SO301000) form
- Lines that have not been linked to any order

During the creation and processing of such a credit memo, the POS system creates an RMA order, creates a credit memo, adds to the credit memo the lines from the RMA order and other lines, and releases the credit memo.

In this topic, you will implement HTTP requests for the following user scenario. In the online shop, a customer creates an RMA order to buy a small jar of apple jam and to return a box of medium-sized jars of apple jam that the customer previously bought in the store. The customer then decides to procure and return the respective items in the store. Before leaving for the store, the customer decides to also return a large jar of apple jam (also bought in the store); the customer then goes to the store. In the POS system, one sales credit memo is created for the whole operation. The customer is given the difference between the return and sale.

Testing of the Requests

Before you test the code below, you need to do the following to configure your client application and the Acumatica ERP instance to be used:

1. Deploy a new Acumatica ERP instance with the *U100* dataset. For details on deploying an instance, see *Instance Deployment: To Deploy an Instance with Demo Data*.
2. To sign in to the instance in the client application, use the tenant name (which you specified when you created the instance) and the *HEADOFFICE* branch.
3. If you use Postman as the client application for testing, in the `IntegrationDevelopmentGuide.postman_collection.json` collection, make sure that the collection's variables have the proper values. This collection is located in the `IntegrationDevelopment\Help` folder of the *Help-and-Training-Examples* repository on GitHub.
4. On the *Enable/Disable Features* (CS100000) form, make sure that the *Inventory and Order Management*, *Lot and Serial Tracking*, and *Advanced SO Invoices* features are enabled.

In the request examples below, `<Acumatica ERP instance URL>` is the URL of the Acumatica ERP instance (such as `https://my.acumatica.com/MyInstance`). You can omit the instance name in the URL (such as `https://my.acumatica.com`) if the instance is installed in the root of the website.

Step 1: Create a Sales Order for the Initially Sold Item

You use the following example of an HTTP request to create a sales order to sell an *APJAM32* item (a box of medium-sized jars of apple jam) to the *FRUITICO* customer.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "FRUITICO" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APJAM32" },
            "OrderQty": { "value": 1 },
            "UOM": { "value": "BOX" },
            "WarehouseID": { "value": "WHOLESALE" }
        }
    ]
}
```

Step 2: Create a Shipment for the First Sales Order

You use the following example of an HTTP request to create a shipment for the sales order. To identify the sales order, you use the values of the key fields (`OrderNbr` and `OrderType`) that are received in the HTTP response of the previous step.

```
POST /SalesOrderCreateShipment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "OrderType": {"value": "SO"},
    "OrderNbr": {"value": "000074"}
```

```
  },
  "parameters": {
      "ShipmentDate": { "value": "2023-02-16T00:00:00+03:00" },
      "WarehouseID": { "value": "WHOLESALE" }
  }
}
```

Step 3: Retrieve the Sales Order Shipment Number

You use the following example of an HTTP request to learn the number of the shipment that you created for the sales order in the previous step.

```
GET /SO/000074?$expand=Shipments&$select=OrderNbr,Shipments/ShipmentNbr HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json
```

Step 4: Confirm the Shipment

You use the following example of an HTTP request to confirm the shipment that you created in Step 2.

```
POST /ConfirmShipment HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "ShipmentNbr": {"value": "000072"},
    "ShipmentType":  {"value": "Shipment"}
  },
  "parameters": { }
}
```

Step 5: Prepare a Sales Invoice for the Shipment

You use the following example of an HTTP request to prepare a sales invoice for the shipment that you created in Step 2.

```
POST /PrepareInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json

{
  "entity":{
    "ShipmentNbr": {"value": "000072"},
    "ShipmentType":  {"value": "Shipment"}
  },
  "parameters": { }
}
```

Step 6: Retrieve the Sales Invoice Number

You use the following example of an HTTP request to learn the number of the sales invoice that you prepared for the shipment in Step 5.

```
GET ?$expand=Orders&$select=ShipmentNbr,Orders/InvoiceNbr&
    $filter=ShipmentNbr eq '000072' HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/Shipment
Accept: application/json
Content-Type: application/json
```

Step 7: Release the Sales Invoice

You use the following example of an HTTP request to release the sales invoice that you prepared in Step 5.

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000137"},
      "Type": {"value": "Invoice"}
      },
  "parameters" : {}
}
```

Step 8: Create an Order of the RM Type

You use the following example of an HTTP request to create an order of the *RM* type on the *Sales Orders* (3010000) form that is intended to return the *APJAM32* item that the *FRUITICO* customer bought earlier. The order also includes the *APJAM08* item, which is being sold to the customer.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "CustomerID": { "value": "FRUITICO" },
    "OrderType": { "value": "RM" },
    "Details": [
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APJAM08" },
            "OrderQty": { "value": 1 },
            "UOM": { "value": "PIECE" },
            "WarehouseID": { "value": "WHOLESALE" },
            "AutoCreateIssue": { "value": false }
        },
        {
            "Branch": { "value": "HEADOFFICE" },
            "InventoryID": { "value": "APJAM32" },
```

```
            "OrderQty": { "value": -1 },
            "UOM": { "value": "BOX" },
            "WarehouseID": { "value": "WHOLESALE" },
            "AutoCreateIssue": { "value": false }
        }
    ]
}
```

Step 9: Correct the Order of the RM Type

You use the following example of an HTTP request to set the value of the `AutoCreateIssue` field of the `SalesOrder` object to *false* to prevent the creation of an additional detail line in the sales order of the *RM* type. You use the `id` fields to identify both the sales order and the detail line you amend. You received the values of the `id` fields in the HTTP response in the previous step.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesOrder
Accept: application/json
Content-Type: application/json

{
    "id": "4cd96bb9-41ae-ed11-9e86-9828a61840c3",
    "Details": [
        {
            "id": "53d96bb9-41ae-ed11-9e86-9828a61840c3",
            "AutoCreateIssue": {"value": false}
        }
    ]
}
```

Step 10: Create a Credit Memo

You use the following example of an HTTP request to create a credit memo. In the credit memo, the item in line *1* of the *000137* invoice is being returned, the item in line *1* of the *000139* sales order of the *RM* type is being sold, and an *APJAM96* item is also being returned.

```
PUT ?$expand=Details HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
    "CustomerID": {"value": "FRUITICO"},
    "Type": {"value": "Credit Memo"},
    "Hold": {"value": false},
    "Details":
    [
        {
            "OrigInvNbr": { "value": "000137" },
            "OrigInvType": { "value": "Invoice" },
            "OrigInvLineNbr": { "value": 1 },
            "Qty": { "value": 1 }
        },
        {
            "OrderNbr": { "value": "000139" },
            "OrderType": { "value": "RM" },
```

```
            "OrderLineNbr": { "value": 1 },
            "Qty": { "value": -1 }
        },
        {
            "Branch": {"value": "HEADOFFICE"},
            "InventoryID": {"value": "APJAM96"},
            "WarehouseID": {"value": "WHOLESALE"},
            "Location": {"value": "MAIN"},
            "Qty": {"value": 1},
            "UOM": {"value": "PIECE"}
        }
    ]
}
```

Step 11: Release the Credit Memo

You use the following example of an HTTP request to release the credit memo that you created in the previous step (whose reference number is assumed to be *000140*).

```
POST /ReleaseSalesInvoice HTTP/1.1
Host: [<Acumatica ERP instance URL>]/entity/Default/24.200.001/SalesInvoice
Accept: application/json
Content-Type: application/json

{
  "entity" : {
      "ReferenceNbr": {"value": "000140"},
      "Type": {"value": "Credit Memo"}
      },
  "parameters" : {}
}
```

Authorizing Client Applications to Work with Acumatica ERP

Acumatica ERP supports the OAuth 2.0 mechanism of authorization and OpenID Connect (OIDC) authentication protocol for applications that are integrated with Acumatica ERP through web services application programming interfaces (APIs) or OData. When a client application of Acumatica ERP uses OAuth 2.0 or OIDC, the client application does not operate with the Acumatica ERP credentials to sign a user in to Acumatica ERP; instead, the application obtains an access token from Acumatica ERP and uses this token when it requests data from Acumatica ERP.

Depending on the flow that the client application implements, the client application either has no information on the credentials of an Acumatica ERP user or uses this information only once to obtain the access token. OAuth 2.0 or OIDC improves the security of the Acumatica ERP data accessed by the application and simplifies the management of access rights.

The client application that implements OAuth 2.0 or OIDC can use one of the authorization flows supported by Acumatica ERP, which are the following:

- Authorization Code (OAuth 2.0 and OIDC)
- Implicit (OAuth 2.0 and OIDC)
- Resource Owner Password Credentials (OAuth 2.0)
- Hybrid (OIDC)

In this part, you can find details on the authorization flows and information about how to register the OAuth 2.0 or OIDC client applications and revoke access of the applications.

Getting Started with OAuth 2.0 and OpenID Connect Authorization

OAuth 2.0 and OpenID Connect (OIDC) are the protocols that can be used for authentication and authorization in Acumatica ERP.

OAuth 2.0 enables third-party applications to obtain limited access to Acumatica ERP web services on behalf of a resource owner. It can be used for enabling secure access to the web services without sharing user credentials. OAuth 2.0 uses access tokens to grant access to resources.

OIDC extends OAuth 2.0 by adding an identity layer on top of the authorization process. It allows client applications to verify the identity of users based on the authentication performed by an authorization server, as well as to obtain basic profile information about the authenticated user. OIDC introduces the concept of an ID token, which is a JSON Web Token (JWT) that contains identity information about the user.

In this chapter, you can find overview information about support of OAuth 2.0 and OIDC in Acumatica ERP and learn about general steps you need to perform to implement OAuth 2.0 or OIDC in your application. The chapter also contains details about the implementation of the common steps for each authorization flow.

OAuth 2.0 and OIDC: General Information

OAuth 2.0 and OpenID Connect (OIDC) are used in scenarios where secure authentication and authorization are required. The implementation of these mechanisms includes multiple steps that you need to do in the client application and in Acumatica ERP.

Learning Objectives

In this chapter, you will learn the following:

- Which steps you need to perform to implement OAuth 2.0 or OIDC
- What the differences between the flows are
- How to work with data in Acumatica ERP after successful authorization
- How to refresh access to Acumatica ERP after access has expired

Applicable Scenarios

You implement OAuth 2.0 or OIDC authorization of a client application in the following scenarios:

- You need to provide secure access to Acumatica ERP through the REST API, SOAP API, or OData without sharing user credentials.
- You need to implement single sign-on solutions where users can sign in once and access multiple applications without having to sign in separately to each one.
- Only for OIDC: In the client application, you need to verify the identity of users and obtain basic profile information from Acumatica ERP.

Authorization Implementation

To use OAuth 2.0 or OIDC, you need to perform the following general steps:

1. You register the client application in Acumatica ERP.
2. You implement the authorization flow in the client application.
3. Optional: You implement the refreshing of the application access in the client application.
4. You include the information about the connected application in the customization project.

According to the OAuth 2.0 and OIDC specifications, a secure connection between a client application and the Acumatica ERP website with a Secure Sockets Layer (SSL) certificate is required. Therefore, you have to set up the Acumatica ERP website for HTTPS before the client application can work with data in Acumatica ERP. For more information, see *Preparation for the Acumatica ERP Installation: System Environment*.

Registration of the Application

Before an OAuth 2.0 or OIDC client application can work with Acumatica ERP, you must register this application in Acumatica ERP. For details about registration, see *Registration of an OAuth 2.0 or OIDC Application: General Information*.

Implementation of the Authorization Flow in the Client Application

An authorization flow is a sequence of steps that the client application and Acumatica ERP follow during the authorization process. The client application that implements OAuth 2.0 or OIDC can use one of the authorization flows supported by Acumatica ERP, which are the following:

- Authorization Code (OAuth 2.0 and OIDC), which is described further in *Authorization Code Flow: General Information*
- Implicit (OAuth 2.0 and OIDC); for more information, see *Implicit Flow: General Information*

- Resource Owner Password Credentials (OAuth 2.0), as described in *Resource Owner Password Credentials Flow: General Information*
- Hybrid (OIDC), which is explored more fully in *Hybrid Flow: General Information*

Each authorization flow has its own use cases and security considerations, as you can see in *OAuth 2.0 and OIDC: Comparison of the Flows*. The choice of the flow depends on multiple factors, such as the type of client application, the level of trust between the client and the authorization server, and the security requirements of the application.

Refreshing of the Application Access

The access token, which the client application obtains from Acumatica ERP during authorization of the application, is valid for a specific period of time, which is specified in the response that returns the access token. When the access token expires, the client application can request a new access token by providing the refresh token to the token endpoint. For details about refreshing the application access, see *OAuth 2.0 and OIDC: Refreshing of an Access Token*.

Inclusion of a Connected Application in a Customization Project

If you need to use a client application that implements the OAuth 2.0 or OpenID Connect authorization mechanism with other Acumatica ERP instances, you need to include the information about this client application in a customization project and publish this customization project to these instances. To include the information about the registered client application in a customization project, you use the *Connected Applications* page of the Customization Project Editor.

Revocation of the Application Access

To revoke the access of an OAuth 2.0 or OpenID Connect client application, you can use either of the following Acumatica ERP forms:

- *Connected Applications* (SM303010): On this form, you can revoke the access of any application registered in the current company. You revoke all access granted to the application.
- *User Profile* (SM203010): On this form, you can revoke the access of any application to which you (that is, the user account to which you are signed in) have granted access. Any access granted to this application by other users remains unchanged.

After you have revoked access, the related access tokens are removed from the Acumatica ERP database, and these tokens cannot be used to access data in Acumatica ERP. However, the client secrets remain valid until their expiration dates (if applicable), and the application can use these secrets to request a new access token.

OAuth 2.0 and OIDC: Comparison of the Flows

The table below summarizes the characteristics of the authorization flows supported by Acumatica ERP.

Characteristic	Authorization Code flow	Implicit flow	Resource Owner Password Credentials flow	Hybrid flow
The OAuth 2.0 authorization mechanism is available.	Yes	Yes	Yes	No
OpenID Connect (OIDC) is available.	Yes	Yes	No	Yes

Characteristic	Authorization Code flow	Implicit flow	Resource Owner Password Credentials flow	Hybrid flow
The access token is returned from the authorization endpoint.	No	Yes	No	Yes
The access token is returned from the token endpoint.	Yes	No	Yes	Yes
The refresh token can be issued.	Yes	No	Yes	Yes
The client application has access to Acumatica ERP credentials (username and password).	No	No	Yes	No
The user explicitly grants access to the requested scopes.	Yes	Yes	No	Yes
The client application is authenticated in Acumatica ERP (that is, the client application provides the client ID and client secret or the client ID and JWT bearer token).	Yes	No	Yes	Yes

Related Links

- *Authorization Code Flow: General Information*
- *Implicit Flow: General Information*
- *Resource Owner Password Credentials Flow: General Information*
- *Hybrid Flow: General Information*

OAuth 2.0 and OIDC: Working with Data in Acumatica ERP

To obtain the data from Acumatica ERP or submit data to the system, the client application connects to the web services API or OData endpoint of Acumatica ERP with the needed HTTP method. The following sections provide details on the request.

HTTP Method and URL

A client application can use the REST API, screen-based SOAP API, or OData if a user has granted access to them for the client application. For details on the methods and URLs that can be used to retrieve or submit data, see one of the following topics:

- *Configuring the REST API*
- *Accessing the Exposed Inquiry Results Through OData*
- *Accessing DACs Through OData*
- *Working with the SOAP API*

HTTP Header

When you are working with Acumatica ERP data, you use the following HTTP header.

Key	Value
`Authorization`	The token type, which is *Bearer*, and the access token that the client application has received from the authorization or token endpoint. The client application should include the access token in the `Authorization` header of each request to Acumatica ERP.

Example

The following example retrieves a sales order from the *Default/24.200.001* endpoint through the REST API. The access token is used for authorization.

```
GET /AcumaticaDB/entity/Default/24.200.001/SalesOrder/SO/000001 HTTP/1.1
Host: localhost
Authorization: Bearer cde78a99a2dc6388eb8c7242a90cf9bc
```

OAuth 2.0 and OIDC: Refreshing of an Access Token

An access token is valid for a specific period of time, which is specified in the response that returns the access token. When the access token expires, the client application can request a new access token by providing the refresh token to the token endpoint. To request a new access token, the client application should use the `POST` method. The following sections provide details on the request and the response.

HTTP Method and URL

To request a new access token, the client application should use the `POST` HTTP method. The client application can use one of the following approaches for the URL:

- If the client application supports OpenID Connect Discovery, the client application can use the discovery endpoint address, as shown in the following code.

  ```
  https://<Acumatica ERP instance URL>/identity/.well-known/openid-configuration
  ```

  

  We recommend that the client application use the discovery endpoint address, which eliminates the need to change the application if the target endpoint address changes.

- The client application can directly use the token endpoint address, as shown in the following code.

  ```
  https://<Acumatica ERP instance URL>/identity/connect/token
  ```

HTTP Header

To refresh an access token, you use the following HTTP header.

Key	Value
`Content-Type`	`application/x-www-form-urlencoded`

Request Body

To refresh an access token, you specify the following parameters in the request body.

Parameter	Description
`grant_type`	The type of the request, which must be set to `refresh_token` for the request of the refresh token.
`client_id`	The client ID that was assigned to the client application during the registration of the application in Acumatica ERP. The client ID must have the format in which the ID was generated during the registration of the application. That is, the client ID must include an auto-generated string and the ID of the tenant, such as *88358B02-A48D-A50E-F710-39C1636C30F6@MyTenant*. The client application will have access to the data of the tenant specified in the client ID.
`client_secret`	For a client application that uses a shared secret, the value of the secret that was created for the client application during the registration of the application in Acumatica ERP.
`client_assertion_type`	For a client application that uses JSON Web Token (JWT) bearer tokens, the client assertion type, which must be set to *urn:ietf:params:oauth:client-assertion-type:jwt-bearer*.
`client_assertion`	For a client application that uses JSON Web Token (JWT) bearer tokens, a single JWT.
`refresh_token`	The refresh token that the client application received from the token endpoint along with the access token if a user granted the `offline_access` scope to the client application.

Response Body

Acumatica ERP verifies the provided application credentials and issues the new access token and the new refresh token. To request the access token once again, the client application should use the latest issued refresh token. That is, if the client application has received a new refresh token, the client application should discard the previous refresh token and use the new one.

A successful response includes the following parameters in the response body.

Parameter	Description
`token_type`	The type of the access token, which is *Bearer*.
`access_token`	The new access token.
`expires_in`	The period of time (in seconds) during which the access token is valid.
`scope`	The scope for which the access token and ID token are provided. The returning of this parameter is optional.

Parameter	Description
`refresh_token`	The new refresh token.
`id_token`	The ID token associated with the authenticated session. The ID token contains three parts, which are separated by periods. The parts are Base64 encoded. The second part contains the claims to which the user granted access. For details on the ID token structure, see *https://openid.net/specs/openid-connect-core-1_0.html#IDToken* and *https://www.rfc-editor.org/rfc/rfc7519.html*. We recommend that you use the existing standard libraries for parsing the tokens. The parameter is returned only if the `openid` scope was granted.

OAuth 2.0 and OIDC: Obtaining of the User Data

To obtain the user data, the client application can connect to the user information endpoint of Acumatica ERP with the `GET` HTTP method. See details on the request and the response in the following sections.

The way of obtaining user data that is described in this topic is optional. The recommended way is to parse the validated ID token, which contains the same claims as the ones that are obtained through the request described in this topic. The recommended way does not require an additional call to Acumatica ERP.

HTTP Method and URL

The client application connects to the user information endpoint of Acumatica ERP with the `GET` HTTP method. The client application can use one of the following options for the URL:

- If the client application supports OpenID Connect Discovery, the client application can use the discovery endpoint address, as shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/.well-known/openid-configuration
```

We recommend that the client application use the discovery endpoint address, which eliminates the need to change the application if the target endpoint address changes.

- The client application can directly use the address of the user information endpoint, which is shown below.

```
https://<Acumatica ERP instance URL>/identity/connect/userinfo
```

HTTP Header

To obtain the user data, you use the following HTTP header.

Key	Value
`Authorization`	The token type, which is *Bearer*, and the access token that the client application has received from the authorization or token endpoint. The client application should include the access token in the `Authorization` header of each request to Acumatica ERP. ⚠ For the application to obtain the user data, the access token must include the `openid` scope.

Response Body

The response body includes the claims to which the user has provided access in JSON format.

Example

An example of a request to the user information endpoint is shown below.

```
GET /AcumaticaDB/identity/connect/userinfo HTTP/1.1
Host: localhost
Authorization: Bearer cde78a99a2dc6388eb8c7242a90cf9bc
```

Acumatica ERP verifies the provided access token and returns the following data in the response body.

```
{
    "name": "Kimberly Gibbs",
    "given_name": "Kimberly",
    "family_name": "Gibbs",
    "preferred_username": "gibbs",
    "email": "gibbs@sweetlife.com",
    "zoneinfo": "",
    "updated_at": "1/1/1900 12:00:00 AM",
    "sub": "gibbs@U100"
}
```

Registering Client Applications That Support OAuth 2.0 or OIDC

Before a client application that implements OAuth 2.0 or OpenID Connect (OIDC) can work with Acumatica ERP, you need to register the application in the system. In this chapter, you will learn how to register a client application in Acumatica ERP and which options are available during registration.

Registration of an OAuth 2.0 or OIDC Application: General Information

You use the *Connected Applications* (SM303010) form to register an OAuth 2.0 or OpenID Connect (OIDC) client application.

To register a client application in Acumatica ERP, you need to know the authorization flow that this application implements. For more information on the flows, see *Authorization Code Flow: General Information*, *Implicit Flow:*

General Information, *Resource Owner Password Credentials Flow: General Information*, and *Hybrid Flow: General Information*.

Learning Objectives

In this chapter, you will learn how to register an OAuth 2.0 or OIDC client application in Acumatica ERP.

Applicable Scenarios

You are a developer who is implementing an OAuth 2.0 or OIDC client application. Before this application can work with an Acumatica ERP instance, you need to register the application in this instance.

Registration of a Client Application

You register an OAuth 2.0 or OIDC client application in the Acumatica ERP instance so that the application can work with the instance.

- According to the OAuth 2.0 and OIDC specifications, a secure connection between a client application and the Acumatica ERP website with a Secure Sockets Layer (SSL) certificate is required. Therefore, you have to set up the Acumatica ERP website for HTTPS before the client application can work with data in Acumatica ERP. For more information, see *Preparation for the Acumatica ERP Installation: System Environment*.
- When you are registering the client application, you have to be signed in to the tenant whose data the client application needs to access.

To register a client application, you perform the following general steps on the *Connected Applications* (SM303010) form:

1. In the **Client Name** box of the Summary area, you type the name of the client application.

   You should leave the **Client ID** box blank. The system will fill it in when you save your settings on the form.

2. In the **Flow** box, you select the authorization flow.
3. Depending on the flow you have selected, you specify the relevant settings, which are listed in the following table. (+ indicates that the setting is available for the flow; – indicates that the setting is unavailable for the flow.)

   Table: Availability of Settings for Each Flow

Settings	*Authorization Code* Flow	*Implicit* Flow	*Resource Owner Password Credentials* Flow	*Hybrid* Flow
Mode of refresh token expiration (in the **Mode** box of the **Refresh Tokens** section of the Summary area)	+	–	+	+
Shared secret (which you add by clicking **Add Shared Secret** on the toolbar of the **Secrets** tab)	+	–	+	+

Settings	*Authorization Code* Flow	*Implicit* Flow	*Resource Owner Password Credentials* Flow	*Hybrid* Flow
JSON Web Key (which you add by clicking **Add JSON Web Key** on the toolbar of the **Secrets** tab)	+	–	+	+
JSON Web Key Set URL (which you add by clicking **Add JSON Web Key Set URL** on the toolbar of the **Secrets** tab)	+	–	+	+
The redirect URI (on the **Redirect URIs** tab)	+	+	–	+
For an OIDC application, the claims that will be included in the ID token (by selecting or clearing the **Active** check boxes on the **Claims** tab)	+	+	–	+
For an OIDC application, the plug-in that contains custom claims (in the **Plug-In** box of the Summary area)	+	+	–	+

After the registration, you have the client ID of the client application and, if you have selected a shared secret, the secret value.

Registration of an OAuth 2.0 or OIDC Application: Sliding Expiration of Refresh Tokens

If you do not want a user to reauthorize the client application to work with Acumatica ERP every 30 days, you can configure the sliding expiration of refresh tokens for client applications. On the *Connected Applications* (SM303010) form, for any client application that has the *Authorization Code*, *Resource Owner Password Credentials*, or *Hybrid* flow, you can select the *Sliding Expiration* mode in the **Refresh Tokens** section in the Summary area. You can also specify the length of the sliding lifetime and indicate whether the refresh tokens for the application have an absolute lifetime.

How the Sliding Expiration Works

When a user grants the `offline_access` scope (along with the `api` or `openid` scope) to a connected application, the application receives a refresh token and an access token. The application can then access data in Acumatica ERP during a specific period of time, which is specified in the response that returns the access token. When the access token expires, the client application can request a new access token by providing the refresh token to the token endpoint. The refresh token can be provided anytime within 30 days of the first issuing of the token.

If during these 30 days, the connected application provides the refresh token to the token endpoint, the system extends the period of time for which the new refresh token is valid. The lifetime is extended by the time that is specified in the **Sliding Lifetime (Days)** box in the Summary area (**Refresh Tokens** section) of the *Connected Applications* (SM303010) form. The lifetime of the refresh token can be extended multiple times by the period of

the sliding lifetime until the refresh token's total lifetime (from its initial issuing) exceeds the number of days that is specified in the **Absolute Lifetime (Days)** box. If the **Infinite** check box is selected for the absolute lifetime, the lifetime of the refresh token can be extended endlessly. The following diagram illustrates the sliding expiration of refresh tokens.

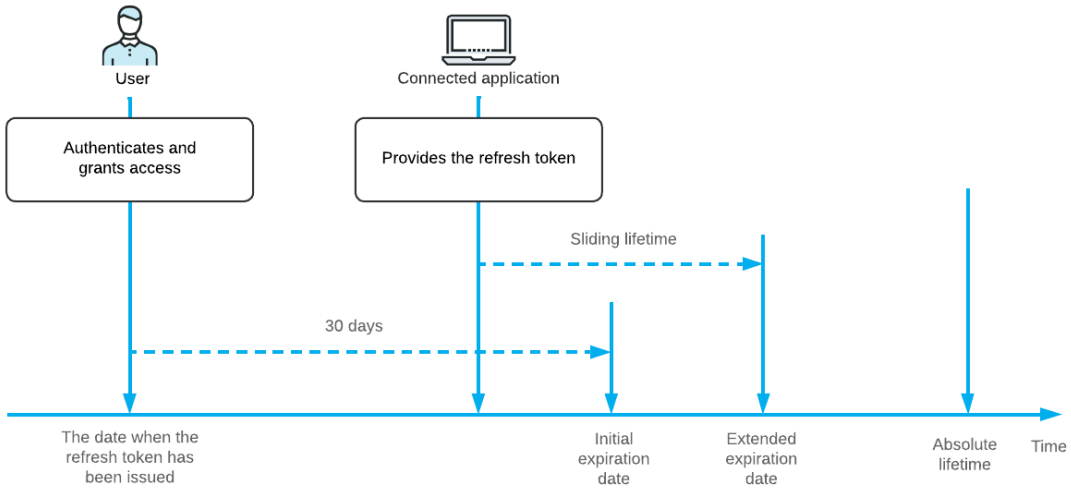

Figure: Lifetime of refresh tokens with sliding expiration

Registration of an OAuth 2.0 or OIDC Application: JWT Bearer Tokens

Acumatica ERP implements support for JSON Web Token (JWT) bearer tokens for client authentication. With this support, the private secret key is stored only in the client application, while the Acumatica ERP instance holds the public key.

Registration of the Application

When you register the application on the *Connected Applications* (SM303010) form, you add either a JSON Web Key (JWK) or a JSON Web Key Set(JWKS) URL on the **Secrets** tab.

To add a JWK, you click the new **Add JSON Web Key** button on the table toolbar and specify the needed settings in the dialog box that opens. For JWK, Acumatica ERP supports the format that is defined in RFC7517 (*https://datatracker.ietf.org/doc/html/rfc7517#section-4*).

To add a JWKS URL, you click the new **Add JSON Web Key Set URL** button on the table toolbar and specify the needed settings in the dialog box that opens. The JWKS URL should point to a location that satisfies the following requirements:

- It is accessible from each Acumatica ERP instance that is used with the client application. If the location is inaccessible, the token request is declined with the `invalid_client` error.
- It complies with RFC7515 (*https://datatracker.ietf.org/doc/html/rfc7517#section-5*).
- It should support a reasonable load because each Acumatica ERP instance that is used with the client application will access this location on every token request.

Registration of an OAuth 2.0 or OIDC Application: Acumatica ERP as an Identity Provider via OIDC

Acumatica ERP can be used as an identity provider via the OpenID Connect (OIDC) protocol. An OIDC client application uses the Acumatica ERP sign-in page for authentication. During the first sign-in, the client application requests access to the user attributes; for the application to be signed in, a user must confirm the granting of access to these attributes.

Registration of the Application

On the *Connected Applications* (SM303010) form, support for OIDC is available for the Authorization Code, Hybrid, and Implicit flows.

On the **Claims** tab of the *Connected Applications* form, the check box should be selected in the **Active** column for the claims that will be included in the token in the response to the client application (when OIDC is used). By default, Acumatica ERP contains a set of claims and a set of scopes; each scope defines the claims that will be included in a response when the scope is specified in a request. These sets of scopes and claims can be redefined in a plug-in included in a customization project. If a plug-in is selected in the Summary area of the *Connected Applications* form, the claims that are defined in this plug-in are added to the table on the **Claims** tab; in this table, these claims are marked as belonging to the plug-in.

Implementing the Authorization Code Flow

The Authorization Code flow is a secure method for authorization used in OAuth 2.0 and OpenID Connect (OIDC). This flow is typically used in scenarios where the client application needs to access resources on behalf of the user, but the client application itself is not trusted with the user's credentials. In this chapter, you can find details about the implementation of the Authorization Code flow.

Authorization Code Flow: General Information

When you implement OAuth 2.0 or OpenID Connect (OIDC) in a client application to make the application work with Acumatica ERP, you can use the Authorization Code flow. With this authorization flow, the client application never gets the credentials of the applicable Acumatica ERP user. After the user is authenticated in Acumatica ERP, the client application receives an authorization code, exchanges it for an access token, and then uses the access token to work with data in Acumatica ERP.

Learning Objectives

In this chapter, you will learn how to implement a client application that uses the Authorization Code flow.

Applicable Scenarios

You implement the Authorization Code flow in a client application when you want to securely obtain an access token without exposing the user's credentials to the client application.

Authorization Code Flow

For the support of the Authorization Code flow, you implement the following general steps in the application:

1. **Obtaining an authorization code**

   If the client application uses the proof key for code exchange (PKCE), the client application generates the code verifier and the code challenge. For details about this generation, see *https://www.rfc-editor.org/rfc/rfc7636*. Only the S256 challenge method is supported.

   The client application connects to the authorization endpoint of Acumatica ERP; if PKCE is used, it provides the code challenge.

   The authorization endpoint directs the user of the client application to the sign-in page of Acumatica ERP, where the user should enter the credentials to sign in to a tenant configured in the Acumatica ERP instance.

   

   The user must sign in to the tenant that was specified in the `client_id` URL parameter passed to the authorization endpoint. (This tenant is selected by default on the sign-in page.)

   If the credentials are accepted by Acumatica ERP, the system displays the consent form, where the user can confirm that the application has access to the requested scopes. Only the scopes that were requested by the application are displayed on the consent form.

   Once the user grants access to the requested scopes, Acumatica ERP redirects the client application to the address that was specified in the request, and adds the authorization code in the URL parameter.

   For details on the request for the authorization code, see *Authorization Code Flow: Obtaining of an Authorization Code*.

2. **Obtaining an access token and ID token**

   If the client application uses JSON Web Token (JWT) bearer tokens, the application generates a JWT and signs it with the private key.

   The client application connects to the token endpoint of Acumatica ERP and submits the following:

   - The authorization code.
   - A signed JWT or a shared secret. If a JWT is provided, Acumatica ERP verifies the JWT signature by using the public key (which was specified during the registration of the client application in Acumatica ERP) and validates the JWT payload. If a shared secret is provided, Acumatica ERP verifies the provided application credentials.
   - If PKCE is used, the code verifier. Acumatica ERP validates the code verifier upon the code challenge that the system has received from the client application with the request for the authorization code.

   If verification is completed successfully, Acumatica ERP issues the access token, the ID token, and the refresh token if these tokens have been requested by the application. The client application should provide the access token with each data request to Acumatica ERP.

   If the ID token is retrieved, the client application validates it by using the key that is available on the *OpenID Connect Preferences* (SM303030) form. The client application can obtain the key through a `GET` request to the following URL: *[<Acumatica ERP instance URL>]/identity/.well-known/openid-configuration/jwks*. The ID token contains the claims to which the user has granted access.

   For more information on this process, see *Authorization Code Flow: Obtaining of an Access Token and ID Token*.

3. **Optional: Retrieving the user information**

   The client application requests user information from Acumatica ERP and provides the access token with this request. Acumatica ERP returns the information for which the user has provided the consent. For details about this request, see *OAuth 2.0 and OIDC: Obtaining of the User Data*.

   

   The recommended way of obtaining the user data is to parse the validated ID token, which contains the same claims as the ones that are obtained through this request.

4. **Optional: Working with data in Acumatica ERP**

The client application requests data from Acumatica ERP and provides the access token with this request. Acumatica ERP returns the requested data. For details on this process, see *OAuth 2.0 and OIDC: Working with Data in Acumatica ERP*.

When the access token expires, the client application can request a new access token by providing a refresh token, as described in *OAuth 2.0 and OIDC: Refreshing of an Access Token*.

For details on the OAuth 2.0 authorization mechanism, see the specification at *https://tools.ietf.org/html/rfc6749*. For details on the OIDC authorization mechanism, see the specification at *https://openid.net/specs/openid-connect-core-1_0.html#Authentication*.

The configuration of the OpenID Connect protocol that is used by an Acumatica ERP website can be displayed by a request to the following URL: *[<Acumatica ERP instance URL>]/identity/.well-known/openid-configuration*. (In this request, *[<Acumatica ERP instance URL>]* stands for the URL of the Acumatica ERP website.)

Authorization Code Flow Diagram

The following diagram illustrates the Authorization Code flow.

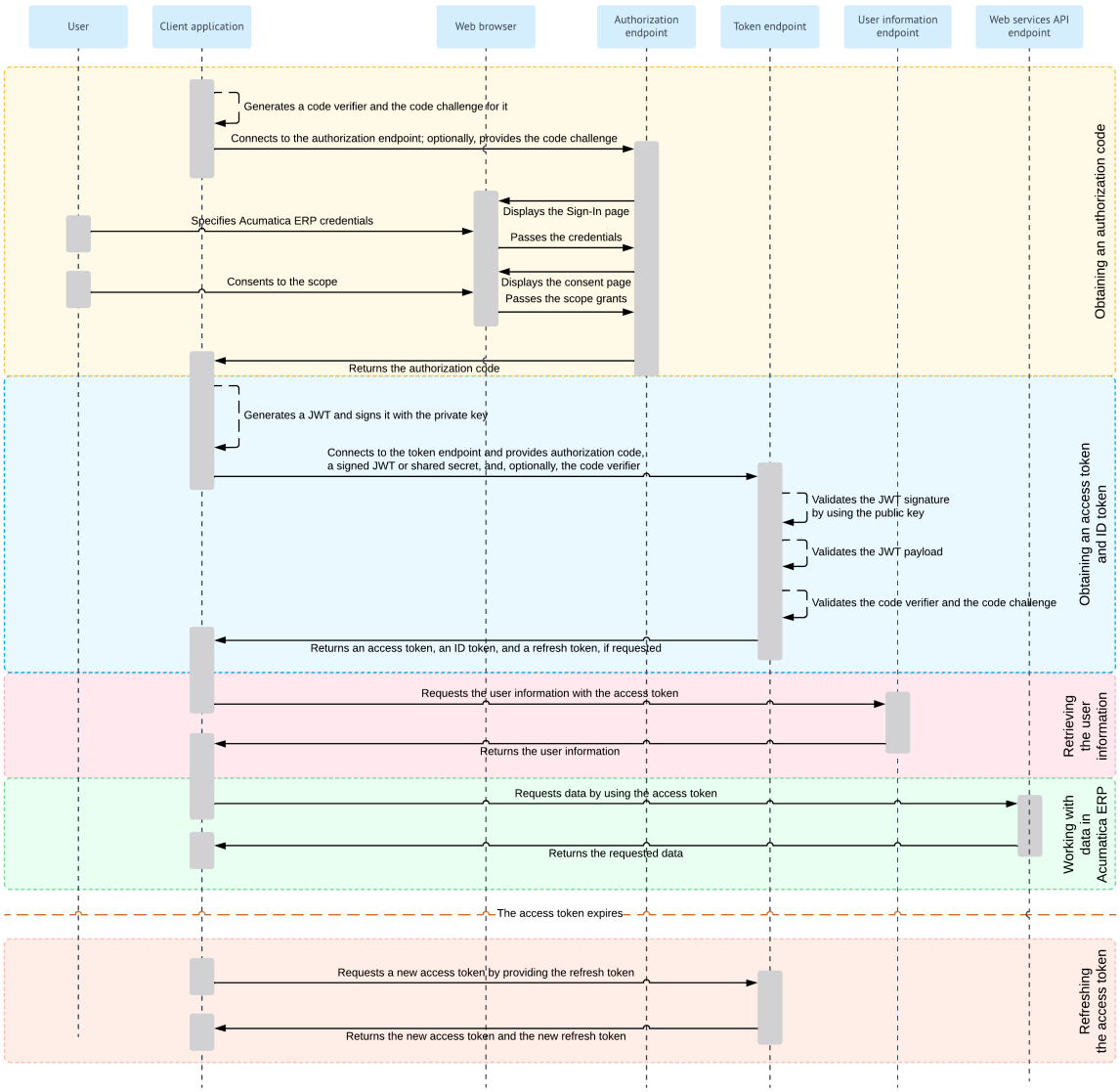

Figure: Authorization Code flow

Authorization Code Flow: Obtaining of an Authorization Code

To obtain an authorization code, the client application connects to the authorization endpoint of Acumatica ERP with the `GET` HTTP method and specifies the parameters of the request in the URL. For details on the request and the response, see the following sections.

HTTP Method and URL

The client application connects to the authorization endpoint of Acumatica ERP with the `GET` method. The client application can use one of the following options for the URL:

- If the client application supports OpenID Connect Discovery, the client application can use the discovery endpoint address, as shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/.well-known/openid-configuration
```

> ⚠ We recommend that the client application use the discovery endpoint address, which eliminates the need to change the application if the target endpoint address changes.

- The client application can directly use the authorization endpoint address, which is shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/connect/authorize
```

Parameters

The client application should specify the following URL parameters.

Parameter	Description
`response_type`	The type of the flow, which must be set to `code` for the Authorization Code flow.
`client_id`	The client ID that was assigned to the client application during the registration of the application in Acumatica ERP. The client ID must have the format in which the ID was generated during the registration of the application. That is, the client ID must include an auto-generated string and the ID of the tenant, such as *88358B02-A48D-A50E-F710-39C1636C30F6@MyTenant*. The client application will have access to the data of the tenant specified in the client ID.
`redirect_uri`	The URI in the client application to which the response to the request should be sent. The URI must exactly match one of the values specified for the application in the **Redirect URI** column on the **Redirect URIs** tab of the *Connected Applications* (SM303010) form.
`scope`	The access scope that is requested by the client application. The scope can be a combination of the following values, delimited by spaces: • `openid`: Requests access to the personal information of the user. If this scope is granted, the OpenID Connect authorization mechanism is used. Without this scope, OAuth 2.0 is used. • `email`: Requests disclosure of the user's email address. • `profile`: Requests disclosure of the user's profile information. • `phone`: Requests disclosure of the user's phone number. • `api`: Requests access to the REST API, screen-based SOAP API, and OData interface. If this scope is granted and the `api:concurrent_access` scope is not granted, Acumatica ERP manages the sessions of the application through tokens. Acumatica ERP issues the first access token along with the session ID. If the client application requests a new access token by presenting a refresh token, Acumatica ERP reuses the session ID that was issued for the first access token issued with the refresh token. That is, the system uses a single session for each access granted to the client application. For details about the license limitations related to the number of sessions for client applications, see *License Restrictions for API Users*. • `offline_access`: Requests that a refresh token be granted. If a user grants this scope to the application, Acumatica ERP issues to the client

Parameter	Description
	application a refresh token along with the access token. When the access token has expired, the client application can request a new access token by sending a request to the token endpoint and providing the refresh token. By default, the whole chain for the refresh token expires 30 days after the initial authentication process. However, you can change these settings in the **Refresh Tokens** section of the Summary area of the *Connected Applications* (SM303010) form. For details, see *Registration of an OAuth 2.0 or OIDC Application: Sliding Expiration of Refresh Tokens*. • `api:concurrent_access`: Requests permission for the concurrent use of multiple types of web service APIs. If a user grants this scope to the application, the client application can access data in Acumatica ERP in concurrent mode. In this case, Acumatica ERP can maintain multiple sessions for the client application, managing session IDs through cookies. We recommend that the client application request this scope only if concurrent access is required for the client application. For details about the license limitations related to the number of sessions for client applications, see *License Restrictions for API Users*.
`code_challenge_method`	For a client application that uses the proof key for code exchange (PKCE), the code challenge method, which must be set to `S256`.
`code_challenge`	For a client application that uses PKCE, the code challenge. The code challenge is the base64url-encoded SHA-256 hash of the code verifier. The code verifier is a cryptographically random string that is used to correlate the authorization request to the token request. For details about the code verifier and the code challenge, see *https://www.rfc-editor.org/rfc/rfc7636*.

Response

Once the user grants access to the requested scopes, Acumatica ERP redirects the client application to the `redirect_uri` address that was specified in the request, and adds the authorization code in the `code` URL parameter.

Example

An example of a request to the authorization endpoint is shown below. (Line breaks are for display purposes only.)

```
GET https://localhost/AcumaticaDB/identity/connect/authorize?
response_type=code
&client_id=58FCCFBD-0CF3-C047-B720-A631C976A8DD@U100
&redirect_uri=http%3A%2F%2Flocalhost%2Fclientapp%2F
&scope=api%20offline_access
```

Once the user grants access to the requested scopes, Acumatica ERP redirects the client application to the following URL: *https://localhost/clientapp/?code=rOBVT0nmPhaXlHeBpE81iJBrflt5r7ud5_2czGYIr14&scope=api%20offline_access*.

Authorization Code Flow: Obtaining of an Access Token and ID Token

To obtain an access token, an ID token, or both, a client application that implements the Authorization Code flow connects to the token endpoint of Acumatica ERP with the `POST` method. For details on the request and the response, see the following sections.

HTTP Method and URL

The client application connects to the token endpoint of Acumatica ERP with the `POST` method. The client application can use one of the following options for the URL:

- If the client application supports OpenID Connect Discovery, the client application can use the discovery endpoint address, as shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/.well-known/openid-configuration
```

We recommend that the client application use the discovery endpoint address, which eliminates the need to change the application if the target endpoint address changes.

- The client application can directly use the token endpoint address, as shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/connect/token
```

HTTP Header

You use the following HTTP header.

Key	Value
`Content-Type`	`application/x-www-form-urlencoded`

Request Body

You specify the following parameters in the request body.

Parameter	Description
`grant_type`	The type of the OAuth 2.0 flow, which must be set to *authorization_code* for the Authorization Code flow.
`client_id`	The client ID that was assigned to the client application during the registration of the application in Acumatica ERP. The client ID must have the format in which the ID was generated during the registration of the application. That is, the client ID must include an auto-generated string and the ID of the tenant, such as *88358B02-A48D-A50E-F710-39C1636C30F6@MyTenant*. The client application will have access to the data of the tenant specified in the client ID.
`code`	The authorization code that the client application has received from the authorization endpoint.

Parameter	Description
`client_secret`	For a client application that uses a shared secret, the value of the secret that was created for the client application during the registration of the application in Acumatica ERP.
`client_assertion_type`	For a client application that uses JSON Web Token (JWT) bearer tokens, the client assertion type, which must be set to *urn:ietf:params:oauth:client-assertion-type:jwt-bearer*.
`client_assertion`	For a client application that uses JSON Web Token (JWT) bearer tokens, a single JWT.
`code_verifier`	For a client application that uses the proof key for code exchange (PKCE), the code verifier for which the client application sent the code challenge during the request for the authorization code. For details about the code verifier and the code challenge, see *https://www.rfc-editor.org/rfc/rfc7636*.
`redirect_uri`	The URI in the client application to which the response to the request should be sent. The URI must exactly match one of the values specified for the application in the **Redirect URI** column on the **Redirect URIs** tab of the *Connected Applications* (SM303010) form.

Response

Acumatica ERP verifies the provided application credentials and issues an access token, an ID token, and a refresh token if they have been requested by the application. The client application should provide the access token with each data request to Acumatica ERP.

A successful response includes the following parameters in the response body.

Parameter	Description
`token_type`	The type of the access token, which is *Bearer*. The parameter is returned only if the `api` scope was granted.
`access_token`	The access token. The parameter is returned only if the `api` scope was granted.
`expires_in`	The period of time (in seconds) during which the access token is valid. The parameter is returned only if the `api` scope was granted.
`scope`	The scope for which the access token and ID token are provided. The returning of this parameter is optional.
`refresh_token`	The refresh token. The parameter is returned only if the `offline_access` scope was granted.
`id_token`	The ID token associated with the authenticated session. The ID token contains three parts, which are separated by periods. The parts are Base64 encoded. The second part contains the claims to which the user granted access. For details on the ID token structure, see *https://openid.net/specs/openid-connect-core-1_0.html#IDToken* and *https://www.rfc-editor.org/rfc/rfc7519.html*. We recommend that you use the existing standard libraries for parsing the tokens. The parameter is returned only if the `openid` scope was granted.

Example

The following example shows a request for an access token with a shared secret provided with the request. (Line breaks are for display purposes only.)

```
POST /identity/connect/token HTTP/1.1
Host: https://localhost/AcumaticaDB
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&code=rOBVT0nmPhaXlHeBpE81iJBrfIt5r7ud5_2czGYIr14
&client_id=58FCCFBD-0CF3-C047-B720-A631C976A8DD@U100
&client_secret=cTUa8QxZnloGoxpT_u3ZBA
&redirect_uri=https%3A%2F%2Flocalhost
```

A successful response has the body shown in the following example.

```
{
    "access_token": "u39uoZj9A4fj2T80Zx0Qirznr0oqNb1qK92c48ZdxUg",
    "expires_in": 3600,
    "token_type": "Bearer",
    "scope": "api offline_access"
}
```

Implementing the Implicit Flow

The Implicit flow is a type of OAuth 2.0 or OpenID Connect (OIDC) flow that is primarily used when the client application (typically a web application running in a browser) is incapable of keeping secrets confidential. In this flow, the access token is returned directly to the client application after authentication, without an intermediate step to exchange authorization code. In this chapter, you can find details about the implementation of the Implicit flow.

Implicit Flow: General Information

When you implement OAuth 2.0 or OpenID Connect (OIDC) in a client application to make the application work with Acumatica ERP, you can use the Implicit flow, which is a simplified variant of the Authorization Code flow.

With the Implicit flow, the client application never gets the credentials of the applicable Acumatica ERP user. When the user is authenticated in Acumatica ERP, the client application does not receive an authorization code (as with the Authorization Code flow); instead, the client application directly receives an access token, and then uses the access token to work with data in Acumatica ERP. The access token is valid for a limited period of time and cannot be renewed.

Learning Objectives

In this chapter, you will learn how to implement a client application that uses the Implicit flow.

Applicable Scenarios

You implement the Implicit flow in a client application when you want to securely obtain an access token without exposing the user's credentials to the client application. This flow can be used for clients using a scripting language (such as JavaScript) or for mobile clients.

Implicit Flow

For the support of the Implicit flow, you implement the following general steps in the application:

1. **Obtaining an access token**

   The client application connects to the authorization endpoint of Acumatica ERP.

   The authorization endpoint directs the user of the client application to the sign-in page of Acumatica ERP, where the user should enter the credentials to sign in to a tenant configured in the Acumatica ERP instance.

   > ⚠ The user must sign in to the tenant that was specified in the `client_id` URL parameter passed to the authorization endpoint. (This tenant is selected by default on the sign-in page.)

   If the credentials are accepted by Acumatica ERP, the system displays the consent form, where the user can confirm that the application has access to the requested scopes. Only the scopes that were requested by the application are displayed on the consent form.

   Once the user grants access to the requested scopes, Acumatica ERP issues the access token and the ID token (if requested). The client application should provide the access token with each data request to Acumatica ERP.

   If the ID token is retrieved, the client application validates it by using the key that is available on the *OpenID Connect Preferences* (SM303030) form. The client application can obtain the key through a `GET` request to the following URL: *[<Acumatica ERP instance URL>]/identity/.well-known/openid-configuration/jwks*. The ID token contains the claims to which the user has granted access.

   For more information on the request that obtains the tokens, see *Implicit Flow: Obtaining of an Access Token and ID Token*.

2. **Optional: Retrieving the user information**

   The client application requests user information from Acumatica ERP and provides the access token with this request. Acumatica ERP returns the information for which the user has provided the consent. For details about this request, see *OAuth 2.0 and OIDC: Obtaining of the User Data*.

   > ⚠ The recommended way of obtaining the user data is to parse the validated ID token, which contains the same claims as the ones that are obtained through this request.

3. **Optional: Working with data in Acumatica ERP**

   The client application requests data from Acumatica ERP and provides the access token with this request. Acumatica ERP returns the requested data. For details on this process, see *OAuth 2.0 and OIDC: Working with Data in Acumatica ERP*.

> ⚠ Refresh tokens are not supported by the Implicit flow.

For details on the OAuth 2.0 authorization mechanism, see the specification at *https://tools.ietf.org/html/rfc6749*. For details on the OIDC authorization mechanism, see the specification at *https://openid.net/specs/openid-connect-core-1_0.html#Authentication*.

> The configuration of the OpenID Connect protocol that is used by an Acumatica ERP website can be displayed by a request to the following URL: *[<Acumatica ERP instance URL>]/identity/.well-known/openid-configuration*. (In this request, *[<Acumatica ERP instance URL>]* stands for the URL of the Acumatica ERP website.)

Implicit Flow Diagram

The following diagram illustrates the Implicit flow.

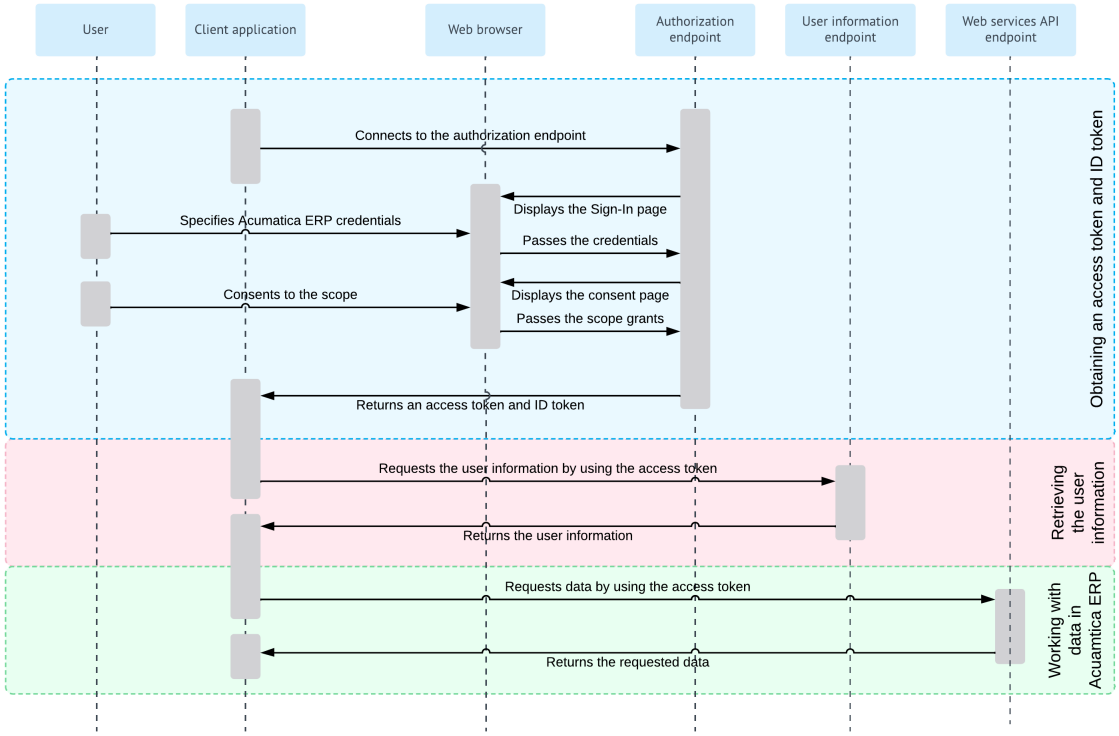

Figure: Implicit flow

Implicit Flow: Obtaining of an Access Token and ID Token

To obtain an access token, an ID token, or both, a client application that implements the Implicit flow connects to the authorization endpoint of Acumatica ERP with the `GET` method. For details on the request and the response, see the following sections.

HTTP Method and URL

The client application connects to the authorization endpoint of Acumatica ERP with the `GET` method. The client application can use one of the following options for the URL:

- If the client application supports OpenID Connect Discovery, the client application can use the discovery endpoint address, as shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/.well-known/openid-configuration
```

> ⚠ We recommend that the client application use the discovery endpoint address, which eliminates the need to change the application if the target endpoint address changes.

- The client application can directly use the authorization endpoint address, which is shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/connect/authorize
```

Parameters

The client application should specify the following URL parameters.

Parameter	Description
`response_type`	The type of the flow. The type can be one of the following: • *token*: Is used for OAuth 2.0 to retrieve an access token. You must include *api* in the `scope` parameter. • *id_token*: Is used for OIDC to retrieve an ID token. No access token is returned. Do not include *api* in the `scope` parameter. • *id_token token*: Is used for OIDC to retrieve an ID token and access token. You must include *api* in the `scope` parameter.
`client_id`	The client ID that was assigned to the client application during the registration of the application in Acumatica ERP. The client ID must have the format in which the ID was generated during the registration of the application. That is, the client ID must include an auto-generated string and the ID of the tenant, such as *88358B02-A48D-A50E-F710-39C1636C30F6@MyTenant*. The client application will have access to the data of the tenant specified in the client ID.
`redirect_uri`	The URI in the client application to which the response to the request should be sent. The URI must exactly match one of the values specified for the application in the **Redirect URI** column on the **Redirect URIs** tab of the *Connected Applications* (SM303010) form.

Parameter	Description
`scope`	The access scope that is requested by the client application. The scope can be a combination of the following values, delimited by spaces: • `openid`: Requests access to the personal information of the user. If this scope is granted, the OpenID Connect authorization mechanism is used. Without this scope, OAuth 2.0 is used. • `email`: Requests disclosure of the user's email address. • `profile`: Requests disclosure of the user's profile information. • `phone`: Requests disclosure of the user's phone number. • `api`: Requests access to the REST API, screen-based SOAP API, and OData interface.  The `api` scope is required if the *token* or *id_token token* response type is specified. If this scope is granted and the `api:concurrent_access` scope is not granted, Acumatica ERP manages the sessions of the application through tokens. The system uses a single session for each access granted to the client application. • `api:concurrent_access`: Requests permission for the concurrent use of multiple types of web service APIs. If a user grants this scope to the application, the client application can access data in Acumatica ERP in concurrent mode. In this case, Acumatica ERP can maintain multiple sessions for the client application, managing session IDs through cookies. We recommend that the client application request this scope only if concurrent access is required for the client application. For details about the license limitations related to the number of sessions for client applications, see *License Restrictions for API Users*.  The `offline_access` scope is not supported for the Implicit flow.
`nonce`	A string value that is used to associate a client session with an ID token, and to mitigate replay attacks. This parameter is required if the *id_token* or *id_token token* response type is specified.

Response

Once the user grants access to the requested scopes, Acumatica ERP redirects the client application to the `redirect_uri` address, which was specified in the request, and adds the requested data in the fragment section of the redirect URL. The redirect URL includes the following fragment parameters.

Parameter	Description
`token_type`	The type of the access token, which is *Bearer*. The parameter is returned only if the `api` scope was granted.
`access_token`	The access token. The parameter is returned only if the `api` scope was granted.
`expires_in`	The period of time (in seconds) during which the access token is valid. The parameter is returned only if the `api` scope was granted.
`scope`	The scope for which the access token and ID token are provided. The returning of this parameter is optional.

Parameter	Description
`id_token`	The ID token associated with the authenticated session. The ID token contains three parts, which are separated by periods. The parts are Base64 encoded. The second part contains the claims to which the user granted access. For details on the ID token structure, see *https://openid.net/specs/openid-connect-core-1_0.html#IDToken* and *https://www.rfc-editor.org/rfc/rfc7519.html*. We recommend that you use the existing standard libraries for parsing the tokens. The parameter is returned only if the `openid` scope was granted.

Example: openid, email, and api Scopes

The following example requests the *openid*, *email*, and *api* scopes. (Line breaks are for display purposes only.)

```
GET https://localhost/AcumaticaDB/identity/connect/authorize?
response_type=id_token%20token
&client_id=2B0C8CF1-FFD4-A0DE-1673-F03084F16240@U100
&redirect_uri=https://localhost
&scope=openid%20email%20api
&nonce=test
```

Once the user grants access to the requested scopes, Acumatica ERP redirects the client application to the following URL.

```
https://localhost/#
id_token=ey...rvE
&access_token=BTGTm5nSGIZWoypYv_QOjD00ziczKaiMEDIVcNf6XpM
&token_type=Bearer
&expires_in=3600
&scope=openid%20email%20api
```

Example: openid and email Scopes

The following example requests the *openid* and *email* scopes. (Line breaks are for display purposes only.)

```
GET https://localhost/AcumaticaDB/identity/connect/authorize?
response_type=id_token
&client_id=8F41DD85-CA55-8518-8464-4C983D64BBA4@U100
&redirect_uri=https://localhost
&scope=openid%20email
&nonce=test
```

Once a user grants access to the requested scope, Acumatica ERP redirects the client application to the following URL.

```
https://localhost/#
id_token=eyJh...3YNA
&scope=openid
```

Example: api Scope

The following example requests the *api* scope. (Line breaks are for display purposes only.)

```
GET http://localhost/AcumaticaDB/identity/connect/authorize?
response_type=token
```

```
&client_id=8F41DD85-CA55-8518-8464-4C983D64BBA4@U100
&redirect_uri=https://localhost
&scope=api
```

Once the user grants access to the requested scope, Acumatica ERP redirects the client application to the following URL.

```
https://localhost/#
access_token=7qgkxo-tJTbouSs8OtU3gdNhW0YQZVA9n6ZQt364nks
&token_type=Bearer
&expires_in=3600
&scope=api
```

Implementing the Resource Owner Password Credentials Flow

The Resource Owner Password Credentials flow in OAuth 2.0 is used when the client application can obtain the user's username and password and directly exchange them for an access token. Unlike other OAuth 2.0 flows—where the client application interacts with the authorization server through redirections, callbacks, and authorization codes—the Resource Owner Password Credentials flow involves sending the user's credentials directly to the authorization server.

In this chapter, you can find details about the implementation of the Resource Owner Password Credentials flow.

Resource Owner Password Credentials Flow: General Information

When you implement OAuth 2.0 in a client application to make the application work with Acumatica ERP, you can use the Resource Owner Password Credentials flow.

With the Resource Owner Password Credentials flow, the credentials (username and password) of the Acumatica ERP user are provided directly to the client application, which uses the credentials to obtain the access token. When the access token expires, the client application can request a new access token by providing a refresh token.

Learning Objectives

In this chapter, you will learn how to implement a client application that uses the Resource Owner Password Credentials flow.

Applicable Scenarios

You can use the Resource Owner Password Credentials flow in environments where the client application can securely store user credentials and there is a high level of trust between the user and the client application, as with native mobile applications.

The Resource Owner Password Credentials flow has significant drawbacks and security concerns, such as the following:

- Handling and transmitting user credentials directly from the client application to the authorization server can introduce security risks.
- Because the Resource Owner Password Credentials flow bypasses the authorization step, a user does not have the opportunity to review and grant consent to the client application's access to the resources.

Therefore, you should carefully consider the use of the Resource Owner Password Credentials flow, and prefer other authorization flows, such as Authorization Code flow, whenever possible.

Resource Owner Password Credentials Flow

For the support of the Resource Owner Password Credentials flow, you implement the following general steps in the application:

1. **Obtaining an access token**

   The client application obtains the username and password of the applicable Acumatica ERP user, which can then be exchanged for an access token.

   If the client application uses JSON Web Token (JWT) bearer tokens, the application generates a JWT and signs it with the private key.

   The client application connects to the token endpoint of Acumatica ERP, submits the user credentials, and provides a signed JWT or a shared secret. If a JWT is provided, Acumatica ERP verifies the JWT signature by using the public key (which was specified during the registration of the client application in Acumatica ERP) and validates the JWT payload. If a shared secret is provided, Acumatica ERP verifies the provided application credentials.

   If verification is completed successfully, Acumatica ERP issues an access token, which the client application should provide with each data request to Acumatica ERP, and a refresh token (if requested).

   For more information on this process, see *Resource Owner Password Credentials Flow: Obtaining of an Access Token*.

2. **Optional: Working with data in Acumatica ERP**

   The client application requests data from Acumatica ERP and provides the access token with this request. Acumatica ERP returns the requested data. For details on this process, see *OAuth 2.0 and OIDC: Working with Data in Acumatica ERP*.

When the access token expires, the client application can request a new access token by providing a refresh token, as described in *OAuth 2.0 and OIDC: Refreshing of an Access Token*.

For details on the OAuth 2.0 authorization mechanism, see the specification at *https://tools.ietf.org/html/rfc6749*.

Diagram of the Resource Owner Password Credentials Flow

The following diagram illustrates the Resource Owner Password Credentials flow.

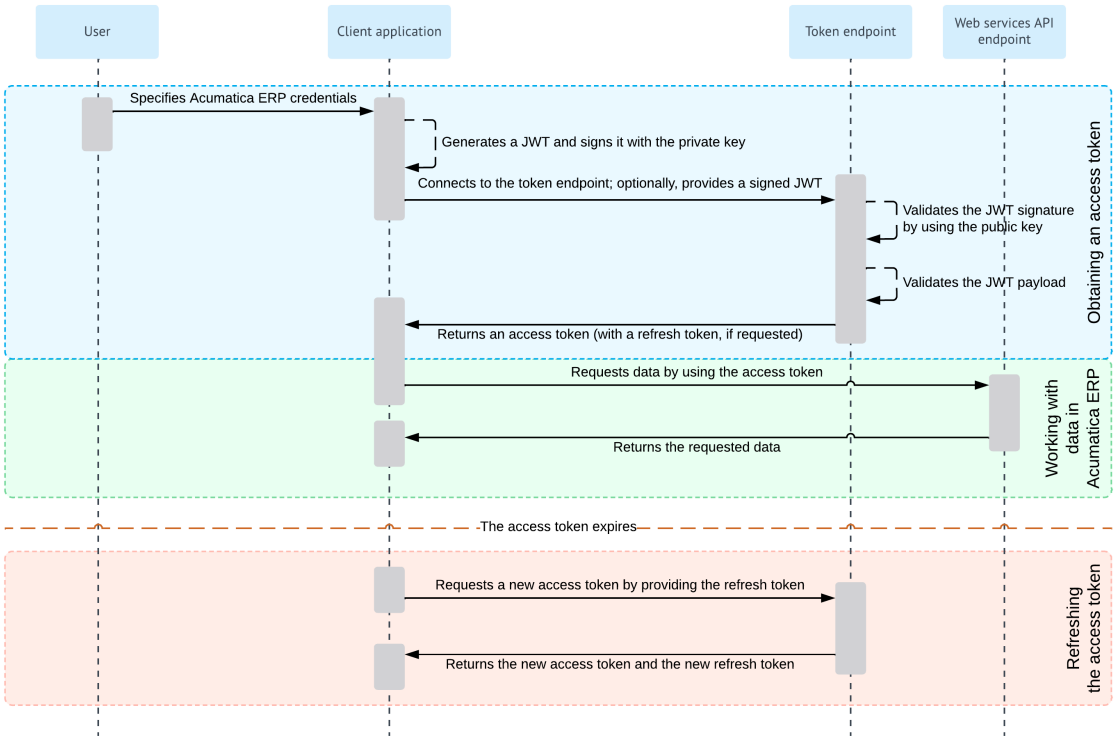

Figure: Resource Owner Password Credentials flow

Resource Owner Password Credentials Flow: Obtaining of an Access Token

To obtain an access token, a client application that implements the Resource Owner Password Credentials flow connects to the token endpoint of Acumatica ERP with the `POST` method. For details on the request and the response, see the following sections.

HTTP Method and URL

The client application connects to the token endpoint of Acumatica ERP with the `POST` method. The client application can use one of the following options for the URL:

- If the client application supports OpenID Connect Discovery, the client application can use the discovery endpoint address, as shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/.well-known/openid-configuration
```

> We recommend that the client application use the discovery endpoint address, which eliminates the need to change the application if the target endpoint address changes.

- The client application can directly use the token endpoint address, as shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/connect/token
```

HTTP Header

You use the following HTTP header.

Key	Value
`Content-Type`	`application/x-www-form-urlencoded`

Request Body

You specify the following parameters in the request body.

Parameter	Description
`grant_type`	The type of the OAuth 2.0 flow, which must be set to `password` for the resource owner password credentials flow.
`client_id`	The client ID that was assigned to the client application during the registration of the application in Acumatica ERP. The client ID must have the format in which the ID was generated during the registration of the application. That is, the client ID must include an auto-generated string and the ID of the tenant, such as *88358B02-A48D-A50E-F710-39C1636C30F6@MyTenant*. The client application will have access to the data of the tenant specified in the client ID.
`client_secret`	For a client application that uses a shared secret, the value of the secret that was created for the client application during the registration of the application in Acumatica ERP.
`client_assertion_type`	For a client application that uses JSON Web Token (JWT) bearer tokens, the client assertion type, which must be set to *urn:ietf:params:oauth:client-assertion-type:jwt-bearer*.
`client_assertion`	For a client application that uses JSON Web Token (JWT) bearer tokens, a single JWT.
`username`	The username of an Acumatica ERP user.
`password`	The password for the specified `username`.
`scope`	The access scope that is requested by the client application. The scope can be a combination of the following values, delimited by spaces: • `api`: Requests access to the REST API, screen-based SOAP API, and OData interface. If this scope is granted and the `api:concurrent_access` scope is not granted, Acumatica ERP manages the sessions of the application through tokens. Acumatica ERP issues the first access token along with the session ID. If the client application requests a new access token by presenting a refresh token, Acumatica ERP reuses the session ID that was issued for the first access token issued with the refresh token. That is, the system uses a single session for each access granted to the client application. For details about the license limitations related to the number of sessions for client applications, see *License Restrictions for API Users*.

Parameter	Description
	• `offline_access`: Requests that a refresh token be granted. If a user grants this scope to the application, Acumatica ERP issues to the client application a refresh token along with the access token. When the access token has expired, the client application can request a new access token by sending a request to the token endpoint and providing the refresh token. By default, the whole chain for the refresh token expires 30 days after the initial authentication process. However, you can change these settings in the **Refresh Tokens** section of the Summary area of the *Connected Applications* (SM303010) form. For details, see *Registration of an OAuth 2.0 or OIDC Application: Sliding Expiration of Refresh Tokens*. • `api:concurrent_access`: Requests permission for the concurrent use of multiple types of web service APIs. If a user grants this scope to the application, the client application can access data in Acumatica ERP in concurrent mode. In this case, Acumatica ERP can maintain multiple sessions for the client application, managing session IDs through cookies. We recommend that the client application request this scope only if concurrent access is required for the client application. For details about the license limitations related to the number of sessions for client applications, see *License Restrictions for API Users*.

Response

Acumatica ERP verifies the provided application credentials and issues the access token, which the client application should provide with each data request to Acumatica ERP.

A successful response includes the following parameters in the response body.

Parameter	Description
`token_type`	The type of the access token, which is *Bearer*. The parameter is returned only if the `api` scope was granted.
`access_token`	The access token. The parameter is returned only if the `api` scope was granted.
`expires_in`	The period of time (in seconds) during which the access token is valid. The parameter is returned only if the `api` scope was granted.
`scope`	The scope for which the access token is valid.
`refresh_token`	The refresh token. The parameter is returned only if the `offline_access` scope was granted.

Example

An example of a request is shown below. (Line breaks are for display purposes only.)

```
POST /identity/connect/token HTTP/1.1
Host: https://localhost/AcumaticaDB
Content-Type: application/x-www-form-urlencoded

grant_type=password
&client_id=8E0761D9-F4EC-2D4B-A60F-BCE2708C6FDD%40U100
&client_secret=O19LLT5Z0SzFbCIKLXLqQQ
&username=admin
&password=123
```

```
&scope=api%20offline_access
```

A successful response has the body shown in the following example.

```
{
    "access_token": "u39uoZj9A4fj2T80Zx0Qirznr0oqNb1qK92c48ZdxUg",
    "expires_in": 3600,
    "token_type": "Bearer",
    "scope": "api offline_access"
}
```

Implementing the Hybrid Flow

The Hybrid flow offers a balance between security and usability by leveraging the advantages of both the Authorization Code flow (such as secure token exchange) and the Implicit flow (such as immediate access tokens). In this chapter, you can find details about the implementation of the Hybrid flow.

Hybrid Flow: General Information

When you implement OpenID Connect (OIDC) in a client application to make the application work with Acumatica ERP, you can use the Hybrid flow. This authorization flow is a combination of the Authorization Code flow and the Implicit flow. As with the Authorization Code flow, with the Hybrid flow, the client application requests the authorization code at the authorization endpoint. As with the Implicit flow, with the Hybrid flow, the client application requests the ID token, access token, or both at the token endpoint. The Hybrid flow allows an application to have immediate access to an ID token while providing secure retrieval of access and refresh tokens.

Learning Objectives

In this chapter, you will learn how to implement a client application that uses the Hybrid flow.

Applicable Scenarios

You implement the Hybrid flow in a client application that can securely store client secrets when the application needs to immediately access information about the user, but must perform some processing before gaining access to protected resources for a long period.

Hybrid Flow

For the support of the Hybrid flow, you implement the following general steps in the application:

1. **Obtaining tokens from the authorization endpoint**

   The client application connects to the authorization endpoint of Acumatica ERP.

   The authorization endpoint directs the user of the client application to the sign-in page of Acumatica ERP, where the user should enter the credentials to sign in to a tenant configured in the Acumatica ERP instance.

   

   The user must sign in to the tenant that was specified in the `client_id` URL parameter passed to the authorization endpoint. (This tenant is selected by default on the sign-in page.)

   If the credentials are accepted by Acumatica ERP, the system displays the consent form, where the user can confirm that the application has access to the requested scopes. Only the scopes that were requested by the application are displayed on the consent form.

If the user has successfully signed in to Acumatica ERP and has granted the access, a response is sent to the redirect URI specified in the authorization request. The response can contain an ID token, access token, and authorization code.

If the ID token is retrieved, the client application validates it by using the key that is available on the *OpenID Connect Preferences* (SM303030) form. The client application can obtain the key through a `GET` request to the following URL: *[<Acumatica ERP instance URL>]/identity/.well-known/openid-configuration/jwks*. The ID token contains the claims to which the user has granted access.

For details on the requesting of tokens from the authorization endpoint, see *Hybrid Flow: Obtaining of an Authorization Code, Access Token, and ID Token from the Authorization Endpoint*.

2. **Obtaining tokens from the token endpoint**

   If the client application uses JSON Web Token (JWT) bearer tokens, the application generates a JWT and signs it with the private key.

   The client application connects to the token endpoint of Acumatica ERP, submits the authorization code, and provides a signed JWT or a shared secret. If a JWT is provided, Acumatica ERP verifies the JWT signature by using the public key (which was specified during the registration of the client application in Acumatica ERP) and validates the JWT payload. If a shared secret is provided, Acumatica ERP verifies the provided application credentials.

   If verification is completed successfully, Acumatica ERP issues the access token, the ID token, and the refresh token if these tokens have been requested by the application. The client application should provide the access token with each data request to Acumatica ERP.

   If the ID token is retrieved, the client application validates it by using the key that is available on the *OpenID Connect Preferences* (SM303030) form. The client application can obtain the key through a `GET` request to the following URL: *[<Acumatica ERP instance URL>]/identity/.well-known/openid-configuration/jwks*. The ID token contains the claims to which the user has granted access.

   For more information on this process, see *Hybrid Flow: Obtaining of an Access Token and ID Token from the Token Endpoint*.

3. **Optional: Retrieving the user information**

   The client application requests user information from Acumatica ERP and provides the access token with this request. Acumatica ERP returns the information for which the user has provided the consent. For details about this request, see *OAuth 2.0 and OIDC: Obtaining of the User Data*.

   

   The recommended way of obtaining the user data is to parse the validated ID token, which contains the same claims as the ones that are obtained through this request.

4. **Optional: Working with data in Acumatica ERP**

   The client application requests data from Acumatica ERP and provides the access token with this request. Acumatica ERP returns the requested data. For details on this process, see *OAuth 2.0 and OIDC: Working with Data in Acumatica ERP*.

When the access token expires, the client application can request a new access token by providing a refresh token, as described in *OAuth 2.0 and OIDC: Refreshing of an Access Token*.

For details on the OAuth 2.0 authorization mechanism, see the specification at *https://tools.ietf.org/html/rfc6749*. For details on the OIDC authorization mechanism, see the specification at *https://openid.net/specs/openid-connect-core-1_0.html#Authentication*.

The configuration of the OpenID Connect protocol that is used by an Acumatica ERP website can be displayed by a request to the following URL: *[<Acumatica ERP instance URL>]/identity/.well-known/openid-configuration*. (In this request, *[<Acumatica ERP instance URL>]* stands for the URL of the Acumatica ERP website.)

Hybrid Flow Diagram

The following diagram illustrates the Hybrid flow.

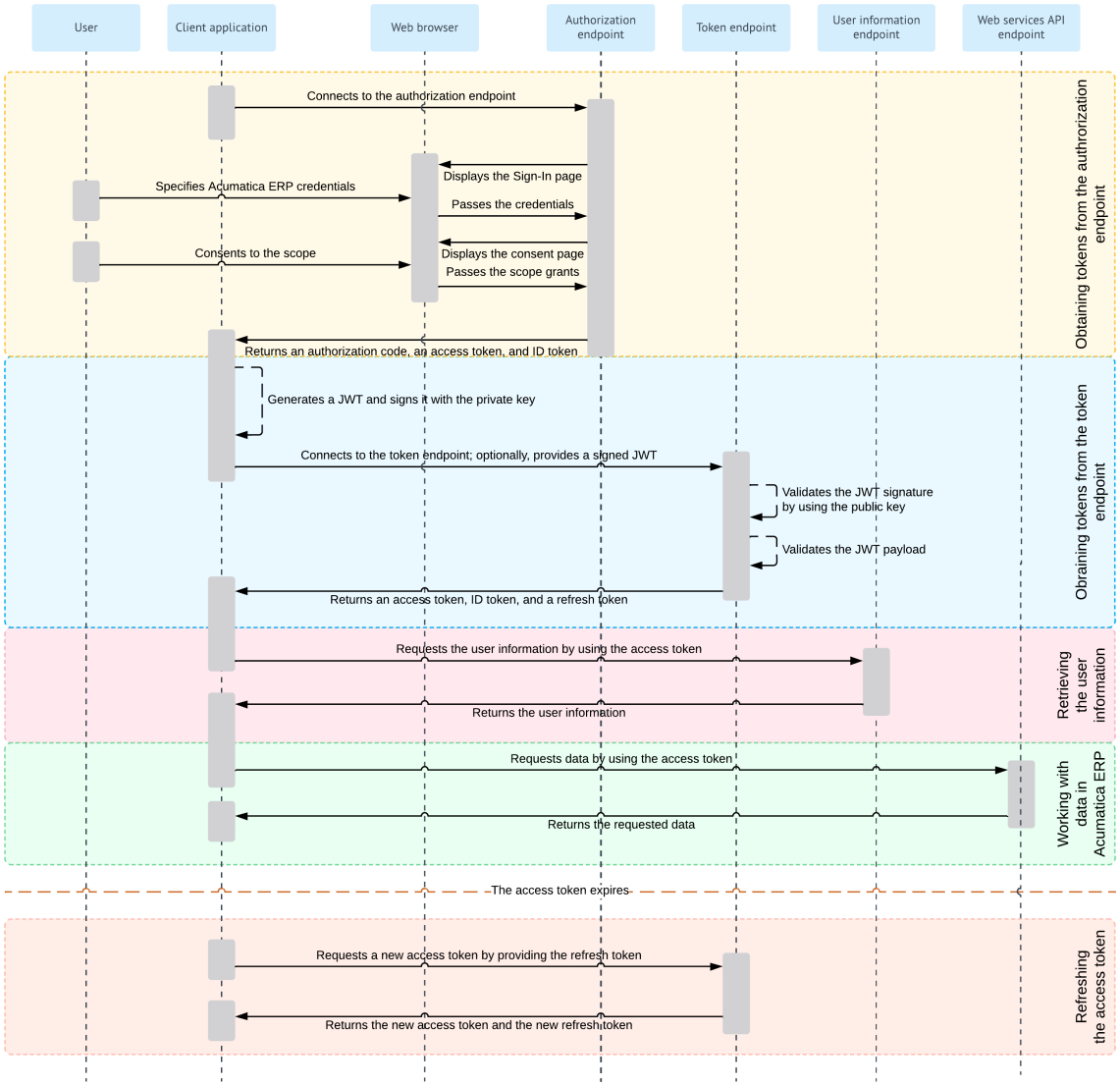

Figure: Hybrid flow

Hybrid Flow: Obtaining of an Authorization Code, Access Token, and ID Token from the Authorization Endpoint

To obtain an authorization code, ID token, and access token from the authorization endpoint, the client application connects to the authorization endpoint of Acumatica ERP with the `GET` HTTP method and specifies the parameters of the request in the URL. For details on the request and the response, see the following sections.

HTTP Method and URL

The client application connects to the authorization endpoint of Acumatica ERP with the `GET` method. The client application can use one of the following approaches for the URL:

- If the client application supports OpenID Connect Discovery, the client application can use the discovery endpoint address, as shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/.well-known/openid-configuration
```

> ⚠ We recommend that the client application use the discovery endpoint address, which eliminates the need to change the application if the target endpoint address changes.

- The client application can directly use the authorization endpoint address, which is shown in the following code.

```
https://<Acumatica ERP instance URL>/identity/connect/authorize
```

Parameters

The client application should specify the following URL parameters.

Parameter	Description
`response_type`	The type of the response, which can be one of the following: • `code id_token`: Is used to retrieve an ID token and authorization code. • `code token`: Is used to retrieve an access token and authorization code. No ID token is returned. • `code id_token token`: Is used to retrieve an ID token, access token, and authorization code.
`client_id`	The client ID that was assigned to the client application during the registration of the application in Acumatica ERP. The client ID must have the format in which the ID was generated during the registration of the application. That is, the client ID must include an auto-generated string and the ID of the tenant, such as *88358B02-A48D-A50E-F710-39C1636C30F6@MyTenant*. The client application will have access to the data of the tenant specified in the client ID.
`redirect_uri`	The URI in the client application to which the response to the request should be sent. The URI must exactly match one of the values specified for the application in the **Redirect URI** column on the **Redirect URIs** tab of the *Connected Applications* (SM303010) form.
`response_mode`	The way the system sends the request to the redirect URI in response for the authorization request. The response mode can be one of the following: • `form_post`: The system uses the `POST` HTTP method to send a request to the `redirect_uri` address. The request body, which includes the response parameters, has `application/x-www-form-urlencoded` format. • `fragment`: The system redirects the client application to the `redirect_uri` address and adds all response parameters to the fragment component of the redirect URI.

Parameter	Description
`scope`	The access scope that is requested by the client application. The scope can be a combination of the following values, delimited by spaces: • `openid`: Requests access to the personal information of the user. This scope is mandatory for the Hybrid flow. • `email`: Requests disclosure of the user's email address. • `profile`: Requests disclosure of the user's profile information. • `phone`: Requests disclosure of the user's phone number. • `api`: Requests access to the REST API, screen-based SOAP API, and OData interface. If this scope is granted and the `api:concurrent_access` scope is not granted, Acumatica ERP manages the sessions of the application through tokens. Acumatica ERP issues the first access token along with the session ID. If the client application requests a new access token by presenting a refresh token, Acumatica ERP reuses the session ID that was issued for the first access token issued with the refresh token. That is, the system uses a single session for each access granted to the client application. For details about the license limitations related to the number of sessions for client applications, see *License Restrictions for API Users*. • `offline_access`: Requests that a refresh token be granted. If a user grants this scope to the application, Acumatica ERP issues to the client application a refresh token along with the access token. When the access token has expired, the client application can request a new access token by sending a request to the token endpoint and providing the refresh token. By default, the whole chain for the refresh token expires 30 days after the initial authentication process. However, you can change these settings in the **Refresh Tokens** section of the Summary area of the *Connected Applications* (SM303010) form. For details, see *Registration of an OAuth 2.0 or OIDC Application: Sliding Expiration of Refresh Tokens*. • `api:concurrent_access`: Requests permission for the concurrent use of multiple types of web service APIs. If a user grants this scope to the application, the client application can access data in Acumatica ERP in concurrent mode. In this case, Acumatica ERP can maintain multiple sessions for the client application, managing session IDs through cookies. We recommend that the client application request this scope only if concurrent access is required for the client application. For details about the license limitations related to the number of sessions for client applications, see *License Restrictions for API Users*.
`nonce`	A string value that is used to associate a client session with an ID token.

Response

If the user is successfully signed in to Acumatica ERP and has granted access, a response is sent to the redirect URI specified in the authorization request. The `response_mode` parameter of the authorization request defines the way the request is sent. The response includes the following parameters.

The refresh token is not returned from the authorization endpoint. To obtain the refresh token, you need to send a request to the token endpoint with the received authorization code, as described in *Hybrid Flow: Obtaining of an Access Token and ID Token from the Token Endpoint*.

Parameter	Description
`code`	The authorization code.

Parameter	Description
`id_token`	The ID token associated with the authenticated session. The ID token contains three parts, which are separated by periods. The parts are Base64 encoded. The second part contains the claims to which the user granted access. For details on the ID token structure, see *https://openid.net/specs/openid-connect-core-1_0.html#IDToken* and *https://www.rfc-editor.org/rfc/rfc7519.html*. We recommend that you use the existing standard libraries for parsing the tokens. The parameter is returned only if the `openid` scope was granted.
`scope`	The scope for which the access token and ID token are provided. The returning of this parameter is optional.
`access_token`	The access token. The parameter is returned only if the `api` scope was granted.
`token_type`	The type of the access token, which is *Bearer*. The parameter is returned only if the `api` scope was granted.
`expires_in`	The period of time (in seconds) during which the access token is valid. The parameter is returned only if the `api` scope was granted.

Example: openid and email Scopes

The following example requests the *openid* and *email* scopes. (Line breaks are for display purposes only.)

```
GET https://localhost/AcumaticaDB/identity/connect/authorize?
response_type=code id_token
&client_id=58FCCFBD-0CF3-C047-B720-A631C976A8DD@U100
&redirect_uri=https://localhost
&scope=openid email
&response_mode=fragment
&nonce=test
```

Once the user grants access to the requested scopes, Acumatica ERP redirects the client application to the following URL.

```
https://localhost/#
code=fXatQXiNwxDc3YSy7Agjz_fKAJBUVN2UmpqTMLtVidY
&id_token=eyJ...gzw
&scope=openid%20email
```

Example: openid, email, profile, and api Scopes

The following example requests the *openid*, *email*, *profile*, and *api* scopes. (Line breaks are for display purposes only.)

```
GET https://localhost/AcumaticaDB/identity/connect/authorize?
response_type=code id_token token
&client_id=58FCCFBD-0CF3-C047-B720-A631C976A8DD@U100
&redirect_uri=https://localhost
&scope=openid email profile api
&response_mode=fragment
&nonce=test
```

Once the user grants access to the requested scopes, Acumatica ERP redirects the client application to the following URL.

```
https://localhost/#
code=Xa8dL8wAL23PmZEdoCBzTDJyj46_NPx_pplzlf-tFas
&id_token=eyJ...EMo
&token_type=Bearer
&expires_in=3600
&scope=openid%20email%20profile%20api
```

Hybrid Flow: Obtaining of an Access Token and ID Token from the Token Endpoint

To obtain an ID token and access token from the token endpoint, a client application that implements the Hybrid flow connects to the token endpoint of Acumatica ERP with the `POST` method. For details on the request and the response, see the following sections.

HTTP Method and URL

The client application connects to the token endpoint of Acumatica ERP with the `POST` method. The client application can use one of the following options for the URL:

- If the client application supports OpenID Connect Discovery, the client application can use the discovery endpoint address, as shown in the following code.

  ```
  https://<Acumatica ERP instance URL>/identity/.well-known/openid-configuration
  ```

  

  We recommend that the client application use the discovery endpoint address, which eliminates the need to change the application if the target endpoint address changes.

- The client application can directly use the token endpoint address, as shown in the following code.

  ```
  https://<Acumatica ERP instance URL>/identity/connect/token
  ```

HTTP Header

You use the following HTTP header.

Key	Value
`Content-Type`	`application/x-www-form-urlencoded`

Request Body

You specify the following parameters in the request body.

Parameter	Description
`grant_type`	The type of the flow, which must be set to *authorization_code* for the Hybrid flow.

Parameter	Description
`client_id`	The client ID that was assigned to the client application during the registration of the application in Acumatica ERP. The client ID must have the format in which the ID was generated during the registration of the application. That is, the client ID must include an auto-generated string and the ID of the tenant, such as *88358B02-A48D-A50E-F710-39C1636C30F6@MyTenant*. The client application will have access to the data of the tenant specified in the client ID.
`code`	The authorization code that the client application has received from the authorization endpoint.
`client_secret`	For a client application that uses a shared secret, the value of the secret that was created for the client application during the registration of the application in Acumatica ERP.
`client_assertion_type`	For a client application that uses JSON Web Token (JWT) bearer tokens, the client assertion type, which must be set to *urn:ietf:params:oauth:client-assertion-type:jwt-bearer*.
`client_assertion`	For a client application that uses JSON Web Token (JWT) bearer tokens, a single JWT.
`redirect_uri`	The URI in the client application to which the response to the request should be sent. The URI must exactly match one of the values specified for the application in the **Redirect URI** column on the **Redirect URIs** tab of the *Connected Applications* (SM303010) form.

Response

Acumatica ERP verifies the provided application credentials and issues an access token, an ID token, and a refresh token if they have been requested by the application. The client application should provide the access token with each data request to Acumatica ERP.

A successful response includes the following parameters in the response body.

Parameter	Description
`token_type`	The type of the access token, which is *Bearer*. The parameter is returned only if the `api` scope was granted.
`access_token`	The access token. The parameter is returned only if the `api` scope was granted.
`expires_in`	The period of time (in seconds) during which the access token is valid. The parameter is returned only if the `api` scope was granted.
`scope`	The scope for which the access token and ID token are provided. The returning of this parameter is optional.
`refresh_token`	The refresh token. The parameter is returned only if the `offline_access` scope was granted.

Parameter	Description
`id_token`	The ID token associated with the authenticated session. The ID token contains three parts, which are separated by periods. The parts are Base64 encoded. The second part contains the claims to which the user granted access. For details on the ID token structure, see *https://openid.net/specs/openid-connect-core-1_0.html#IDToken* and *https://www.rfc-editor.org/rfc/rfc7519.html*. We recommend that you use the existing standard libraries for parsing the tokens. The parameter is returned only if the `openid` scope was granted.

Example

The following example shows a request for access token with a shared secret provided with the request. (Line breaks are for display purposes only.)

```
POST /identity/connect/token HTTP/1.1
Host: https://localhost/AcumaticaDB
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&client_id=C07F7B7A-8947-3C56-2B27-A46CA1F8EF8F@U100
&client_secret=sJli5nNartFBX4Ckzpb68g
&code=z0ExIPH9pAdJSc5nDVakHwYW2jnt91B9oyoZQvdp3cQ
&scope=openid%20email%20profile%20api%20offline_access
&redirect_uri=https://localhost
```

A successful response has the body shown in the following example.

```
{
    "id_token": "eyJ...Z5A",
    "access_token": "9zx2acU03l0ORLHfqwdCPxFHWJMzlLDqrOfJjlZcb_I",
    "expires_in": 3600,
    "token_type": "Bearer",
    "refresh_token": "sbUAHaA7xST3vnvlsanRh3M5EWNmAW_fu6CX16ZEQqM",
    "scope": "openid email profile api offline_access"
}
```

Limiting Connections of Integrated Applications

Acumatica ERP licenses include limitations that affect the external applications that are integrated with Acumatica ERP through the web services APIs. In this chapter, you can find details about these limitations and the limitations that can also be configured for integrated applications.

For general information about Acumatica ERP licenses, see *Instance Deployment: Feature Activation and Licensing*.

License Restrictions for API Users

API users are client applications that sign in to Acumatica ERP by using one of the following ways:

- The sign-in method of the contract-based REST API
- The sign-in method of the screen-based SOAP API
- The OAuth 2.0 mechanism of authorization for applications

Each Acumatica ERP license includes the limits on the number of web services API users, the number of concurrent API requests, and the number of web services API requests per minute. You can view these limits of the Acumatica ERP license on the *License Monitoring Console* (SM604000) form. The trial license sets the limit on the number of web services API users to two and imposes no other limits.

The following sections describe these limits and the lifecycle of API requests.

Number of Web Services API Users

On the **License** tab of the *License Monitoring Console* (SM604000) form, the **Maximum Number of Web Services API Users** box displays the license restriction for the number of API users that can work with Acumatica ERP. When an extra API user tries to sign in to the system and the number of API user sessions exceeds your license restriction, an error message is returned and the sign-in process is interrupted. The following diagram shows an example of how this limitation works with three sign-in requests when the **Maximum Number of Web Services API Users** is set to *2*.

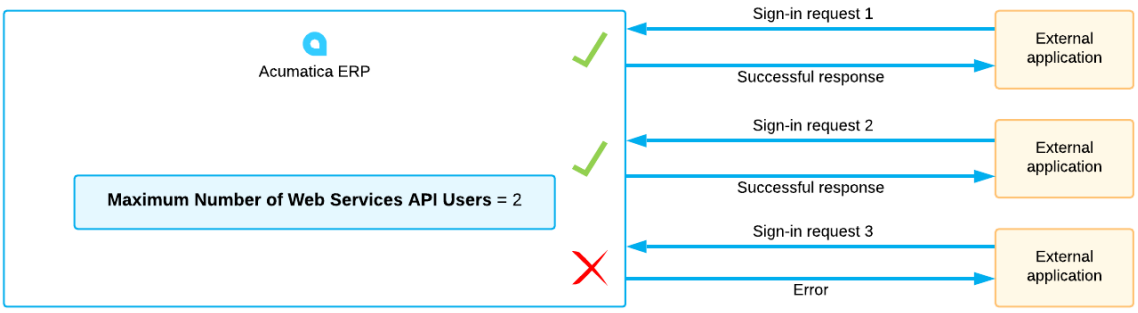

Figure: Rejection of an API user

To avoid exceeding the maximum number of API users, external applications must properly implement the signing in to and signing out from Acumatica ERP. If an external application does not close its sessions in Acumatica ERP (that is, does not sign out from Acumatica ERP in each session), this application can prevent other applications from signing in. For details about the implementation of signing in and signing out, see the following sources:

- *Sign In to the Service* for the REST API
- *Login() Method* for the screen-based SOAP API

Number of Concurrent Web Services API Requests

All incoming API requests (except the requests to sign out from Acumatica ERP, which are processed immediately) are placed in an internal queue. To process these requests, the server uses the API processing cores, which execute the requests in parallel. The number of cores is no more than the maximum number of concurrent web services API requests, which is specified in the **Maximum Number of Concurrent Web Services API Requests** box on the **License** tab of the *License Monitoring Console* (SM604000) form.

The API processing cores take the requests from the queue one by one. If the limit for the number of concurrent web services API requests has been reached (that is, if all API cores are processing the requests), the next concurrent request waits in the queue and is processed when one of the previous requests has completed. These requests in the queue are taken into account in the statistics of the delayed requests (see *Lifecycle of the Requests*).

> ⚠ The system sends a response to the request when the request is fully processed or declined. For details about the lifecycle of the requests, see *Lifecycle of the Requests*.

The following diagram shows an example of how this limitation works with **Maximum Number of Concurrent Web Services API Requests** set to 3.

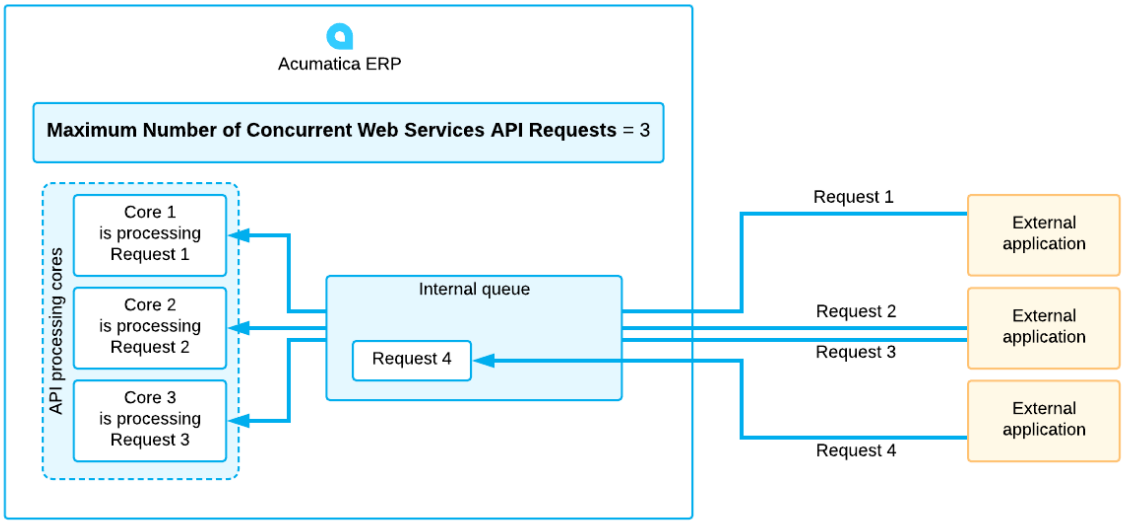

Figure: Request 4 waiting in the internal queue

Number of Web Services API Requests per Minute

The maximum number of requests that can be processed per minute is shown in the **Maximum Number of Web Services API Requests per Minute** box on the **License** tab of the *License Monitoring Console* (SM604000) form.

If the number of requests in the past minute has reached 50% of the limit specified for the license, the current request is delayed. The system calculates the delay as follows:

1. When a request enters the queue, the system checks how many requests have been processed for the past minute. The system calculates the past minute from the moment the request has come.

   For the example in this section, suppose that in a particular license, the limit of the number of web services API requests per minute is 50. Further suppose that the system has processed 25 requests in the past minute. These numbers will be used in further computations.

2. The system finds the time when the first request in the past minute has been received.

For this example, suppose that the first request in the past minute has been received at 02:40:20. Further suppose that the time of the current request is 02:41:00.

3. The system finds the time that remains until the end of the minute since the receipt of the first request in the past minute.

   In this example, the end of the minute since the first request is 02:41:20. The remaining time is 20 seconds, calculated as `02:41:20 - 02:41:00`.

4. The system calculates the remaining number of requests that can be processed in the minute, which is the number in the **Maximum Number of Web Services API Requests per Minute** box minus the number of requests that the system has already processed in the past minute.

   In the example of this section, the remaining number of requests is 25, calculated as `50 - 25`.

5. The system calculates the delay as the remaining time divided by the remaining number of requests.

   In our example, the delay is 0.8 seconds, calculated as `20 / 25`.

> ⚠ The system sends a response to the request when the request is fully processed or declined. For details about the lifecycle of the requests, see *Lifecycle of the Requests*.

The following diagram shows an example of how this limitation works with **Maximum Number of Web Services API Requests per Minute** set to 50 and with 25 requests processed in the minute.

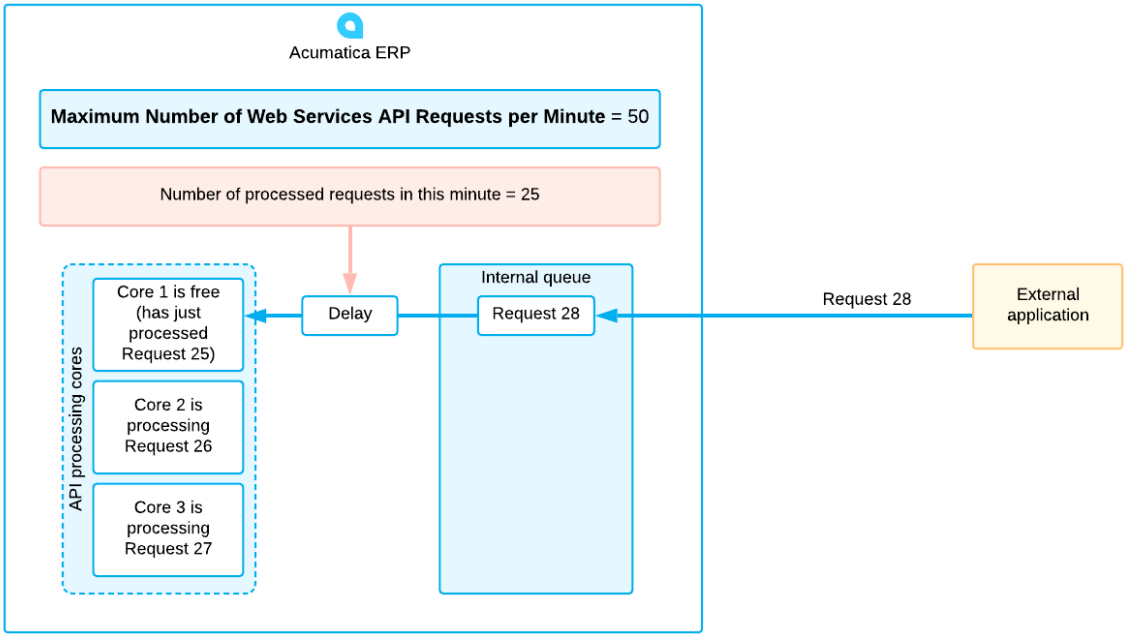

Figure: Request 28 delayed in the internal queue

Decline of the Requests

The request will be declined only if the number of requests in the internal queue is greater than 20, or the request remains in the queue for more than 10 minutes. Acumatica ERP gives no indication that a request is added to the queue or being processed; the application that makes a request should stay connected and wait for the response. A declined request is not counted in the number of processed requests.

For example, suppose that during one second, each of 50 external applications sends a web services API request to one Acumatica ERP server. Suppose also that each request is processed for 1 second and that the maximum number of concurrent web services API requests in the license is 16. In this case, the first 16 requests will be

passed to 16 API processing cores immediately. The next 20 requests will wait in the internal queue, and the last 14 requests will be declined.

Lifecycle of the Requests

The system sends a response to the request when the request is fully processed or declined. You can view the statistics of the delayed and declined requests on the **Statistics** tab of the *License Monitoring Console* (SM604000) form.

The following diagram shows the lifecycle of a web services API request. For details about how the sign-in requests are processed, see *Limitation of API Connections for Integrated Applications*.

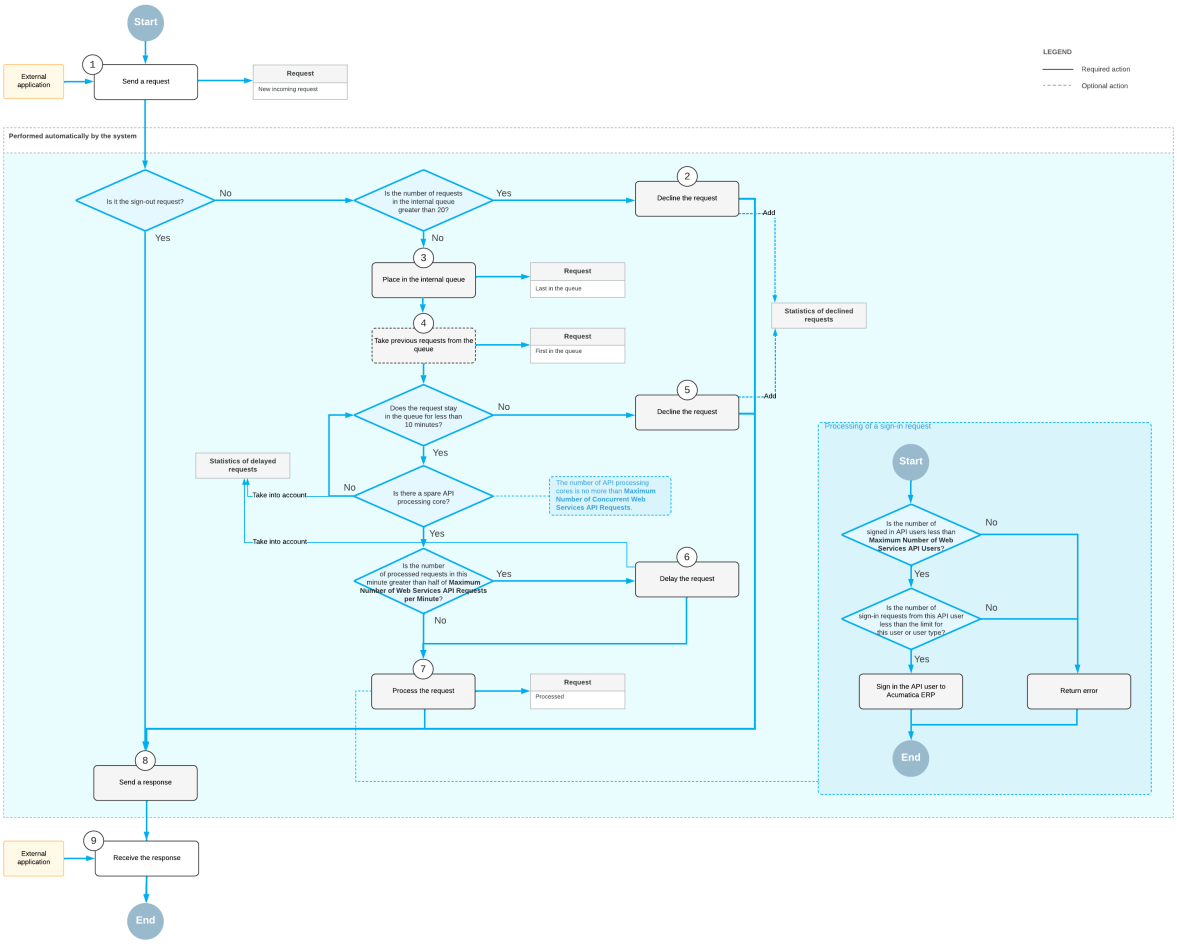

Figure: Lifecycle of a request

Limitation of API Connections for Integrated Applications

If an integrated application does not properly sign out from Acumatica ERP, this application can use all of the API sessions included in the license (that is, reach the maximum number of API users, which is described in *License Restrictions for API Users*), thus preventing other integrated applications from signing in. To avoid such situations, you can limit the number of sessions for either each integration application assigned to a user type or an individual integration application.

Before configuring these limits, on the **Warnings** tab of the *License Monitoring Console* (SM604000) form, you can review whether any of the integrated application opens more than three sessions at a time (which is a predefined system value for this form). If the number of sessions has not been limited for an integrated application and this

application opens more than three sessions at a time, a warning is generated and listed in the table on this tab. This warning can indicate that you should configure a limitation for the particular integration application or review the implementation of signing in and signing out in the application.

Configuration of the Limitations

You limit the number of sessions used by each integration application on the *User Types* (EP202500) form. Also, if it is necessary to specify a different limit for a particular integration application, you can specify this limit on the *Users* (SM201010) form. For details about how to limit the number of sessions in both of these ways, see *To Limit the Number of API Connections of Integrated Applications* and *To Limit API Connections of a Particular Application*.

Lifecycle of Sign-in Requests

The lifecycle of any type of requests (including sign-in requests) is described in *Lifecycle of the Requests*. This section describes peculiarities of the lifecycle of sign-in requests.

The limitations for the number of API users and the limit of the number of sessions for a particular application are taken into account when one of the API processing cores takes the sign-in request. The API processing core then processes the sign-in request as follows:

1. Checks whether the maximum number of API users is exceeded. For details about this limit, see *License Restrictions for API Users*.
2. Checks whether the limit of the number of sessions specified for this API user is exceeded.

   If the number of sessions opened by the integrated application reaches the limit specified for this integrated application on either the *User Types* (EP202500) form or the *Users* (SM201010) form, the integrated application cannot open any new session (that is, cannot sign in to Acumatica ERP) until it closes one of the existing sessions. The system returns an error in the response to the sign-in request.

   If no limit is specified for the integration application on the *User Types* form or on the *Users* form, the integration application can open no more sessions than the maximum number of API users in the license.

The diagram in *Lifecycle of the Requests* shows how a sign-in request is processed by an API processing core.

Related Links

- *To Limit the Number of API Connections of Integrated Applications*
- *To Limit API Connections of a Particular Application*

To Limit the Number of API Connections of Integrated Applications

On the *User Types* (EP202500) form, you limit the number of sessions (that is, API connections) used by each of the integrated applications. You can also specify a different limit for a particular integrated application as described in *To Limit API Connections of a Particular Application*.

Before You Proceed

On the *User Roles* (SM201005) form, create a user role that will be used by integrated applications. For details about user roles, see *Configuring User Roles*.

To Limit API Connections of Each Integrated Application

1. Open the *User Types* (EP202500) form.
2. Create a user type that will be used by integrated applications as described in *To Add a User Type*. On the **Allowed Roles** tab, add the role that you created in the preliminary instructions (*Before You Proceed*).

3. On the **Login Rules** tab, specify the following settings:
   - **Allowed Login Type**:
     - *API* if your integration application uses the `Login` method of one of the web services API to sign in to Acumatica ERP
     - *Unrestricted* if your integration application uses OAuth 2.0 or OpenID Connect
   - **Max. Number of Concurrent Logins**: The maximum number of sessions (API connections) you want to allow for each integration application
   - **Turn Off Two-Factor Authentication**: Selected
4. Save your changes.
5. On the *Users* (SM201010) form, create a user account for each integrated application that can work with this Acumatica ERP instance. For each user account, specify the type that you have created in the previous steps and add the role that you have created in the preliminary instructions (*Before You Proceed*). For details about the creation of a user account, see *User Access: To Add a User Account*.
6. In each of the integrated applications, implement the signing in to Acumatica ERP with its own user account (one of the user accounts created in the previous step). For details about the implementation of signing in, see the following sources:
   - *Sign In to the Service* for the REST API
   - *Login() Method* for the screen-based SOAP API

Related Links

- *Limitation of API Connections for Integrated Applications*
- *To Limit API Connections of a Particular Application*

To Limit API Connections of a Particular Application

On the *Users* (SM201010) form, you can specify a limit for the number of sessions used by a particular integrated application.

> ⚠ Instead of specifying a limit for a particular integrated application, on the *User Types* (EP202500) form, you can limit the number of sessions for each integrated application. For details about how to limit the number of sessions for each integrated application, see *To Limit the Number of API Connections of Integrated Applications*.

If a limit is specified on the *Users* form for a user account associated with an integrated application, this limit overrides any limit specified on the *User Types* form. If no limit is specified on the *Users* form for a user account associated with an integration application, any limit specified on the *User Types* form is used.

To Limit API Connections of a Particular Application

1. Open the *Users* (SM201010) form.
2. For the user account that the integration application uses, type the account's maximum number of sessions in the **Max. Number of Concurrent Logins** box.

Related Links

- *Limitation of API Connections for Integrated Applications*
- *To Limit the Number of API Connections of Integrated Applications*

Configuring Push Notifications for Real-Time Monitoring

Acumatica ERP provides the ability to configure push notifications so that external applications can track the changes to the data in Acumatica ERP. That is, you can configure Acumatica ERP to send notifications to a destination (such as an HTTP address) when specific data changes occur in Acumatica ERP. The external application can receive these notifications and process the information on the data changes. With push notifications, the external application does not need to continually poll the data to find out whether there are any changes to it, which helps improve the performance of the external application.

In this chapter, you can find information on how to configure Acumatica ERP to send push notifications. You can also find information on how to configure push notifications in code.

Push Notifications: General Information

Push notifications are notifications in JSON format that are sent by Acumatica ERP to notification destinations when specific data changes occur in Acumatica ERP. External applications can receive the notifications and process them to retrieve the information about the changes.

Learning Objectives

In this chapter, you will learn the following:

- How to configure push notifications
- Which requirements exist in Acumatica ERP for the data queries that are used by push notifications
- Which push notification destinations you can use
- Which format of push notifications is used by Acumatica ERP
- How to process failed push notifications
- How to include push notification configuration in a customization project

Applicable Scenarios

You define push notifications in Acumatica ERP in the following scenarios:

- You need to send notifications about changes in particular data.
- You need to implement real-time synchronization of changes in the data in Acumatica ERP with the data in an external system.

Configuration of Push Notifications

To work with Acumatica ERP push notifications, you need to configure the following items:

- The data query that defines the data for whose changes Acumatica ERP should send notifications
- The destination to which Acumatica ERP should send notifications
- The way the external application processes the notifications
- The definition of the push notification in Acumatica ERP, which specifies the data query and the notification destination

The following diagram illustrates the sending of a push notification and shows the items that you need to configure to receive push notifications when changes occur.

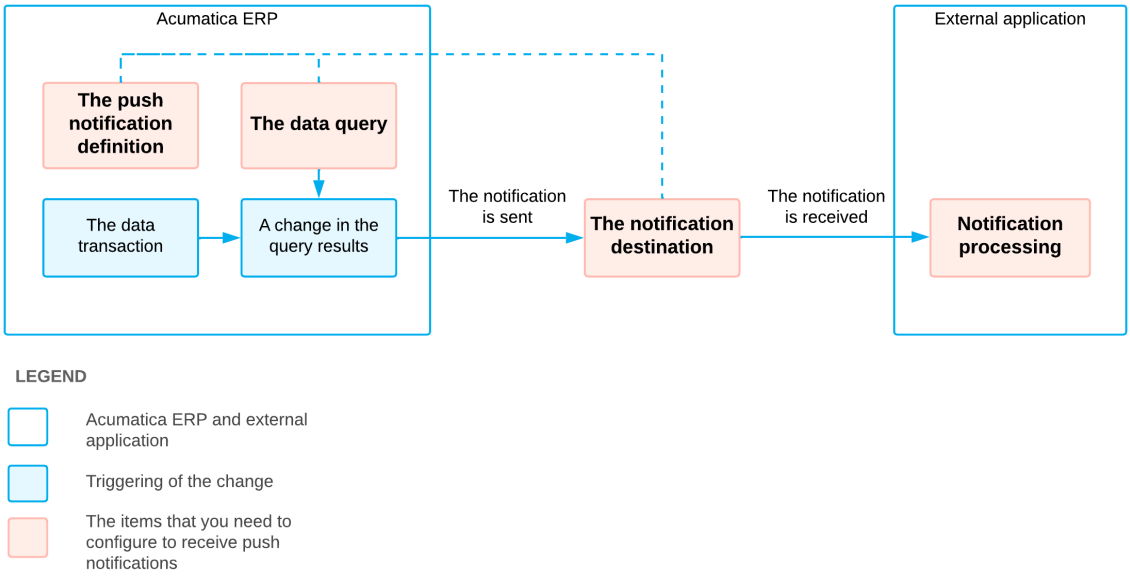

Figure: Sending a push notification

To configure push notifications, you use the *Push Notifications* (SM302000) form. You can configure as many push notifications as you wish.

Data Query

The data query can be defined by either a generic inquiry or a built-in query definition (which is a data query defined in code). For details on generic inquiries, see *Managing Generic Inquiries*. For information on how to create a built-in query definition, see *Push Notifications: To Create a Built-In Query Definition*. You can define multiple queries for one notification destination.

The data query should adhere to the recommendations described in *Push Notifications: Recommendations for the Data Queries*.

Notification Destination

The following predefined notification destinations are provided: webhook (HTTP address), message queue, SignalR hub, and commerce push destination. For more information on the predefined notification destinations, see *Push Notifications: Destinations*. You can also create your own destination type, as described in *Push Notifications: To Create a Custom Destination Type*.

Processing of the Notifications in the External Application

You should configure your external application so that it can process the notifications and extract the information about the data changes. Acumatica ERP sends notifications to notification destinations in JSON format. For details on the format of the notifications, see *Push Notifications: Format*. If your application watches notifications in the SignalR hub, you need to connect to the hub, as described in *Push Notifications: To Connect to the SignalR Hub*.

Definition of the Push Notification

In the definition of the push notification on the *Push Notifications* (SM302000) form, you specify the notification destination and the data queries for which the notifications should be sent. You can also specify particular fields that the system should track in the results of the data queries.

Push Notifications: Recommendations for the Data Queries

For optimal results, you need to follow these recommendations when you create each data query for which you want to configure push notifications:

- Do not use aggregation and grouping in the query; Acumatica ERP does not guarantee that push notifications will work correctly with such queries.
- Do not use joins of multiple detail tables in the query because this may cause the system to hang.
- If you need to join multiple tables, use a left join or an inner join in the data query. If you use an inner join, the query execution may be slower than for a left join.
- Use as simple a data query as possible.
- For a query defined by using a generic inquiry, do not use a formula on the **Results Grid** tab of the *Generic Inquiry* (SM208000) form.
- For a query defined by using a generic inquiry, do not use description fields of selectors on the **Results Grid** tab of the *Generic Inquiry* form. To track changes in the description value, in the generic inquiry, explicitly join the table that contains the description and add the field from this table in the results of the generic inquiry.

Push Notifications: Destinations

When you configure a push notification on the *Push Notifications* (SM302000) form of Acumatica ERP, you select the type of the notification destinations, which can be any of the predefined types described in this topic. You can also create your own destination type, as described in *Push Notifications: To Create a Custom Destination Type*.

Webhook

A webhook is an HTTP address to which Acumatica ERP sends HTTP `POST` requests with notification information. For this destination type, you specify a valid HTTP address in the **Address** box on the *Push Notifications* (SM302000) form. For security reasons, you can specify a header of the HTTP request in the **Header Name** and **Header Value** boxes.

> ⚠ Do not specify the `Accept` and `Content-Type` headers for the request. The values of these headers are specified automatically by the system.

If an integrated application returns an error in reply for a push notification, the push notification is resent. If your application processed the request and sent an error in the reply, your application receives the same push notification again, which can lead to duplicate data in the application. Therefore, you should not reply with an error if a push notification is successfully received.

Acumatica ERP makes at most five attempts to send a push notification automatically. For details on the notifications that failed to be sent, see *Push Notifications: Failed Notifications*.

Message Queue

The message queue is a local or remote private Microsoft message queue. You specify the address of the message queue (such as `MyComputer\private$\TestQueueForPushNotificatons`) in the **Address** box on the *Push Notifications* (SM302000) form. For information on how to configure a private Microsoft message queue, see the Microsoft documentation.

The message queue is the most reliable destination type protected from network failures. However, Acumatica ERP makes at most five attempts to send a push notification automatically. For details on the notifications that failed to be sent, see *Push Notifications: Failed Notifications*.

SignalR Hub

The SignalR hub is the destination type implemented in Acumatica ERP by using the ASP.NET SignalR library. The address of this destination type is `PushNotificationsHub`, which is filled in automatically in the **Address** box on the *Push Notifications* (SM302000) form. This destination type can be used if you can expose neither an HTTP address (webhook) nor a message queue to receive push notifications. If Acumatica ERP is configured to send notifications to the SignalR hub, the external application can connect to Acumatica ERP through websoket or a long-polling mechanism and receive notifications through this connection. If multiple external applications are connected to the SignalR hub, they receive notifications simultaneously. For information on how to connect your application to the SignalR hub of Acumatica ERP, see *Push Notifications: To Connect to the SignalR Hub*.

The SignalR hub destination type is not reliable: If the connection fails or there are no clients connected to the SignalR hub when a notification comes, this notification will not be sent and it cannot be resent later.

Commerce Push Destination

The commerce push destination is a local private Microsoft message queue that is used for notifications sent to a commerce connector. The address of this destination type is `Commerce`, which is filled in automatically in the **Address** box on the *Push Notifications* (SM302000) form. For details about commerce push notifications, see *Real-Time Synchronization for a Connector: How Webhooks and Push Notifications Are Used*.

Acumatica ERP makes at most five attempts to send a push notification automatically. For details on the notifications that failed to be sent, see *Push Notifications: Failed Notifications*.

Push Notifications: Format

Acumatica ERP sends push notifications in JSON format. This topic describes the structure of the notifications.

Elements

The push notifications that Acumatica ERP sends include the following elements in JSON format.

Element	Description
`Inserted`	The rows that are new in the results of the query execution.
`Deleted`	The rows that were in the results of the query execution but are missing after the latest data transaction. You can compare the `Inserted` and `Deleted` rows to identify the rows that have been updated.
`Query`	The query for which Acumatica ERP has produced the notification. The value of the element can be either the name of the generic inquiry or the name of the class with the built-in query definition.
`CompanyId`	The name of the tenant.
`Id`	The unique identifier of the data transaction in Acumatica ERP that has initiated the notification. The external application can use this identifier to omit duplicated notifications.

Element	Description
`TimeStamp`	The `long` value that corresponds to the date and time when the transaction that initiated the notification happened in Acumatica ERP. By using the value of this element, the external application can define the order of notifications.
`AdditionalInfo`	Any additional information that is added to the notification. This element can contain additional information added by the system as well as the custom information. For more information on how to add custom information to push notifications, see *Push Notifications: To Include Additional Information in Push Notifications*.

Example

Suppose that push notifications are configured for the *Stock Items: Last Modified Date* generic inquiry (which displays the **InventoryID**, **StockItem**, **ItemStatus**, and **InventoryItem_lastModifiedDateTime** columns). Acumatica ERP sends the following notification when the status of the *AACOMPUT01* inventory item has been changed from *Active* to *Inactive*.

```
{
  "Inserted":
  [{
    "InventoryID":"AACOMPUT01",
    "StockItem":true,
    "ItemStatus":"Inactive",
    "InventoryItem_lastModifiedDateTime":"2024-05-05T15:16:23.1"
  }],
  "Deleted":
  [{
    "InventoryID":"AACOMPUT01",
    "StockItem":true,
    "ItemStatus":"Active",
    "InventoryItem_lastModifiedDateTime":"2024-05-05T15:16:23.103"
  }],
  "Query":"Stock Items: Last Modified Date",
  "CompanyId":"MyTenant",
  "Id":"1af4d140-5321-41f2-a2ec-50b67f577c6c",
  "TimeStamp":636295833829493672,
  "AdditionalInfo":{}
}
```

Push Notifications: Failed Notifications

Acumatica ERP makes at most five attempts to send a push notification to the HTTP address, message queue, or commerce push destination automatically. If Acumatica ERP cannot send a notification, it logs information on the failed notification and displays this notification on the *Process Push Notifications* (SM502000) form. You can resend the failed notifications for two days, after which the notifications are removed from the Acumatica ERP database.

> If a notification has failed in Acumatica ERP before it was sent (for example, if Acumatica ERP could not retrieve the data for the notification), this notification is not displayed in the table on the *Process Push Notifications* form. Acumatica ERP saves the information about these notifications in the `PushNotificationsErrors` database table if the `api:push-notifications:enable-dead-message-log` key is set to `true` in the `appSettings` section of the `web.config` file of the Acumatica ERP instance.

Viewing a Failed Push Notification

To view a failed push notification, on the *Process Push Notifications* (SM502000) form, for the row that contains the notification you want to view, you click **Show Notification**. The **Notification Event** dialog box, which opens, displays the notification in JSON format. For details on the format of the notification, see *Push Notifications: Format*.

Resending Failed Push Notifications

On the *Process Push Notifications* (SM502000) form, you can resend failed push notifications by doing one of the following:

- If you want to resend particular notifications, in the table on the form, you select the check boxes in the rows that correspond to the notification you want to send, and click **Send** on the form toolbar.
- If you want to resend all notifications, on the form toolbar, you click **Send All** on the form toolbar.

> Push notifications that have the **Truncated** check box selected cannot be resent.

You can also automate the process of resending failed notifications. You first create a generic inquiry based on the `PX.PushNotifications.UI.DAC.PushNotificationsFailedToSend` data access class, which corresponds to the *Process Push Notifications* form. You then configure a push notification for this generic inquiry.

Deleting Failed Push Notifications

On the *Process Push Notifications* (SM502000) form, you can delete failed push notifications by doing one of the following:

- If you want to delete particular notifications, in the table on the form, you select the unlabeled check boxes in the rows that correspond to the notifications you want to delete, and click **Delete** on the form toolbar.
- If you want to delete all notifications, on the form toolbar, you click **Delete All**.

You then need to save your changes on the form.

Push Notifications: To Configure Push Notifications

This activity will walk you through the process of configuring Acumatica ERP to send push notifications to an HTTP address.

Story

The business intelligence (BI) application of the MyStore company needs to display up-to-date information about item availability in warehouses. You will configure Acumatica ERP to track the changes in item availability and to send notifications to an HTTP address when the availability of any item changes. (The processing of these notifications in the BI application is outside of the scope of this activity.)

Process Overview

You will configure Acumatica ERP to monitor changes in the Item Availability Data (INGI0002) custom generic inquiry, which has been configured for this activity. You will then test how Acumatica ERP sends push notifications about the changes to an HTTP address in JSON format.

System Preparation

Before you begin performing the steps of this activity, deploy an instance of Acumatica ERP with the *MyStoreInstance* name and a tenant that has the *MyStore* name and contains the *T100* data.

Step 1: Obtaining an HTTP Address for Push Notifications

To test the push notifications that you will set up Acumatica ERP to send to an HTTP address, you will use the *https://webhook.site/* website, which is an open-source service. To obtain the HTTP address that you will use for testing, do the following:

1. Open *https://webhook.site/*.
2. In the **Your unique URL** section, copy the HTTP address that the site has generated. An example of the address is shown in the following screenshot.

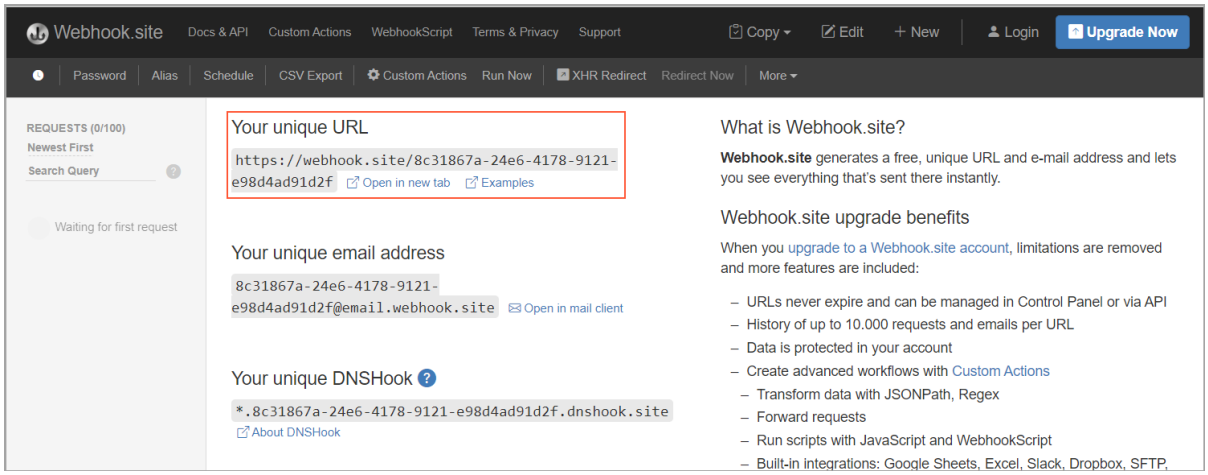

Figure: An HTTP address

Step 2: Configuring Push Notifications

To configure Acumatica ERP to send push notifications to an HTTP address, do the following:

1. In the Summary area of the *Push Notifications* (SM302000) form, specify the following settings:
   - **Destination Name**: `MyBIIntegration`.

     You can specify any name for the target notification destination.
   - **Destination Type**: *Webhook*.

     With *Webhook* selected, Acumatica ERP will send HTTP `POST` requests with notification information to an HTTP address.
   - **Address**: The address that you have created in Step 1.
2. On the **Generic Inquiries** tab, click **Add Row** on the toolbar of the **Inquiries** table.
3. In the added row, do the following:

a. In the **Inquiry Title** column of the added row, select *Item Availability Data*.

b. Select the **Track All Fields** check box.

> If all fields in the results of the data query are tracked, the system produces push notifications for changes of any field in the results of the data query, which can cause the push notification queue to overflow. If you need to track only particular fields in the results of the data query, push notifications for changes in other fields are useless for you but consume system resources. Therefore, in this case, we recommend that you specify particular fields to be tracked in the **Fields** table.

4. On the form toolbar, click **Save**. The following screenshot shows the final configuration.

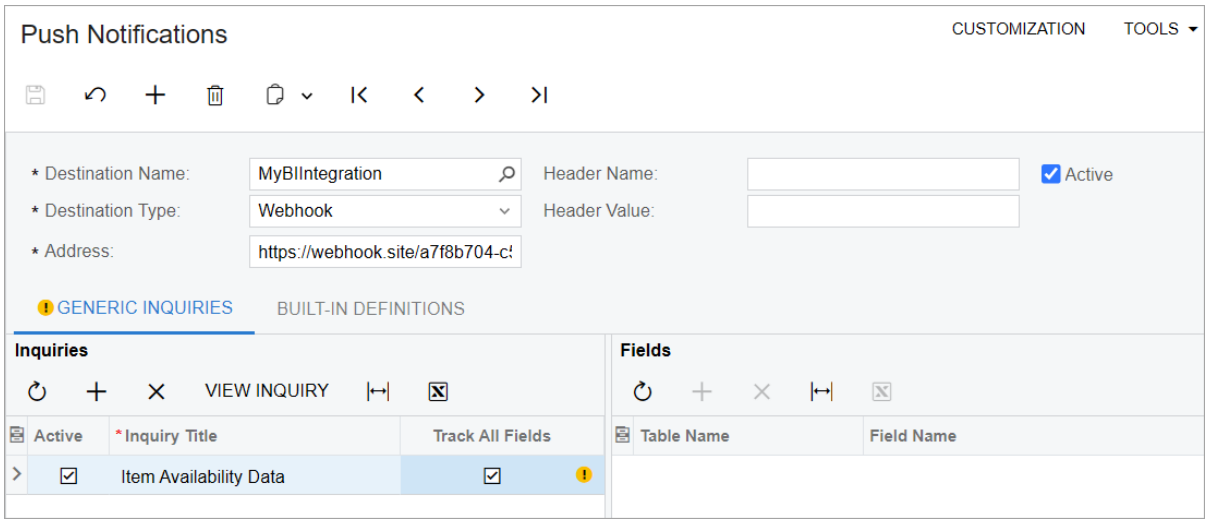

Figure: Push notification configuration

Step 3: Testing Push Notifications

To test the configured notifications, do the following:

1. On the *Sales Orders* (SO301000) form, create a sales order as follows:

   a. In the **Customer** box, select *C000000001*.

   b. Click **Remove Hold**.

   c. On the table toolbar of the **Details** tab, click **Add Items**. In the dialog box that opens, select the unlabeled check box for the *AALEGO500* inventory item, and click **Add & Close**.

   d. On the form toolbar, click **Save**.

2. On the *https://webhook.site/* site, review the push notification that Acumatica ERP sent for the change in the available quantity for the *AALEGO500* inventory item. When you created a sales order, the available quantity of the *AALEGO500* inventory item was changed. The notification contains the information about the new value of the `QtyAvailable` field of the changed record in the `Inserted` element and the information about the previous value of this field in the `Deleted` element. An example of the notification is shown in the following code.

```
{
  "Inserted":
  [
    {
      "Warehouse":"MAIN ",
      "InventoryID":"AALEGO500",
```

```
      "Description":"Lego, 500 piece set",
      "QtyOnHand":1999,
      "QtyAvailable":1974
    }
  ],
  "Deleted":
  [
    {
      "Warehouse":"MAIN",
      "InventoryID":"AALEGO500 ",
      "Description":"Lego, 500 piece set",
      "QtyOnHand":1999,
      "QtyAvailable":1975
    }
  ],
  "Query":"Item Availability Data",
  "CompanyId":"MyStore",
  "Id":"06284bdf-2952-4a9a-bf60-2ad0cd33924e",
  "TimeStamp":638465233453014192,
  "AdditionalInfo":
  {
    "PXPerformanceInfoStartTime":"03/20/2024 09:22:24"
  }
}
```


Push Notifications: To Create a Built-In Query Definition

The following activity shows you how to create a built-in query definition for push notifications.

Story

Suppose that you need to configure Acumatica ERP to send push notifications for a query that is defined as a class in the source code of the application—that is, for a built-in definition of the query.

Process Overview

In a project of your Acumatica ERP extension library, you will define a class that implements the *`PX.PushNotifications.Sources.IInCodeNotificationDefinition`* interface. You will then build the project of your Acumatica ERP extension library and test the built-in query definition.

System Preparation

Before you begin performing the steps of this activity, do the following:

1. Prepare an Acumatica ERP instance with any predefined dataset. You can do it by completing the following prerequisite activity: *Instance Deployment: To Deploy an Instance with Demo Data*.
2. Create a customization project, such as in the *Customization Projects: To Create a Customization Project* prerequisite activity.
3. Create an extension library, as described in the *To Create an Extension Library* prerequisite activity.

You can find the final code of this activity in the *TestInCodeDefinition.cs* file in the Help-and-Training-Examples repository on GitHub.

Step 1: Creating a Built-In Query Definition

To create a built-in query definition, do the following:

1. In a project of your Acumatica ERP extension library, define a class that implements the *PX.PushNotifications.Sources.IInCodeNotificationDefinition* interface. The following code demonstrates the definition of such a class.

```
using PX.PushNotifications.Sources;

public class TestInCodeDefinition : IInCodeNotificationDefinition
{
}
```

Make sure references to the `PX.Data.dll`, `PX.Data.BQL.Fluent`, and `PX.PushNotifications.dll` libraries are included in your project.

2. In the class that implements the `IInCodeNotificationDefinition` interface, implement the `GetSourceSelect()` method of the interface so that the method satisfies the following requirements:
   - The method must return a tuple of a `BqlCommand` object, which defines the data query, and a `PXDataValue` array, which defines the parameters that should be passed to the query.
   - The data query that the method defines should adhere to *Push Notifications: Recommendations for the Data Queries*.

   The following example shows the `GetSourceSelect()` method implementation.

```
using PX.Data;
using PX.PushNotifications.UI.DAC;
using System;
using PX.Data.BQL.Fluent;
using PX.PushNotifications.Sources;

public class TestInCodeDefinition : IInCodeNotificationDefinition
{
    public Tuple<BqlCommand, PXDataValue[]> GetSourceSelect()
    {
        return
          Tuple.Create(
            SelectFrom<PushNotificationsHook>.
            LeftJoin<PushNotificationsSource>.
              On<PushNotificationsHook.hookId.
                 IsEqual<PushNotificationsSource.hookId>>.View
            .GetCommand(), new PXDataValue[0]);
    }
}
```

3. In the class that implements the `IInCodeNotificationDefinition` interface, implement the `GetRestrictedFields()` method of the interface so that the method satisfies the following requirements:
   - The method must return an array of `IBqlField`-derived types, which contains the fields that should be returned from the query.

- If you need to return all fields, the method must return `null`.

The following code shows an example of the implementation of the `GetRestrictedFields()` method.

```
using PX.Data;
using PX.PushNotifications.UI.DAC;
using System;
using PX.Data.BQL.Fluent;
using PX.PushNotifications.Sources;

public class TestInCodeDefinition : IInCodeNotificationDefinition
{
...
```

```
    public Type[] GetRestrictedFields()
    {
        return new[]
        {
          typeof(PushNotificationsHook.address),
          typeof(PushNotificationsHook.type),
          typeof(PushNotificationsSource.designID),
          typeof(PushNotificationsSource.inCodeClass),
          typeof(PushNotificationsSource.lineNbr)
        };
    }
}
```

4. Build the project of your Acumatica ERP extension library.

Step 2: Testing the Built-In Query Definition

On the **Built-In Definitions** tab of the *Push Notifications* (SM302000) form, create a push notification definition and make sure that you can select the new built-in query definition by its class name in the **Class Name** column. The class of the built-in query definition, which implements the `IInCodeNotificationDefinition` interface, is detected by the system and automatically added to the list of classes available for selection on the tab.

After you have defined a push notification definition that uses a built-in query definition on the *Push Notifications* form, you can obtain the results of the data query defined with a built-in query definition by using the following endpoint: *http(s)://<Acumatica ERP instance URL>/PushNotifications/<full class name of the built-in query definition>*.

For example, suppose that you want to retrieve the results of the data query defined with the `PX.PushNotifications.Sources.TestInCodeDefinition` class from a local Acumatica ERP instance with the name *AcumaticaDB*. You should use the following URL to obtain the data: *http(s)://localhost/AcumaticaDB/PushNotifications/PX.PushNotifications.Sources.TestInCodeDefinition*. The endpoint returns the data in JSON format.

Related Links

- *IInCodeNotificationDefinition Interface*

Push Notifications: To Connect to the SignalR Hub

The following activity shows you how to connect the external application to the SignalR hub.

Story

Suppose that you want your external application to receive push notifications from Acumatica ERP but you cannot publish a webhook (for example, for security reasons). Instead you need to configure the system to send notifications to the SignalR hub, from which any connected application can receive the notifications.

Process Overview

In a project of your Acumatica ERP extension library, you will implement the code for sending notifications to the SignalR hub.

System Preparation

Before you begin performing the steps of this activity, do the following:

1. Prepare an Acumatica ERP instance with any predefined dataset. You can do it by completing the following prerequisite activity: *Instance Deployment: To Deploy an Instance with Demo Data*.
2. Create a customization project, such as in the *Customization Projects: To Create a Customization Project* prerequisite activity.
3. Create an extension library, as described in the *To Create an Extension Library* prerequisite activity.

You can find the final code of this activity in the *TestSignalR.cs* file in the Help-and-Training-Examples repository on GitHub.

Step 1: Creating a Push Notification Definition

To configure the system to send push notifications to the SignalR hub, create a push notification definition as follows:

1. On the *Push Notifications* (SM302000) form, in the **Destination Name** box, type the name of the target notification destination, such as `TestSignalR`.
2. In the **Destination Type** box, select *SignalR Hub*. The **Address** box is filled in automatically with *PushNotificationsHub*.
3. For each generic inquiry for which you want Acumatica ERP to send notifications on changes in the inquiry results, do the following on the **Generic Inquiries** tab:
   a. On the table toolbar of the **Inquiries** table, click **Add Row**. The new row has the **Active** check box selected.
   b. In the **Inquiry Title** column of the added row, select the generic inquiry for which you want Acumatica ERP to send notifications.
   c. Do one of the following:
      - If you want the system to send push notifications for changes in any field in the results of the generic inquiry, select the **Track All Fields** check box in the added row.

If all fields in the results of the generic inquiry are tracked, the system produces push notifications for changes of any field in the results of the generic inquiry, which can cause overflow of the push notification queue. If you need to track only particular fields in the results of the generic inquiry, push notifications for changes in other fields are useless for you but consume system resources. Therefore, we recommend that you specify particular fields to be tracked in the **Fields** table.

- If you need to track only particular fields in the results of the generic inquiry, while the row of the **Inquiries** table is still selected, do the following in the **Fields** table for each field that you need to track:
  a. On the table toolbar, click **Add Row**.
  b. In the **Table Name** column of the added row, select the name of the table that contains the field that the system should track.
  c. In the **Field Name** column, select the name of the field that the system should track.

4. On the form toolbar, click **Save**.

Step 2: Connecting to the SignalR Hub

To connect to the SignalR hub, do the following:

1. In a project of your Acumatica ERP extension library, set up a Basic authentication token to authenticate the application in Acumatica ERP, as shown in the following code.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.AspNet.SignalR.Client;

class Program
{
    static void Main(string[] args)
    {
        var login = "admin";
        var tenant = "Tenant";
        var password = "123";
        // Set up a Basic authentication token
        var basicAuthToken = Convert.ToBase64String(
         Encoding.UTF8.GetBytes(login + "@" + tenant + ":" + password));
}
```

2. Connect to an instance of Acumatica ERP, as shown in the following code.

```
class Program
{
    static void Main(string[] args)
    {
        ...
```

```
        //Connect to an Acumatica ERP instance
        var connection = new HubConnection("http://localhost:8081/AcumaticaDB/");
        connection.Headers.Add("Authorization", "Basic " + basicAuthToken);
    }
}
```

3. Create a proxy to the SignalR hub, based on the name of the hub that was specified in the **Destination Name** box when the push notification was defined on the *Push Notifications* (SM302000) form in Step 1.

```
class Program
{
    static void Main(string[] args)
    {
```

```
        ...
```

```
        //Create a proxy to hub
        //Use "PushNotificationsHub" as the address of the hub
        var myHub = connection.CreateHubProxy("PushNotificationsHub");
        connection.Start().ContinueWith(task =>
        {
            if (task.IsFaulted)
            {
                Console.WriteLine(
                  "There was an error during open of the connection:{0}",
                  task.Exception.GetBaseException());
            }
            else
            {
                //Instead of "TestSignalR", specify the name
                //that you specified on the Push Notifications form
                myHub.Invoke<string>("Subscribe", "TestSignalR").Wait();
            }
        }).Wait();
    }
}
```

4. Define the class for a notification, as shown in the following code. For details on the format of the push notifications, see *Push Notifications: Format*.

```
public class NotificationResult
{
    public object[] Inserted { get; set; }
    public object[] Deleted { get; set; }
    public string Query { get; set; }
    public string CompanyId { get; set; }
    public Guid Id { get; set; }
    public long TimeStamp { get; set; }
    public Dictionary<string, object> AdditionalInfo { get; set; }
}
```

5. Implement processing of notifications. The following code displays the number of inserted and updated records specified in the notification in the console application window.

6.
```
class Program
{
    static void Main(string[] args)
    {
        ...
```

```
        //Process the notifications
        myHub.On<NotificationResult>("ReceiveNotification", nr =>
        {
            Console.WriteLine("Inserted {0}", nr.Inserted.Length);
            Console.WriteLine("Deleted {0}", nr.Deleted.Length);
        });
        Console.Read();
        connection.Stop();
    }
}
```

7. Build and test the project.

Push Notifications: To Include Additional Information in Push Notifications

The following activity shows you how to include additional information in push notifications.

> ⚠ The additional information that you add to push notifications is included in all notifications that are sent from the Acumatica ERP instance on which the additional information is configured.

Story

In Acumatica ERP, you need to add additional information to push notifications, such as the username of the user that performed the data change or the business date when the data transaction is performed.

Process Overview

In a project of your Acumatica ERP extension library, you will include the additional information in the `AdditionalInfo` element of notifications in JSON format.

System Preparation

Before you begin performing the steps of this activity, do the following:

1. Prepare an Acumatica ERP instance with any predefined dataset, as described in the *Instance Deployment: To Deploy an Instance with Demo Data* prerequisite activity.
2. Create a customization project, as specified in the *Customization Projects: To Create a Customization Project* prerequisite activity.
3. Create an extension library, as described in the *To Create an Extension Library* prerequisite activity.

> ℹ You can find the final code of this activity in the *CommitEventEnricher.cs* file in the Help-and-Training-Examples repository on GitHub.

Step 1: Including Additional Information in Push Notifications

To include additional information in push notifications, do the following:

1. In the project of your Acumatica ERP extension library, define a class that implements the `PX.Data.PushNotifications.ICommitEventEnricher` interface.
2. In the class that implements the `ICommitEventEnricher` interface, implement the `Enrich()` method of the interface. In the method, add the properties that you want to be returned in push notifications.

   > ⚠ The `Enrich()` method is called in the `PX.Data.PXTransactionScope.Dispose()` method. Therefore, the `Enrich()` method must not return data that is not accessible in this scope.

   The following code shows a sample implementation of the `ICommitEventEnricher` interface, which adds the business date and the name of the user to the `AdditionalInfo` element of notifications in JSON format.

```
using PX.Common;
using PX.Data.PushNotifications;
```

```
public class CommitEventEnricher : ICommitEventEnricher
{
    public void Enrich(IQueueEvent commitEvent)
    {
        var businessDate = PXContext.PXIdentity?.BusinessDate;
        var userName = PXContext.PXIdentity?.IdentityName;
        commitEvent.AdditionalInfo.Add(nameof(businessDate), businessDate);
        commitEvent.AdditionalInfo.Add(nameof(userName), userName);
    }
}
```

3. Build the project of your Acumatica ERP extension library.

Step 2: Testing Push Notifications with Additional Information

Run Acumatica ERP and make sure that when the results of the query change, the application includes the additional information in the `AdditionalInfo` element of the notifications in JSON format, as shown in the following notification example.

```
{
  ...
  "TimeStamp":636295833829493672,
  "AdditionalInfo":
  {
    "businessDate":"2024-05-05T15:16:23.1",
    "userName":"admin"
  }
}
```

Push Notifications: To Create a Custom Destination Type

The following activity shows you how to create a custom destination type for push notifications.

Story

Suppose that predefined types of push notification destinations that are available in Acumatica ERP (which are webhook, message queue, SignalR hub, and commerce push destination) do not satisfy the requirements of your integration application. You need to create a custom type in code.

Process Overview

In a project of your Acumatica ERP extension library, you will define a class that implements the *`PX.PushNotifications.NotificationSenders.IPushNotificationSender`* interface and a class that implements the *`PX.PushNotifications.NotificationSenders.IPushNotificationSenderFactory`* interface.

System Preparation

Before you begin performing the steps of this activity, do the following:

1. Prepare an Acumatica ERP instance with any predefined dataset, as described in the *Instance Deployment: To Deploy an Instance with Demo Data* prerequisite activity.

2. Create a customization project, as specified in the *Customization Projects: To Create a Customization Project* prerequisite activity.
3. Create an extension library, as described in the *To Create an Extension Library* prerequisite activity.

Step: Creating a Custom Destination Type

To create a custom destination type, do the following:

1. In a project of your Acumatica ERP extension library, define a class that implements the `PX.PushNotifications.NotificationSenders.IPushNotificationSender` interface, which is a sender of push notifications.
2. In the class that implements the `IPushNotificationSender` interface, implement the following methods and properties of the interface:
   - The `Address` property, which is the address to which the system should send notifications. A user specifies the value of this property in the **Address** box on the *Push Notifications* (SM302000) form. The property uses the following syntax.

     ```
     string Address { get; }
     ```

   - The `Name` property, which is the name of the notification destination. A user specifies the value of this property in the **Destination Name** box on the *Push Notifications* (SM302000) form. Use the following syntax for the property.

     ```
     string Name { get; }
     ```

   - The `Send` method, which sends a notification synchronously and uses as the parameters the notification to be sent and a cancellation token. The method uses the following syntax.

     ```
     void Send(
       NotificationResultWrapper results,
       CancellationToken cancellationToken
     );
     ```

   - The `SendAndForget` method, which sends a notification without blocking the current thread. We recommend that you use `HostingEnvironment.QueueBackgroundWorkItem` in the method implementation to delegate the execution to a parallel thread. The following code shows a sample implementation of the method.

     ```
     using System;
     using System.Threading;
     using PX.PushNotifications;
     using PX.PushNotifications.NotificationSenders;

     public void SendAndForget(
       NotificationResultWrapper result,
       CancellationToken cancellationToken,
       Action onSendingFailed,
       Action finalizer)
     {
       try
       {
         Send(result, cancellationToken);
       }
       catch (Exception e)
       {
         onSendingFailed($"Send to target {Name} failed: ({e.Message})");
       }
     ```

```
  finally
  {
    finalizer();
  }
}
```

3. Define a class that implements the `PX.PushNotifications.NotificationSenders.IPushNotificationSenderFactory` interface, which creates a sender of push notifications.
4. In the class that implements the `IPushNotificationSenderFactory` interface, implement the following methods and properties of the interface:
   - The `Create` method, which creates a sender and uses as the parameters the destination address, the name of the notification destination, and the additional parameters (such as a header for an HTTP address). Use the following syntax for the method.

     ```
     IPushNotificationSender Create(
       string address,
       string name,
       IDictionary<string, object> additionalParameters
     );
     ```

   - The `Type` property, which is a `string` identifier of the destination type that is exactly four characters long. The value of this property is stored in the database. The property uses the following syntax.

     ```
     string Type { get; }
     ```

   - The `TypeDescription` property, which is a `string` label of the destination type. A user selects the value of this property in the **Destination Type** box on the *Push Notifications* (SM302000) form. Use the following syntax for the property.

     ```
     string TypeDescription { get; }
     ```

5. Build the project of your Acumatica ERP extension library.
6. Run Acumatica ERP and test the new destination type.

Push Notifications: Inclusion in a Customization Project

If you need to transfer the configuration of push notifications to another Acumatica ERP instance, you need to include the respective push notification definition in a customization project.

> After you have included all needed items in a customization project, you export the project as a ZIP file. In the target instance, you import the file and publish this customization project. For details about importing, exporting, and publishing customization projects, see *Managing Customization Projects* and *Publishing Customization Projects*.

Push Notification Definitions in a Customization Project

You can add each push notification definition to a customization project as a *PushNotification* item, which contains the dataset of a push notification definition. A push notification definition includes the push notification destination and the data query, which defines the data changes for which Acumatica ERP sends notifications.

You use the *Push Notifications* (AU210000) page of the Customization Project Editor to manage *PushNotification* items in the customization project. This page displays the list of the push notification definitions that have been added to the customization project.

For each data query that defines the data changes for which Acumatica ERP should send push notifications in the added push notification definition or definitions, you need to do one of the following:

- If the data query has been defined with a generic inquiry, add the generic inquiry to the customization project, as described in the *Including Generic Inquiries in a Customization Project* chapter.
- If the data query has been defined with a built-in query definition, make sure that the built-in query definition is available in the customization project as a *File* item.

If a push notification included in a customization project has been changed on the *Push Notifications* (SM302000) form and you want to include these changes in the customization project, you have to update the appropriate item in the customization project by clicking **Reload from Database** on the toolbar of the *Push Notifications* (AU210000) page of the Customization Project Editor.

Push Notifications: To Include a Push Notification Definition in a Customization Project

This activity will walk you through the process of including a push notification definition in a customization project.

Story

Suppose that you need to distribute an integration application that uses Acumatica ERP push notifications to the other Acumatica ERP instances of the company. You need to include definitions for these push notifications in a customization project. Also, suppose that you have already included in this customization project all generic inquiries that are used by the required push notifications. You can then export this customization project to a ZIP file, import the file to the target instance, and publish this customization project.

Process Overview

You will include the needed push notification definitions in a customization project.

System Preparation

Before you begin performing this activity, do the following:

1. Deploy an instance of Acumatica ERP with the name *MyStoreInstance* and a tenant that has the *MyStore* name and contains the *T100* data.
2. Create a push notification by completing the following prerequisite activity: *Push Notifications: To Configure Push Notifications*.
3. Complete the following prerequisite activity: *Generic Inquiries in a Customization ProjectActivity 4.1.1: To Include Generic Inquiries in a Customization Project*. In this activity, a customization project is created, and the generic inquiry that is used in the push notification definition is added to the customization project.

Step: Including a Push Notification Definition in the Customization Project

You will include in the customization project the *MyBIIntegration* push notification definition, which was created in *Push Notifications: To Configure Push Notifications*.

To include the push notification definition in the customization project, do the following:

1. In the navigation pane of the Customization Project Editor, click **Push Notifications**. The *Push Notifications* page opens.
2. On the page toolbar, click **Add New Record**.
3. In the **Add Push Notifications** dialog box, which opens, select the check box in the row with the *MyBIIntegration* destination name.
4. Click **Save**.

   You have added the *MyBIIntegration* definition to the *Push Notifications* page.

Configuring Webhooks

In this chapter, you will learn how to configure the system to process data from an external application in Acumatica ERP with webhooks. The external application submits data to a particular URL, which is defined by the webhook. Acumatica ERP processes the webhook request from the external application and saves the data from the request.

Webhooks: General Information

A webhook helps you to integrate external applications that provide data in their own format and need to submit this data to Acumatica ERP. For example, HubSpot collects data about email clicks and can export this data in a specific format to a particular URL. By using a webhook, you can configure Acumatica ERP to process the data submitted to a particular URL and save the data in Acumatica ERP.

Learning Objectives

In this chapter, you will learn how to do the following:

- Create a webhook handler that will process the webhook requests from the external application
- Register the webhook on the *Webhooks* (SM304000) form of Acumatica ERP
- Include the webhook handler and the registered webhook in a customization project

Applicable Scenarios

You configure the processing of webhooks in Acumatica ERP if you need to implement an integration with an external application that provides data in its own format, and if this format is not compatible with the web services APIs of Acumatica ERP.

Configuration of Webhooks

To configure webhooks, you perform the following general steps:

1. You create a webhook handler that processes the requests from the external application.
2. You register a webhook that is supported with the webhook handler on the *Webhooks* (SM304000) form of Acumatica ERP.
3. You copy the URL that is generated during the registration of the webhook handler, and then specify this URL in the external application so that it sends requests to this URL.
4. You test the processing of the requests.
5. Optional: You include the webhook implementation in a customization project.

Webhook Handler

A webhook handler is a custom class that processes the requests passed to a particular URL. This class must implement the `PX.Api.Webhooks.IWebhookHandler` interface.

The `PX.Api.Webhooks.IWebhookHandler` interface is available in the `PX.Api.Webhooks.Abstractions.dll` assembly.

The `IWebhookHandler` interface has one method with the following signature.

```
Task HandleAsync(WebhookContext context, CancellationToken cancellation);
```

In the `HandleAsync` method, you perform the following general steps:

1. You process authentication information in the request.
2. You transform the data in the external format to the data that can be saved in Acumatica ERP.
3. You invoke graph methods that save the data in Acumatica ERP.

You can write unit tests for your webhook handler. In a test, you can override only the `virtual` properties and methods of `Webhook*` classes that are relevant to the test. For more information about unit tests, see *Unit Test Framework Guide*.

Registration of the Webhook

After you have created a webhook handler and placed the DLL of the class in the `Bin` folder of your Acumatica ERP instance, you need to register the webhook handler on the *Webhooks* (SM304000) form. In the **Webhook Name** box, you enter the name of the webhook. In the **Implementation Class** box, you enter the name of the webhook handler that you have created.

After the new webhook is saved on the form, the **URL** box contains the URL that can be used by an external application to send data to Acumatica ERP.

Webhook requests are summed with other API requests in the Acumatica ERP license restrictions.

Preparation of the External Application

For an external application to send requests to Acumatica ERP, you need to prepare the external application by specifying in it the URL generated on the *Webhooks* (SM304000) form.

You also need to implement requests to Acumatica ERP that satisfy the following requirements:

- The request type must be `POST` or `GET`.
- The body must contain only data that can be transferred with the HTTP protocol.
- The body of the request must be no longer than the value specified by the `webhook:maxrequestsize` key of the `web.config` file of the Acumatica ERP instance. By default, this value is 1 MB. You can change the default value by specifying a different value in the key of the `web.config` file.

If an error occurs during the processing of a webhook, the error is returned in JSON format, no matter which `Accept` header is submitted in the request.

Request Log

On the **Request History** tab of the *Webhooks* (SM304000) form, you can remove requests from the log and check the statuses of the processing of requests. You can also specify the type and amount of requests to be stored in the log.

In the log, the system stores the request body as a string. If the system can determine the encoding of the request body, the system reads the body as a string in this encoding. (The system determines the encoding based on the `charset` parameter in the `Content-Type` header. If there is no `charset` parameter, the system tries to use the default encoding for the media type in `Content-Type`.)

You can limit the length of the body of each request that is stored in the history by using the `webhook:maxbodysizetolog` key of the `web.config` file of the Acumatica ERP instance. By default, the length is 10 KB. The system trims the part of the body that exceeds the specified length.

Webhooks in a Customization Project

If the webhook must be used in multiple Acumatica ERP instances, you can include the webhook in a customization project and publish this project to the needed Acumatica ERP instances. For details about working with customization projects, see *Managing Customization Projects*.

To include the webhook in a customization project, you need to include the following project items:

- *Webhook*, which includes the webhook registered on the *Webhooks* (SM304000) form. You use the *Webhooks* (AU210020) page of the Customization Project Editor to manage webhooks in the customization project.
- *File*, which includes the DLL file with code of the webhook handler. You use the *Custom Files* (AU202500) page of the Customization Project Editor to manage files.

If you want to make the implementation class of the webhook unavailable for editing in the instance where the customization project is published, select the **Predefined** check box, and click **Save**. In this instance, an administrative user can still make the webhook inactive and modify the request history settings.

Webhooks: To Configure Webhooks

The following activity shows you how to configure webhooks to save the time tracking information from Toggl in Acumatica ERP.

Story

Suppose that you need to register employee time activities. For that purpose, you plan to use the Toggl application, which can track employees' time. To send data about tracked time periods from Toggl to Acumatica ERP, you need to configure a webhook in Acumatica ERP.

Process Overview

You will define a webhook handler that implements the `PX.Api.Webhooks.IWebhookHandler` interface. You will then register in Acumatica ERP a webhook that will use this webhook handler. You will include the registered webhook and the DLL file with the code of the webhook handler in a customization project.

The webhook will receive data from the Zapier application, which will transfer data from Toggl to Acumatica ERP. You will prepare both the Toggl application and the Zapier application.

System Preparation

Before you begin performing the steps of this activity, do the following:

1. Prepare an Acumatica ERP instance with the *U100* dataset, as described in the *Instance Deployment: To Deploy an Instance with Demo Data* prerequisite activity.

   

   You need the *U100* dataset because the activity uses the *gibbs* user, which has been defined in this dataset.

2. Create a new customization project, as specified in the *Customization Projects: To Create a Customization Project* prerequisite activity.

3. Create an extension library, as described in the *To Create an Extension Library* prerequisite activity.

> You can find the final code and customization project of this activity in the *Help-and-Training-Examples* repository on GitHub.

Step 1: Preparing Toggl

Prepare the Toggl application as follows:

1. On *https://toggl.com/*, sign up for an account, and sign in to that account.
2. Create a time entry, which you will use later to test the connection. In the entry, specify its description.
3. On your profile page, copy the API token, which is located at the bottom of the page.

Step 2: Creating a Webhook Handler

To create a webhook handler, do the following:

1. In the Visual Studio project of the extension library, add references to the following libraries from the `Bin` folder of the Acumatica ERP instance:
   - `PX.Api.Webhooks.Abstractions.dll`
   - `Microsoft.AspNetCore.Http.Abstractions.dll`
   - `Microsoft.Extensions.Primitives.dll`
   - `Microsoft.Net.Http.Headers.dll`
   - `Newtonsoft.Json.dll`
2. In the project, define the `TogglWebhookHandler` class, which implements the `PX.Api.Webhooks.IWebhookHandler` interface, as shown in the following code.

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.Net.Http.Headers;
using Newtonsoft.Json;
using PX.Api.Webhooks;
using PX.Data;
using PX.Objects.CR;
using PX.Objects.PM;

namespace TogglWebhook
{
    public class TogglWebhookHandler : IWebhookHandler
    {
        private static readonly JsonSerializer Serializer = new JsonSerializer();
        public async Task HandleAsync(WebhookContext context,
            CancellationToken cancellation)
        {
            if (!context.Request.Headers.TryGetValue(HeaderNames.Authorization,
                out var authValue))
            {
                context.Response.StatusCode = StatusCodes.Status401Unauthorized;
                return;
            }
```

```
            if (authValue != "Bearer token")
            {
                context.Response.StatusCode = StatusCodes.Status403Forbidden;
                return;
            }

            using (var scope = GetAdminScope())
            {
                using (var jr = new JsonTextReader(
                    context.Request.CreateTextReader()))
                {
                    var payload = Serializer.Deserialize<
                        Dictionary<String, Object>>(jr);
                    if (payload == null)
                    {
                        context.Response.StatusCode = StatusCodes.Status400BadRequest;
                        return;
                    }
                    var graph = PXGraph.CreateInstance<TimeEntry>();
                    var ta = graph.Items.Insert(new PMTimeActivity()
                    {
                        Date = Convert.ToDateTime(payload["at"]).ToLocalTime(),
                        TimeSpent = Convert.ToInt32(payload["duration"]),
                        OwnerID = PXAccess.GetContactID(),
                        Summary = "Test time entry"
                    });
                    graph.Items.Cache.SetValueExt<PMTimeActivity.projectID>(ta, "X");
                    graph.Items.Cache.SetValueExt(ta, "NoteText",
                        "Created from Toggl");
                    graph.Actions.PressSave();
                    using (var w = context.Response.CreateTextWriter())
                    {
                        Serializer.Serialize(w, new
                        {
                            echo = payload,
                            ts = DateTimeOffset.Now
                        });
                    }
                }
            }
        }
        private IDisposable GetAdminScope()
        {
            //Specify the name of your tenant with the U100 dataset after @.
            var userName = "gibbs@01_U100";
            return new PXLoginScope(userName);
        }
    }
}
```

In the code above, you have implemented the `HandleAsync` method of the `IWebhookHandler` interface. In the method, you have validated the `Authorization` header, created an instance of the `TimeEntry` graph, parsed the contents of the request, inserted the parsed data in the cache, and saved your changes to the database.

> For simplicity, in the code above, the `Authorization` header is expected to be `Bearer token`. You need to implement an actual authorization validation in the production code.

3. Build the project. The DLL file of the extension library is available in the `Bin` folder of your Acumatica ERP instance.

Step 3: Registering the Webhook

To register the webhook, do the following:

1. In Acumatica ERP, open the *Webhooks* (SM304000) form.
2. In the **Webhook Name** box, enter the name of the webhook: `TogglWebhook`.
3. In the **Implementation Class** box, select the name of the webhook handler, which is *TogglWebhook.TogglWebhookHandler*.
4. On the **Request History** tab, select *All* in the **Requests to Keep** box. You make the system save all webhook requests for testing purposes.
5. On the page toolbar, click **Save**.

   In the **URL** box, the generated URL appears. An example of this URL is shown below.

```
https://example.acumatica.com/2023R1/Webhooks/01_U100/3d5c2d34-26ed-48bf-9879-cd7f52163c0a
```

Step 4: Saving the Webhook to a Customization Project

To simplify the deployment of a created webhook in another environment, add the created webhook and the DLL file of the extension library to the customization project by doing the following:

1. Open the customization project of the extension library with the webhook handler in the Customization Project Editor. For details, see *To Open a Project*.
2. Add the *TogglWebhook* webhook to the customization project. For details about adding to a customization project, see *Webhooks in a Customization Project*.
3. Add the DLL file of the extension library as a *File* item to the customization project as specified in *To Add a Custom File to a Project*.

Step 5: Preparing the External Application

Because the Toggl application cannot interact using HTTP, you will use the Zapier application to connect the Toggl application and Acumatica ERP. To prepare the Zapier application, do the following:

1. On *https://zapier.com*, sign up for an account, and sign in to that account.
2. Create a new zap based on the following template: *https://zapier.com/app/editor/template/90071*.

   > If the template is not available, create a new zap that connects the Toggl app with Acumatica ERP.

   In the new zap, specify the Toggl account that you created in Step 1, set up the trigger, and test it. Make sure that the test time entry that you created in Step 1 is displayed.
3. Specify the request settings and test the request as follows:

   a. Add the URL that you generated in Step 3 to configure the webhook request to be sent to Acumatica ERP.

b. Add the `Authorization` header with the `Bearer token` value. (You use this value to pass the validation of the `Authorization` header that you defined in the `TogglWebhookHandler` webhook handler.)

c. Make sure that the test request is completed successfully.

4. Publish the created zap and turn it on.

Step 6: Testing the Webhook

To test the webhook that you have created, do the following:

1. On the Toggl website, create a time entry and specify its description.

   The information about the time entry is sent to Acumatica ERP via Zapier.

2. Open the *Webhooks* (SM304000) form. On the **Request History** tab, find the latest *POST* request.

3. To view the body of the request, in the table, select the request, and on the table toolbar, click **Show Request Details**.

   The request contains the body in JSON format with the time entry details, as shown in the following screenshot.

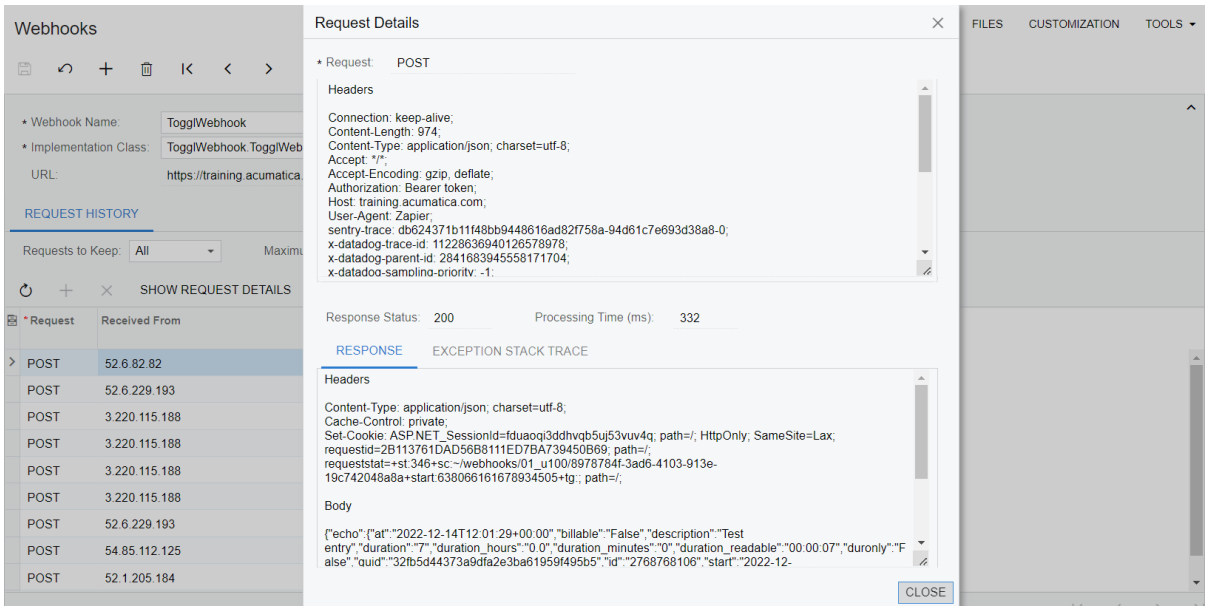

Figure: Request details

Working with the SOAP API

In this chapter, you will find information about Acumatica ERP screen-based SOAP API.

Working with the Screen-Based SOAP API

The screen-based SOAP API of Acumatica ERP provides the SOAP interface of the Acumatica ERP contract-based web services through which external systems can get data records from Acumatica ERP, process these records, and save new or updated records to Acumatica ERP.

Screen-Based Web Services API

The screen-based web services API is a part of Acumatica ERP integration services, which provides integration with external data sources and third-party systems by using a SOAP interface.

External applications and systems that use the Acumatica ERP web services API can access the data managed by Acumatica ERP and the business functionality of Acumatica ERP. For example, you can integrate Acumatica ERP with eCommerce or an online store system, so that an external system pushes all information about customers, sales orders, and payments to Acumatica ERP, and Acumatica ERP provides information on the availability of stock items and processes all incoming data.

The screen-based web services API works with Acumatica ERP forms. That is, it provides API objects and methods for working with elements on Acumatica ERP forms.

To upload data to and from Acumatica ERP by using the screen-based web services API, you define the sequence of commands for the system to work with elements on an Acumatica ERP form. This sequence of commands reflects the sequence of actions to be executed for a data record as if the record is being manipulated by a user through an Acumatica ERP form. That is, when you enter data into the system manually, you perform a sequence of actions. You open the needed data entry form and start entering data. As you add a new record, you use the UI elements one by one—you type text, select values from combo boxes, clear or select check boxes, and click buttons. In the sequence of commands for the web services, you compose exactly the same sequence of actions by specifying a command for each user action on the form. For more information on the commands you can use, see *Working with Commands of the Screen-Based SOAP API*.

This sequence of commands is similar to the sequence of commands you configure when you create import and export scenarios. You can find more information on import and export scenarios in *Configuring Import Scenarios* and *Configuring Export Scenarios*.

To use screen-based web services API in your application, you should generate the WSDL file of the web service, as described in *To Generate the WSDL File of the Web Services*, and import this file to your development project as described in *To Import the WSDL File Into the Development Environment*. After that, you can start developing your application. You can find the description of the API methods in *Screen-Based SOAP API Reference*.

Related Links

- *API Objects Related to Acumatica ERP Forms*
- *Screen-Based SOAP API Reference*

API Objects Related to Acumatica ERP Forms

The main object, which provides access to all other objects and methods of the Acumatica ERP web services API, is a `Screen` object. By using the methods of a `Screen` object, you can log in to Acumatica ERP and retrieve, insert, update, and delete data. You can also use the methods to perform any actions that are exposed by Acumatica ERP forms available through the web service.

Screen Object

After you have logged in to Acumatica ERP by using the web services API, you can access data on Acumatica ERP forms available through the web service. The `Screen` class provides the same set of API methods for working with all Acumatica ERP forms available through the service. You can find out which form a method accesses by noting the prefix in the name of the method, which is the form ID. For example, the `Export()` method that you use to export data from the *Stock Items* (IN202500) form is `IN202500Export()`.

Content Object

To get the description of the structure (schema) of a form, you should use the `GetSchema()` method of the `Screen` object. This method is specific for each Acumatica ERP form, and you should use the method with the ID of the needed form in the prefix of the method name. The method returns the schema of the form as the corresponding `Content` object, which is specific for each form. For example, to get the schema of the *Stock Items* form, you should call the `IN202500GetSchema()` method of the `Screen` object. You will receive the result as a `IN202500Content` object, as the following code shows.

```
Screen context = new Screen();
...
IN202500Content stockItemsSchema = context.IN202500GetSchema();
```

Command Object

By using subobjects of a `Content` object, you configure the sequence of commands that should be executed during data import, data export, or data processing through the web service. You configure the sequence of commands inside an array of objects of the `Command` type. When you are reflecting the selection of an element on a form in the sequence of commands, you have to select the object of the Acumatica ERP form whose element you want to access. The objects that are available inside a `Content` object have names that are similar to the names of the UI elements that you see on the form and have the ID of the form as a prefix.

For example, to be able to select the **Item Class** element, which is located in the **Item Defaults** group on the **General** tab of the *Stock Items* form (shown in the following screenshot), you would select the `GeneralSettingsItemDefaults` property of the `IN202500Content` object. This property provides access to the `IN202500GeneralSettingsItemDefaults` object, which includes the `ItemClass` property.

Each element on an Acumatica ERP form (such as a text box, combo box, or table column) is associated with a particular web services API class and is available through corresponding property of this class. The property has a similar name to that of the corresponding box on the form, such as `ItemClass`, as the following screenshot shows.

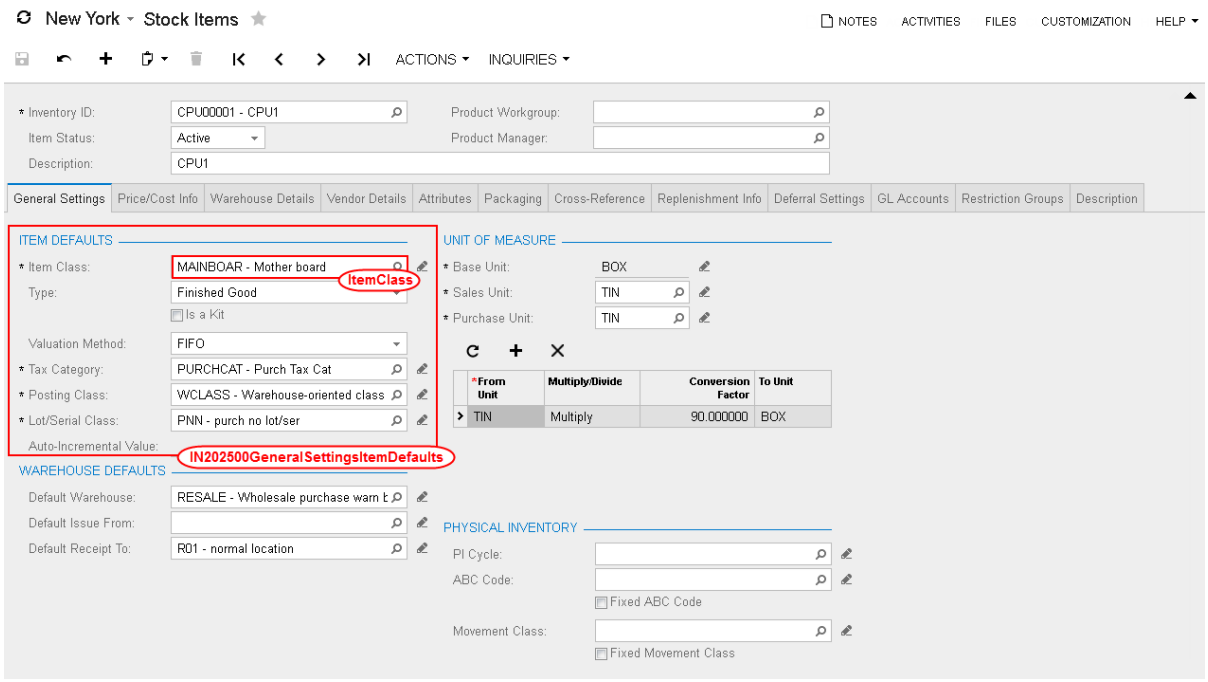

Figure: API object on an Acumatica ERP form

The following code shows an example of configuring a list of commands for the *Stock Items* form inside an array of `Command` objects.

```
//stockItemsSchema is a IN202500Content object
var commands = new Command[]
{
    stockItemsSchema.StockItemSummary.ServiceCommands.EveryInventoryID,
    stockItemsSchema.StockItemSummary.InventoryID,
    stockItemsSchema.StockItemSummary.Description,
    stockItemsSchema.GeneralSettingsItemDefaults.ItemClass,
    stockItemsSchema.GeneralSettingsUnitOfMeasureBaseUnit.BaseUnit
};
```

For more information on the commands, see *Working with Commands of the Screen-Based SOAP API*.

Actions Object

To insert an action to a sequence of commands, such as clicking a button, you use the corresponding property of the special API object `Actions` (which has a prefix with the ID of the form in the name). The `Actions` object is available through the `Actions` property of the `Content` object that corresponds to the form. The properties of the `Actions` object have names that are similar to the names of corresponding buttons on the form, such as `Delete`.

You can view the classes available through the web services API by using Object Browser in Visual Studio.

Related Links

- *Working with Commands of the Screen-Based SOAP API*

Screen-Based API Wrapper

Because of the connection of the screen-based SOAP API with Acumatica ERP forms, the applications that are developed based on this API are sensitive to the UI changes in the system. That is, any changes made to the UI after the application is created require the application to be updated and recompiled. If you want your application to not depend on the UI changes in the system, you should use the screen-based API wrapper, which is described in this topic.

How a Client Application Based on the Screen-Based API Works

A client application that uses the screen-based web services API includes the WSDL description of the service, which contains the API elements that the application can use to work with the service. The API elements have names that are similar to the names of the elements on the UI of Acumatica ERP. For example, the **Customer ID** element of the *Customers* (AR303000) form corresponds to the `Customer.CustomerSummary.CustomerID` property.

When the application calls the `Screen.GetSchema()` method, it retrieves from Acumatica ERP the schema of an Acumatica ERP form. The schema of the form is a `Content` object, which defines the correspondence between the API elements and the internal data fields that are used for operations with data by Acumatica ERP. If something has been changed in the schema of an Acumatica ERP form, the `Content` object that is returned by the `Screen.GetSchema()` method contains a different correspondence between the API elements and internal data fields than the correspondence for which the application is compiled, and the application fails.

For example, suppose that a client application requests the customer ID by the `Customer.CustomerSummary.CustomerID` property. Suppose also that in an update of Acumatica ERP, the **Customer ID** element of the *Customers* form was renamed to **Customer**, and therefore it should be requested by using the `Customer.CustomerSummary.Customer` property from the API. The client application requests the customer ID by using the `Customer.CustomerSummary.CustomerID` property and fails.

What the Screen-Based API Wrapper Is

The screen-based API wrapper is a special wrapper designed to prevent the UI changes in the system from causing application failure. The wrapper works with any changes in the schema of an Acumatica ERP form. That is, the wrapper makes the application work regardless of the changes in the names of UI elements and the changes in the internal names of data fields and objects.

The wrapper is distributed as the `PX.Soap.dll` file, which is installed automatically during Acumatica ERP installation. You can find the `PX.Soap.dll` file in the `ScreenBasedAPIWrapper` folder of your Acumatica ERP installation folder.

The `PX.Soap` library, which the wrapper provides, includes the `Helper.GetSchema()` method, which you should use instead of the `Screen.GetSchema()` method of the screen-based web services API. For information on how to use the screen-based API wrapper, see *To Use the Screen-Based API Wrapper*.

How the Screen-Based API Wrapper Works

When the client application is executed for the first time and requests the schema of an Acumatica ERP form by using the `Helper.GetSchema()` method, the wrapper requests the schema from the Acumatica ERP screen-based web service by using the `Screen.GetSchema()` method of the screen-based API. The web service interacts with the Acumatica ERP import and export engine and returns the current schema of the form. The wrapper saves the schema in an XML file and returns the schema to the client application as a `Content` object. The client application uses this schema to work with Acumatica ERP.

> Instead of the `Helper.GetSchema()` method you can use the `Helper.ReuseStoredSchema()` method to upload a schema that was saved earlier by the wrapper.

The following diagram illustrates the way the screen-based API wrapper works during the first execution of a client application.

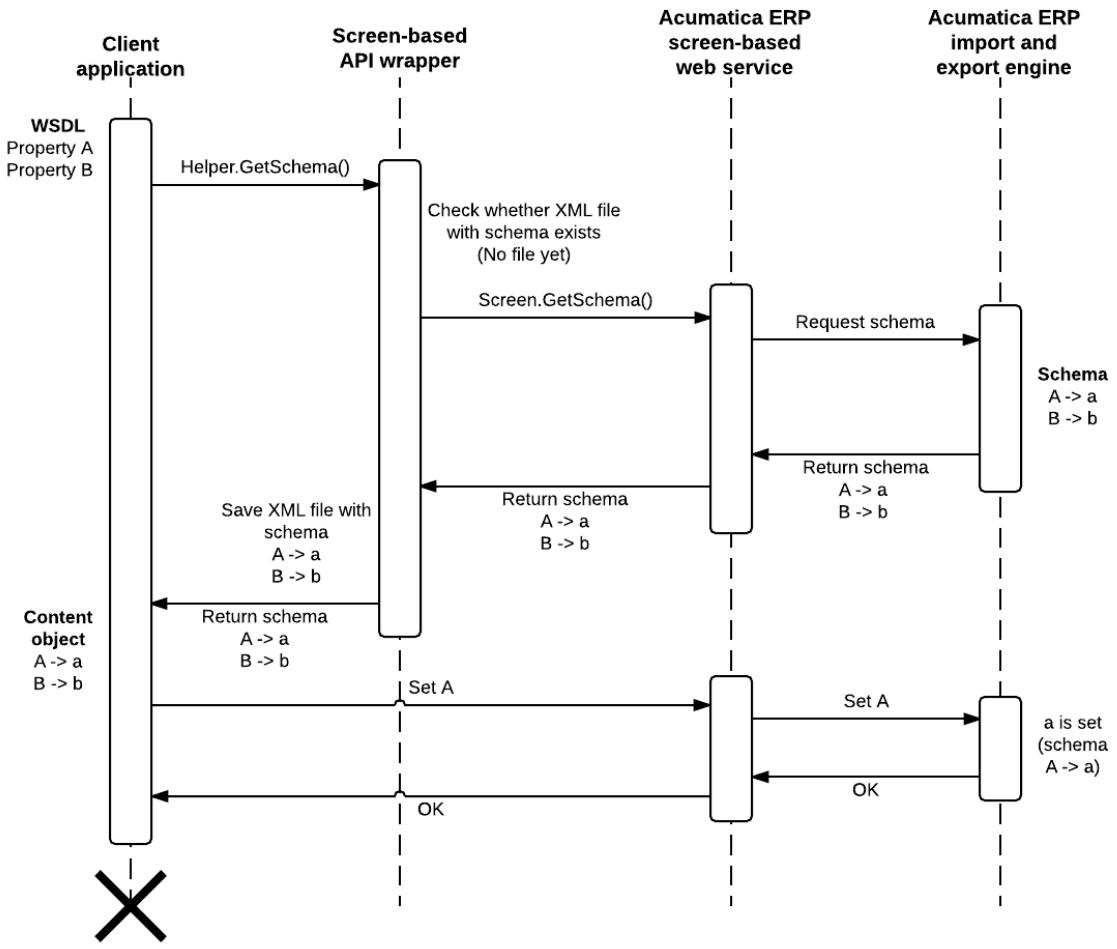

Figure: First execution of an application

When the client application is executed for the second time and all subsequent times and it requests the schema of an Acumatica ERP form by using the `Helper.GetSchema()` method, the wrapper retrieves the XML schema that is saved locally and submits this schema to the Acumatica ERP screen-based web service by using the `Screen.SetSchema()` method of the screen-based API. The web service interacts with the Acumatica ERP import and export engine, which replaces the current schema of the form that is stored on the server with the one that was saved locally. The wrapper returns the schema that was saved locally to the client application. The client application uses the local schema to work with Acumatica ERP.

> The schema that is submitted to Acumatica ERP by using the screen-based API wrapper is used on the server during the current session and is discarded after the end of the session.

The following diagram illustrates the way the screen-based API wrapper works during the second execution and all subsequent executions of a client application.

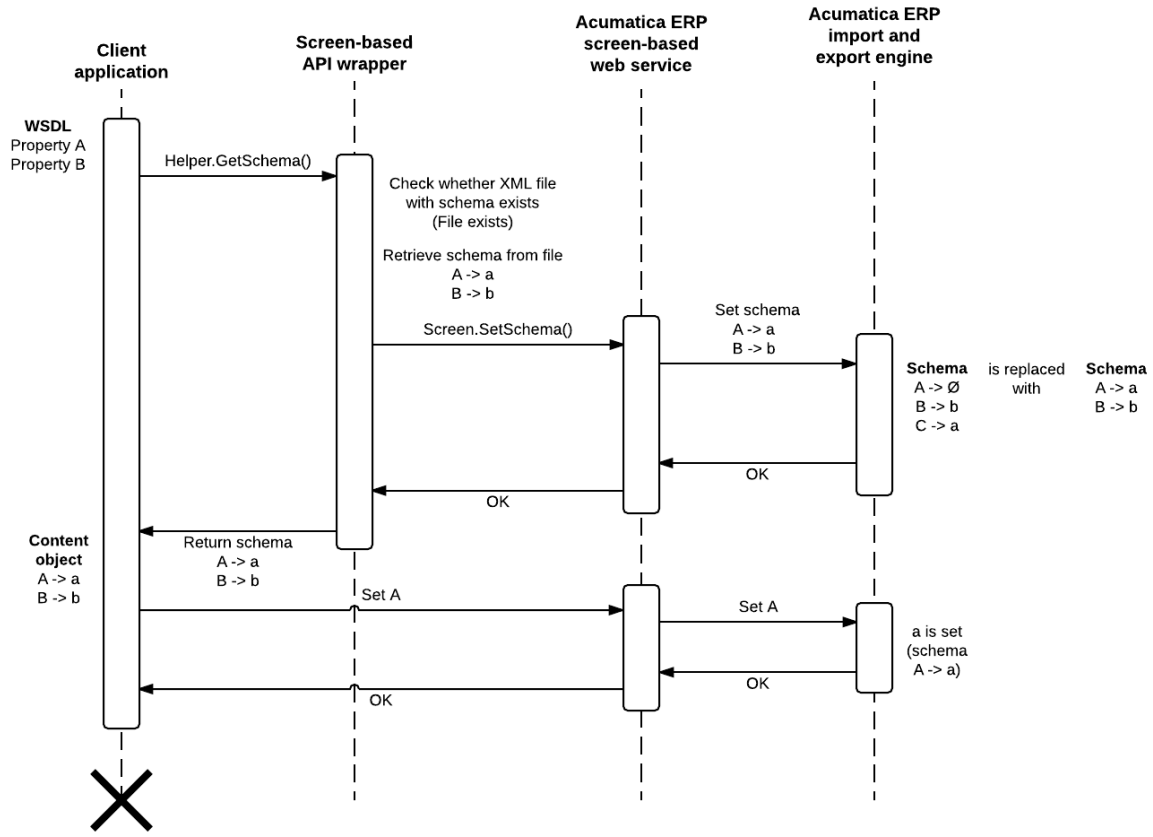

Figure: Subsequent executions of the application

> You should distribute the XML file with the schema along with your client application. If the wrapper has not found the XML file, it requests the new schema.
>
> The name of the XML file with the schema contains a hash code that depends on the WSDL description. If you update the WSDL description in your application, the wrapper works in the same way as it did during the first execution of the application and creates a new XML file with the schema.

Related Links

- *To Use the Screen-Based API Wrapper*

To Generate the WSDL File of the Web Services

The WSDL description of the Acumatica ERP screen-based web services API contains the descriptions of the API objects and methods that you can use to access Acumatica ERP forms.

Because of the connection of the API with Acumatica ERP forms, each generated WSDL description of this API reflects the current state of the system. That is, the WSDL description does not include any changes made to the system after the WSDL file was generated, and each time you change the system, you should regenerate the WSDL description and update your application accordingly.

> ⚠ You can prevent breaking changes in your application and omit regeneration of the WSDL description for each change in the system by using the screen-based API wrapper. For details on how to use the wrapper, see *To Use the Screen-Based API Wrapper*.

For example, suppose that you have generated a WSDL description of the web service that contains the definition of the `CustomerID` property, which corresponds to the **Customer ID** element on the *Customers* (AR303000) form. Further, suppose that you changed the name of the **Customer ID** element to **Customer Identifier** in a customization project. To access the **Customer Identifier** element on the form through the screen-based web services API, you need to regenerate the WSDL description so that it contains the definition of the `CustomerIdentifier` property (which corresponds to the **Customer Identifier** element) and update your application accordingly.

> ⚠ For any container (that is, a form, tab, grid, tree, or panel), element, or action with the **#** or **%** title, the generated WSDL file contains `NUMBER` instead of the `#` symbol, and `PERCENT` instead of the `%` symbol.

You can generate the WSDL file of an Acumatica ERP web service in one of the following ways.

To Generate a WSDL File for One Acumatica ERP Form

If your application needs to work with only one Acumatica ERP form, you can generate the web service that provides access to only this form.

To generate a WSDL file for a form, on the title bar of this form, click **Tools > Web Service** on the UI. This opens the page that contains the description of the web service, with the following URL: `http(s)://<ApplicationPath>/Soap/<FormID>.asmx`. In this URL, `<ApplicationPath>` is replaced with the actual URL of your application, and `<FormID>` is replaced with the ID of the form. For example, suppose that you generated a web service for the *Customers* form of the *WebServiceAPITest* application, which is accessed under a secure connection and is running on a local computer. The URL of this service is `https://localhost/WebServiceAPITest/Soap/AR303000.asmx`.

To Generate a WSDL File for Multiple Acumatica ERP Forms

If your application needs to work with multiple Acumatica ERP forms, you can generate one web service that provides access to all needed forms.

To generate a WSDL file for multiple forms, on the *Web Services* (SM207040) form, you should do the following:

1. Type the ID of the web service in the **Service ID** box of the Summary area of the form. This ID will be used in the URL of the generated service.
2. Select the Acumatica ERP forms that your application needs to use on the left pane of the form, and click **Add to Grid** for each form.
3. Select the types of integration these forms should provide (which can include importing data, exporting data, and submitting data) by selecting the appropriate check boxes (any combination of **Import**, **Export**, or **Submit**) for corresponding rows on the right pane of the form.
4. Click **Generate**.

> ⚠ The functional areas that expose the selected forms should be properly configured and the forms should open in a web browser. If a form could not be opened in a web browser, the web service definition will not be generated for this form.

After the web service is generated, you can view the WSDL description by clicking **View Generated** on the form toolbar. This opens the page that contains the description of the web service with the following URL: `http(s)://<ApplicationPath>/Soap/<ServiceID>.asmx`. In this URL, `<ApplicationPath>` is replaced with

the actual URL of your application, and `<ServiceID>` is replaced with the ID of the service that you specified in the **Service ID** box when configuring the service. For example, suppose that you generated a web service for multiple forms with the `MYSTORE` service ID for the *WebServiceAPITest* application, which is accessed under a secure connection and is running on a local computer. The URL of this service is `https://localhost/WebServiceAPITest/Soap/MYSTORE.asmx`.

To Import the WSDL File Into the Development Environment

When the WSDL file is generated, you must import it into your development environment to generate proxy classes. If necessary, see the documentation of your development environment to find out the correct way of building the proxy classes based on the WSDL definition.

The procedure below is described based on an example of the creation of a console project with the *MyBIIntegrationSBAPI* name that uses the *MYBIINTEGRATION* web service. You can create another type of Visual Studio project with another name and use any other web service define on the *Web Services* (SM207040) form in this procedure.

In this topic, you will find instructions on how to implement the proxy classes by using Visual Studio 2019.

Importing the WSDL File into a Visual Studio Project

To import the WSDL description into your development environment, proceed as follows:

1. Open Visual Studio, and create a new Visual C# console application with the *MyBIIntegrationSBAPI* name and the *.NET Framework 4.8* target framework.

   > ⚠ To create a console application, click **File > New > Project** on the main menu of Visual Studio. In the **Create a new project** dialog box, which appears, select the *Console App (.Net Framework)* C# template and click **Next**. In the **Configure your new project** dialog box, specify the name and location of the solution, select the *.NET Framework 4.8* framework version, and click **Create**.

   The MyBIIntegration application that you have created contains the `Program.cs` file with the `Program` class.

2. Add to the project a reference to the Acumatica ERP web service as follows:
   a. On the main menu of Visual Studio, click **Project > Add Service Reference**.
   b. In the **Add Service Reference** dialog box, which appears, click **Advanced**.
   c. In the **Service Reference Settings** dialog box, which appears, click **Add Web Reference**.
   d. In the **URL** box of the **Add Web Reference** dialog box, which appears, specify the URL of the *MYBIINTEGRATION* web service (see item 1 in the following screenshot).

      > ⚠ To copy the URL of the service, on the *Web Services* (SM207040) form, select *MYBIINTEGRATION* in the **Service ID** box, click **Generate** and then **View Generated** on the form toolbar, and copy the URL from the address line in the browser for the page that opens. In this example, the URL is *http://localhost/MyStoreInstance/Soap/MYBIINTEGRATION.asmx*.

   e. Click **Go** (2) to make Visual Studio connect to the web service.

      > ⚠ If your Acumatica ERP website uses a self-signed certificate, Visual Studio displays security alert windows with warnings on the certificate. Click **Yes** in these windows to proceed.

f. In the **Web reference name** box, type `MyBIIntegration` (3). This name will be used as a namespace for the web service classes generated by Visual Studio based on the WSDL description of the service.

g. Click **Add Reference** (4) to close the dialog box and add to the project the reference to the specified service.

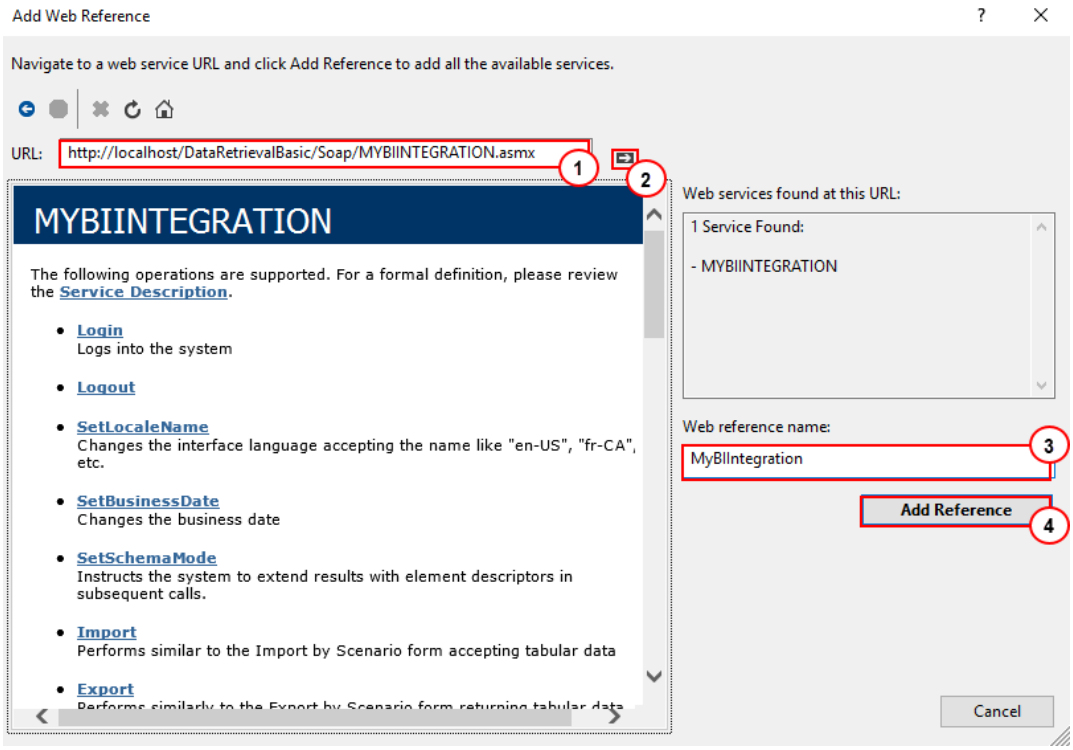

Figure: Add Web Reference dialog box

Visual Studio adds to the project the `Web References` project folder with the *MyBIIntegration* web service reference in it, as shown in the following screenshot.

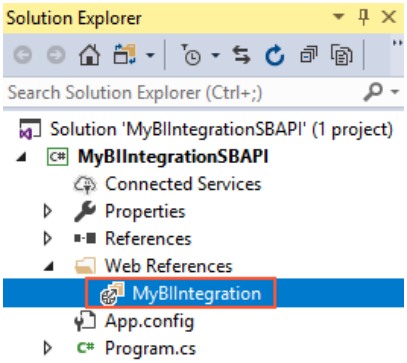

Figure: Solution Explorer

3. Rebuild the project.

To Use the Screen-Based API Wrapper

The screen-based SOAP API depends on the user interface of Acumatica ERP forms. That is, each generated WSDL description of this API includes the API elements that correspond to the current names of UI elements of the

system. Therefore, if any changes have been made to the user interface of the system after the WSDL file was generated, the WSDL description will not include these changes, and the client application that uses this WSDL will fail when it works with the system.

To prevent application failures and omit the regeneration of the WSDL description for each change of the user interface of the system, you can use the screen-based API wrapper, as described in this topic. You can find more information on the screen-based API wrapper in *Screen-Based API Wrapper*.

To Use the Screen-Based Web Services API Wrapper in a Client Application

1. Add to your project a reference to `PX.Soap.dll`.

   > ⚠ The `PX.Soap.dll` file is installed automatically during Acumatica ERP installation. You can find this file in the `ScreenBasedAPIWrapper` folder of your Acumatica ERP installation folder.

2. Use the `Helper.GetSchema()` method from the `PX.Soap` library in the code of your application instead of the corresponding `Screen.GetSchema()` method to obtain the schema of each form. For example, the following code retrieves the schema of the *Customers* (AR303000) form by using the screen-based API wrapper.

```
Screen context = new Screen();
context.CookieContainer = new System.Net.CookieContainer();
context.Url = Properties.Settings.Default.MyStoreIntegration_MyStore_Screen;
context.Login
(
  Properties.Settings.Default.Login,
  Properties.Settings.Default.Password
);

AR303000Content custSchema =
  PX.Soap.Helper.GetSchema<AR303000Content>(context);
```

3. Work with the retrieved `Content` object by using the standard screen-based web services API methods.

Related Links

- *Screen-Based API Wrapper*
- *To Update a Client Application that Uses Screen-Based Web Services*

To Update a Client Application that Uses Screen-Based Web Services

Acumatica ERP is the client application that uses screen-based web services. Before you update your Acumatica ERP instance, we recommend that you follow the procedure described in this topic. It can help to prevent application failures and omit the regeneration of the WSDL description for each change of the user interface of the system.

To Update a Client Application that Uses Screen-Based Web Services

To prevent application failures with the update to a newer version of Acumatica ERP, which is the client application using screen-based web services, perform the following steps before you install the update:

1. Create a test copy of your production Acumatica ERP instance.
2. Make changes to your client application, as described in *To Use the Screen-Based API Wrapper*.

3. Test the client application with the test copy of your Acumatica ERP instance.
4. Update the test copy of your Acumatica ERP instance to a new version of Acumatica ERP.
5. Test the client application with the updated test copy of your Acumatica ERP instance.
6. Update the production instance of Acumatica ERP to the new version.

You should distribute the client application along with the XML schema file that is generated by the screen-based API wrapper. For details, see *Screen-Based API Wrapper*.

Working with Commands of the Screen-Based SOAP API

To upload data to and from Acumatica ERP by using the screen-based SOAP API, you should define the sequence of commands for the system as it works with elements on an Acumatica ERP form. This sequence of commands reflects the sequence of actions to be executed for a data record as if the record is being manipulated by a user through an Acumatica ERP form.

Commands for Retrieving the Values of Elements

You configure the sequence of commands that should be executed during data import, data export, or data processing through the web service by using an array of objects of the `Command` type.

As you specify this sequence of commands, when you need to reflect obtaining the value of an element on a form, you should use a `Field` object. To specify the element whose value you need to obtain, you can do one of the following:

- Select the needed `Field` object from the subobjects of the `Field` type of a `Content` object that corresponds to the needed Acumatica ERP form
- Create an object of the `Field` type and specify its properties

Both of these ways are described in detail in the following sections of this topic.

Selection of the Fields Available in a Content Object

If you want to obtain the value of an element on an Acumatica ERP form, you can select the needed `Field` object from the subobjects of the `Field` type of the `Content` object that corresponds to the form. For example, if you want to export the values of the **Inventory ID** and **Description** elements on the *Stock Items* (IN202500) form, you can use the following code.

```
//stockItemsSchema is an IN202500Content object
var commands = new Command[]
{
   ...
   stockItemsSchema.StockItemSummary.InventoryID,
   stockItemsSchema.StockItemSummary.Description,
   ...
};
```

Field Object Creation

You can retrieve the values of not only the fields that are available on the Acumatica ERP form, but also the data fields of the data access classes (DACs) underlying the form. Some of these data fields are not available directly

through the elements of the form. That is, you cannot select the needed `Field` object among the subobjects of the `Content` object.

If you want to retrieve the value of an element that is not available on the form, you can create a `Field` object and specify its properties so that it specifies the needed data field of the DAC. You should specify the name of the object that corresponds to the needed DAC as the `ObjectName` and type the name of the data field in `FieldName`. The following code illustrates the creation of a `Field` object for the `LastModifiedDateTime` data field (which specifies the date and time when a record was modified) that is available through the DAC underlying the `StockItemSummary` object of the *Stock Items* form.

To find the names of the data fields that belong to DACs, you should read the applicable Acumatica ERP code. You can find the source code of Acumatica ERP on the *Source Code* (SM204570) form or in <ApplicationFolder>\App_Data\CodeRepository\PX.Objects\, where <ApplicationFolder> is replaced with the path to the folder of the Acumatica ERP application instance. You can learn the details of working with the source code of Acumatica ERP in the T300 Acumatica Customization Platform training course.

```
//stockItemsSchema is an IN202500Content object
var commands = new Command[]
{
    ...
    new Field
    {
         ObjectName = stockItemsSchema.StockItemSummary.InventoryID.ObjectName,
         FieldName = "LastModifiedDateTime"
    },
    ...
};
```

You can create `Field` objects for the elements that are available on a form. If you want to create a `Field` object for an element available on the form, set the `ObjectName` property to the `ObjectName` property of the needed subobject of the `Field` type of a `Content` object, and set the `FieldName` property to the `FieldName` property of this `Field` object. The following code illustrates the creation of a `Field` object for the `InventoryID` field of the *Stock Items* form.

```
new Field
{
     ObjectName = stockItemsSchema.StockItemSummary.InventoryID.ObjectName,
     FieldName = stockItemsSchema.StockItemSummary.InventoryID.FieldName
}
```

Selection of a Group of Records for Export

Each record in the system is identified by the values of the key elements on the applicable Acumatica ERP form. For example, a record on the *Stock Items* (IN202500) form is identified by the value of the **Inventory ID** key element. Key elements are available through the Summary object of a form. You can use the key element or elements of the form to select all records of the form for export. The other way to specify a group of records for export is to use the elements of the Summary area of the form in custom filters. Both ways of selecting records are described in detail below.

Export of All Records from a Form

If you want to export every record from an Acumatica ERP form, in the array of `Command` objects that you pass to the `Export()` method, you should insert the service command of the `EveryValue` type for the corresponding key element on the form. The `EveryValue` service command is available through the `ServiceCommands` subobject of the Summary object of the `Content` object that corresponds to the form. This service command specifies that every record of the specific type should be processed during export.

For example, if you want to export all stock item records available in the system, you should insert the `EveryInventoryID` service command, as the following code shows.

```
//stockItemsSchema is an IN202500Content object
var commands = new Command[]
{
   stockItemsSchema.StockItemSummary.ServiceCommands.EveryInventoryID,
   ...
};
```

Export of a Group of Records from a Form

You can filter the data available in the Acumatica ERP database to select the records for export. For example, you can configure the system to export only the records that have a particular status.
To define a filter for the data being exported, you should create an array of `Filter` objects and add the needed filters to it. To define a filter, you should specify:

- The UI element whose value should be used for filtering (in the `Field` property of the `Filter` object).
- The value or values with which the value of the element should be compared (in the `Value` property or `Value` and `Value2` properties of the `Filter` object).
- The condition of comparison (in the `Condition` property of the `Filter` object).

If you define multiple filters in the array, you should also specify the logical operator (either `And` or `Or`) that defines how to apply these filters. If filters are passed to an `Export()` method, during export, the system selects only the records that conform to the specified condition (or conditions) and exports only these records.

For example, suppose that you want to export only the stock item records that have the *Active* status. In this case, you can specify the filtering condition that the **Item Status** element should be equal to *Active*, as the following code shows. During export, the system processes the records that match the filtering conditions and exports only the records with the *Active* status.

```
var filters = new Filter[]
{
    new Filter
    {
         Field = stockItemsSchema.StockItemSummary.ItemStatus,
         Condition = FilterCondition.Equals,
         Value = "Active",
    }
};
```

For filtering records, you can use either the data fields of the Summary object of the form from which you are exporting data, or the data fields of the data access class (DAC) underlying the Summary object. (In Acumatica Framework, this DAC is called the main DAC of the primary data view.) If a data field of the DAC is not available directly through the elements of the Summary object, you cannot select the needed `Field` object among the subobjects of a `Content` object, as the previous code example shows for the `ItemStatus` data field. Instead, you should create a new `Field` object and specify its properties as follows: Specify the name of the Summary

object as `ObjectName` (that is, the object name that corresponds to the object to which the key data field belongs), and type the name of the data field as `FieldName` in the code directly.

The following example shows filtering by the `LastModifiedDateTime` data field (which specifies the date and time when a record was modified) that is available through the DAC underlying the `StockItemSummary` object of the *Stock Items* form.

```
new Filter
{
    Field = new Field
    {
         ObjectName = stockItemsSchema.StockItemSummary.InventoryID.ObjectName,
         FieldName = "LastModifiedDateTime"
    },
    Condition = FilterCondition.Greater,
    Value = DateTime.Now.AddMonths(-1).ToLongDateString()
}
```

Commands for Setting the Values of Elements

As you specify the sequence of commands in an array of `Command` objects, when you need to specify the value of an element on a form, you should use `Value` commands.

To set the value of an element on a form, you should do the following:

1. Create a `Value` object.
2. Specify the value of the element on the form in the `Value` property of the created `Value` object.
3. Specify the element on the form whose value should be set by using the `LinkedCommand` property of the `Value` object.

The following code illustrates setting the value of the **Customer Name** element on the *Customers* (AR303000) form.

```
//custSchema is an AR303000Content object
var commands = new Command[]
{
    ...
    new Value
    {
        Value = "John Good",
        LinkedCommand = custSchema.CustomerSummary.CustomerName
    },
    ...
}
```

Commands for Clicking Buttons on a Form

As you specify the sequence of commands in an array of `Command` objects, when you need to reflect the clicking of a button on a form (such as clicking **Save**, **Delete**, or **Release** to perform the action on a document), you should use the corresponding `Action` command. Actions are available for all buttons on the form.

In an array of `Command` objects, to use an `Action` command, you have to select the needed action in the `Actions` subobject of the `Content` object that corresponds to the form. Actions have names that are similar to the names of the buttons on the form.

The following code reflects the clicking of the **Save** button on the *Customers* (AR303000) form in a command.

```
//custSchema is an AR303000Content object
var commands = new Command[]
{
    ...
    custSchema.Actions.Save,
    ...
};
```

Commands for Adding Detail Lines

When you need to add a detail line to an Acumatica ERP form, you can use one of the following approaches:

- Add detail lines one by one on the Details tab of the form. For example, on the *Sales Orders* (SO301000) form, you can click **Add Row** on the **Details** tab and specify the values of the elements of each detail line.
- Add detail lines by using a pop-up panel. For example, on the *Shipments* (SO302000) form, you can use the **Add Sales Order** pop-up panel, which is opened when you click **Add Order** on the table toolbar of the **Details** tab.

In this topic, you will find the description of the `NewRow` command, which imitates the first approach listed above in the screen-based API. The second approach is described in *Commands for Pop-Up Panels*.

NewRow Service Command

When you are specifying the sequence of commands in an array of `Command` objects for a processing method and you need to add a new detail line to a document, you should use commands as follows:

1. To add a new row, use the `NewRow` service command, which is an available service command of the Details subobject of a `Content` object.
2. To specify the values of the elements of the created row, use the `Value` commands corresponding to the elements.

The following code shows an example of an order line being added to a sales order.

```
//orderSchema is an SO301000Content object
var commands = new Command[]
{
    ...
    orderSchema.DocumentDetails.ServiceCommands.NewRow,
    new Value
    {
        Value = "AALEGO500",
        LinkedCommand = orderSchema.DocumentDetails.InventoryID
    },
    new Value
    {
        Value = "10.0",
        LinkedCommand = orderSchema.DocumentDetails.Quantity
    },
    new Value
    {
        Value = firstItemUOM,
        LinkedCommand = orderSchema.DocumentDetails.UOM
    },
```

```
    ...
}
```

Commands for Pop-Up Dialog Boxes and Pop-Up Forms

In this topic, you will learn how to enter data to pop-up dialog boxes and pop-up forms by using the screen-based SOAP API.

Pop-Up Dialog Boxes

When you update specific fields on some forms under certain circumstances, the system displays pop-up dialog boxes where you need to respond to a question (by clicking a button) in order to proceed. For example, when you update the **Customer Class** value on the *Customers* (AR303000) form for an existing customer, the system displays a **Warning** dialog box with the text *Please confirm if you want to update current customer settings with the customer class defaults. Otherwise, original settings will be preserved.* and the **Yes** and **No** buttons. You should click **Yes** to proceed with changing the customer class.

When you are specifying the sequence of commands in an array of `Command` objects for a processing method and you need to specify an answer to a question that would appear in a pop-up dialog box if the data was being entered manually, you should create a `Value` command and set its properties as follows:

- In the `Value` property, specify the answer that you select in the dialog box during manual entry of a record.
- In the `LinkedCommand` property, use a `DialogAnswer` service command, which is available through the `ServiceCommands` subobject of an object that invokes the appearance of the pop-up dialog box.

You should insert this `Value` command directly before the field that causes the appearance of the dialog box.

The following code shows how you would update the customer class in an existing customer record on the *Customers* form.

```
//custSchema is an AR303000Content object
var commands = new Command[]
{
    ...
    new Value
    {
        Value = "Yes",
        LinkedCommand = custSchema.CustomerSummary.ServiceCommands.DialogAnswer
    },
    new Value
    {
        Value = "INTL",
        LinkedCommand = custSchema.GeneralInfoFinancialSettings.CustomerClass
    },
    ...
};
```

Pop-Up Forms

When you click specific buttons on some forms, the system opens a pop-up window with another Acumatica ERP form where you can specify or edit the values of elements as needed. For example, if you click **Add Contact** on the **Contacts** tab of the *Customers* form, the system displays the *Contacts* (CR302000) form.

> ⚠ Do not confuse a situation when the system opens a pop-up window that contains an Acumatica ERP form with a situation when the system opens a pop-up panel (where you can specify needed settings but no Acumatica ERP form is shown). A pop-up window that contains a form has an address line in the browser where you can see the ID of the form. A pop-up panel looks like a dialog box and does not have an address line.

When you are specifying a sequence of commands in an array of `Command` objects for a processing method and you need to reflect the setting of values of elements of a pop-up form in these commands, you should perform the following steps:

1. Call an action that invokes a pop-up form as follows:
   a. By using the `GetSchema()` method of the `PX.Soap.Helper` class, get the `Content` object that corresponds to the form that invokes a pop-up form.
   b. Specify the command that invokes a pop-up form in the sequence of commands by using the corresponding `Action` command.
   c. Submit this sequence of commands to the form that invokes the pop-up form by using the corresponding `Submit()` method.
2. Specify the values on the pop-up form as follows:
   a. By using the `GetSchema()` method of the `PX.Soap.Helper` class, get the `Content` object that corresponds to the form that appears as a pop-up.
   b. Specify the list of commands that specifies the values of needed elements of the pop-up form.
   c. Add the `Save` action to the list of commands.
   d. Submit this sequence of commands to the pop-up form by using the corresponding `Submit()` method.

The following code illustrates the setting of the values of the *Contacts* form, which appears as a pop-up after the user clicks **Add Contact** on the **Contacts** tab of the *Customers* form.

```
//context is a Screen object
//custSchema is an AR303000Content object
var commands = new Command[]
{
    new Value
    {
        Value = customerID,
        LinkedCommand = custSchema.CustomerSummary.CustomerID
    },

    custSchema.Actions.NewContact
};
context.AR303000Submit(commands);

//contSchema is a CR302000Content object
commands = new Command[]
{
    new Value
    {
        Value = "Green",
        LinkedCommand = contSchema.DetailsSummary.LastName
    },
    contSchema.Actions.Save,
};
context.CR302000Submit(commands);
```

Commands for Pop-Up Panels

In some instances, when you click a specific button on some form, the system opens a pop-up panel, where you can set the values of needed elements. This pop-up panel looks like a dialog box and does not contain an Acumatica ERP form. For example, if you click **Add Order** on the table toolbar of the **Details** tab of the *Shipments* (SO302000) form, the system displays the **Add Sales Order** pop-up panel.

Do not confuse a situation when the system opens a pop-up panel (where you can specify needed settings but no Acumatica ERP form is shown) with a situation when the system opens a pop-up window that contains an Acumatica ERP form. A pop-up window that contains a form has an address line in the browser where you can see the ID of the form. A pop-up panel looks like a dialog box and does not have an address line.

When you are specifying a sequence of commands in an array of `Command` objects for a processing method and you need to reflect the setting of values of elements on a pop-up panel in these commands, you should perform the following steps:

1. Insert the `DialogAnswer` service command of the pop-up panel object before the action that opens the panel, and set the `Commit` property to `true` for this command.
2. Specify the action that opens the pop-up panel, and set the `Commit` property of the action to `true`.
3. Specify the values of elements as needed on the pop-up panel.
4. Specify the action that closes the panel, and set the `Commit` property of the action to `true`.

For some pop-up panels, you need to specify only one action to select values from the pop-up panel. For example, you need to specify only one action if you are creating a shipment by using the **Create Shipment** action on the *Sales Orders* (SO301000) form, which displays the **Specify Shipment Parameters** pop-up panel.

The following code illustrates the setting of the values on the **Add Sales Order** pop-up panel, which appears after a user clicks **Add Order** on the toolbar of the **Details** tab of the *Shipments* form.

```
//shipmentSchema is a SO302000Content object
//Force a commit for the SelectSO action
var selectSOwithCommit = shipmentSchema.Actions.SelectSO;
selectSOwithCommit.Commit = true;

//Force a commit for the AddSO action
var addSOwithCommit = shipmentSchema.Actions.AddSO;
addSOwithCommit.Commit = true;

//Configure the list of commands
var commands = new Command[]
{
  ...
  //Open the Add Sales Order panel
  new Value
  {
    Value = "OK",
    LinkedCommand =
     shipmentSchema.AddSalesOrderOperation.ServiceCommands.DialogAnswer,
    Commit = true
  },
```

```
  selectSOwithCommit,

  //Specify the first order number on the Add Sales Order panel
  //and get the values of item elements
  new Value
  {
    Value = firstOrderNbr,
    LinkedCommand = shipmentSchema.AddSalesOrderOperation.OrderNbr
  },
  new Value
  {
    Value = "True",
    LinkedCommand = shipmentSchema.AddSalesOrder.Selected
  },
  shipmentSchema.AddSalesOrder.InventoryID,
  shipmentSchema.AddSalesOrder.Quantity,
  shipmentSchema.AddSalesOrder.OpenQty,
  shipmentSchema.AddSalesOrder.LineDescription,
};
//context is a Screen object
//Submit the commands to the form
var soLines = context.SO302000Submit(commands);

//Select all items of the first order for shipment
List<Command> commandList = new List<Command>();
for (int index = 0; index < soLines.Length; index++)
{
  commandList.Add(new Value
  {
    Value = index.ToString(),
    LinkedCommand = shipmentSchema.AddSalesOrder.ServiceCommands.RowNumber
  });
  commandList.Add(new Value
  {
    Value = "True",
    LinkedCommand = shipmentSchema.AddSalesOrder.Selected,
    Commit = index < soLines.Length - 1
  });
}

//Add items to the shipment
commandList.Add(addSOwithCommit);
context.SO302000Submit(commandList.ToArray());
```

Commands for Record Searching: Filter Service Command

The system uses the key element or elements on a form to find records that belong to different documents. For example, on the *Invoices and Memos* (AR301000) form, there are two key elements: **Type** and **Reference Nbr**.

If you know the values of the key element or elements of the needed record, you can select this record for update by specifying the key values in the sequence of commands that you pass to the processing method of the web services API. In the sequence of commands, you should first specify key element or elements to identify the record that you are going to update. After you have specified the values of key element or elements, you should specify the values of other elements in the order in which you would specify them on the form.

If you do not know the values of the key element or elements of the needed record, you can update records in the system by searching for them using their unique field values that you know. For example, you can identify customers by email addresses or phone numbers. To search for a record, you have to imitate the use of a column of a **Select** dialog box, declare a custom key, or declare a custom field in the sequence of commands that you pass to a processing method. In this topic, you will find a detailed description of the `Filter` service command, which imitates the use of a selector column. You can find a description of two other approaches in *Commands for Record Searching: Key Command* and *Commands for Record Searching: Custom Field*.

Filter Service Command

Selector columns on an Acumatica ERP form appear when a user clicks the Magnifier icon of the key element of the form to bring up the **Select** dialog box. Service commands for selector columns have the `Filter` prefix in their names. For example, to search for a customer record, you can use the `FilterCity`, `FilterCountry`, `FilterEmail`, and `FilterPhone1` service commands.

To use a column of a **Select** dialog box for a search, you have to do the following:

1. Create a `Field` object, and initialize its properties with the values of the properties of the key field.
2. Concatenate the `FieldName` property of this object (which is now equal to the value of the `FieldName` property of the key field) with `!` and the `FieldName` property of the needed `Filter` service command.
3. In the `Value` command in the array of `Command` objects, set the `Value` property to the value that should be used for the search and the `LinkedCommand` property to the created `Field` object.

For example, the following code searches for a customer record by email address.

```
//custSchema is an AR303000Content object
Field customerIDSelector = custSchema.CustomerSummary.CustomerID;
customerIDSelector.FieldName += "!" +
        custSchema.CustomerSummary.ServiceCommands.FilterEmail.FieldName;

var commands = new Command[]
{
    new Value
    {
        Value = "demo@gmail.com",
        LinkedCommand = customerIDSelector
    },
    ...
};
```

If you need to get the value of the field that was used for a search as a result of the processing, you should assign an initial `FieldName` to the field before getting the value. For example, the following code shows how to get the value of the Customer ID element after you have modified the corresponding field for the search.

```
//custSchema is an AR303000Content object
//Save the initial field name of the CustomerID field
string initialCustomerIDFieldName =
custSchema.CustomerSummary.CustomerID.FieldName;

//Configure the command that searches for a customer record
//by using the FilterEmail service command
Field customerIDSelector = custSchema.CustomerSummary.CustomerID;
customerIDSelector.FieldName += "!" +
custSchema.CustomerSummary.ServiceCommands.FilterEmail.FieldName;

//Configure the list of commands
var commands = new Command[]
{
    //Search for the needed customer record
    new Value
    {
         Value = customerMainContactEmail,
         LinkedCommand = customerIDSelector
    },

    //Do the needed modifications and save changes on the form
    ...
};

//context is a Screen object
//Submit commands to the form
context.AR303000Submit(commands);

//Assign an initial field name to the CustomerID field
custSchema.CustomerSummary.CustomerID.FieldName = initialCustomerIDFieldName;

//Get the customer ID
commands = new Command[]
{
    custSchema.CustomerSummary.CustomerID
};

//Submit commands to the form
AR303000Content customer = context.AR303000Submit(commands)[0];
```

Commands for Record Searching: Key Command

To search for a record, you have to imitate the use of a column of a **Select** dialog box, declare a custom key, or declare a custom field in the sequence of commands that you pass to a processing method. In this topic, you will find a detailed description of the use of the `Key` command, which you use to declare a custom key. You can find descriptions of two other approaches in *Commands for Record Searching: Filter Service Command* and *Commands for Record Searching: Custom Field*.

Key Command

If you specify the value of key elements on an Acumatica ERP form, the system does not change the current record in the system; instead, it searches for the record, which is identified by the values of the key elements, and selects this record. By using custom keys, you can make some elements on an Acumatica ERP form work as key elements.

You can specify custom keys to search for a record or a detail line. You can use the fields of the summary object and the detail objects, but not the fields that belong to other objects, as custom key fields. For example, you can find the needed customer record in the system by using the `CustomerName` field of the `AR303000CustomerSummary` object of the *Customers* (AR303000) form as the custom key, but you cannot find the record by using the `Email` field of the `AR303000GeneralInfoMainContact` object as the custom key.

To specify a custom key, you have to define the key by using the `Key` command as follows:

1. Create a `Key` command by using the operator `new`.
2. To specify the element that should be used as a custom key, set the `ObjectName` and `FieldName` properties of the created `Key` command to the values of the `ObjectName` and `FieldName` properties of the field corresponding to the element.
3. To specify the value of the custom key, set the `Value` property of the created `Key` command to the needed value in the format `='<Key value>'`, where you should replace `<Key value>` with the needed value of the key.

Below is an example of the `Key` command that declares the **Warehouse** column as the custom key on the **Document Details** tab of the *Sales Orders* (SO301000) form.

```
new Key
{
    ObjectName = orderSchema.DocumentDetails.Warehouse.ObjectName,
    FieldName = orderSchema.DocumentDetails.Warehouse.FieldName,
    Value = "='MAIN'"
},
```

Commands for Record Searching: Custom Field

To search for a record, you have to imitate the use of a column of a **Select** dialog box, declare a custom key, or declare a custom field in the sequence of commands that you pass to a processing method. In this topic, you will find a detailed description of the declaration of a custom field. You can find descriptions of two other approaches in *Commands for Record Searching: Filter Service Command* and *Commands for Record Searching: Key Command*.

Custom Field

Acumatica ERP does not include `Filter` service commands for all selector columns that are available on Acumatica ERP forms. To use the needed selector column for record searching, you can create a custom field as follows:

1. Initialize the properties of a `Field` object with the values of the properties of the key field.
2. Concatenate the `FieldName` property of this object (which is now equal to the value of the `FieldName` property of the key field) with `!` and the internal name of the selector column that you are going to use for the search. The internal name of the selector column is equal to the value of the `FieldName` property of the corresponding element on the form.
3. In the `Value` command in the array of `Command` objects, set the `Value` property to the value that should be used for the search and the `LinkedCommand` property to the created `Field` object.

For example, the following code searches for a sales order by order number.

```
//orderSchema is an SO301000Content object
var searchCustomerOrder = orderSchema.OrderSummary.OrderNbr;
searchCustomerOrder.FieldName +=
  "!" + orderSchema.OrderSummary.CustomerOrder.FieldName;

var commands = new Command[]
{
  new Value
  {
    Value = "SO",
    LinkedCommand = orderSchema.OrderSummary.OrderType
  },
  new Value
  {
    Value = "SO248-563-06",
    LinkedCommand = searchCustomerOrder
  },
  ...
}
```

Commands That Require a Commit

There are two types of elements on an Acumatica ERP form: elements with a commit, and elements without a commit. A commit is an action initiated by the form that, when triggered, sends the data to the server. On the server, the commit causes all data on the form to be updated, including the insertion of default values and the recalculation of the values of elements on the form.

A commit is a costly operation that causes the browser to make requests to the server and takes server time. As such, a commit is invoked for only a limited number of elements: mainly the key elements and the elements the other elements depend on.

In a Visual Studio project, if you specify the value of an element for which the system performs a commit by using a `Value` command, the `Commit` property of the `LinkedCommand` property (which specifies this element) is automatically set to `true`. You can check the value of the `Commit` property when you are debugging an application if you insert a breakpoint in the code after an array of commands has been configured. In the following screenshot, you can see that for the command that sets the value of the **Customer ID** element on the *Customers* (AR303000) form, the `Commit` property of the `LinkedCommand` is set to `true`. (The **Customer ID** element has the internal field name `AcctCD`.)

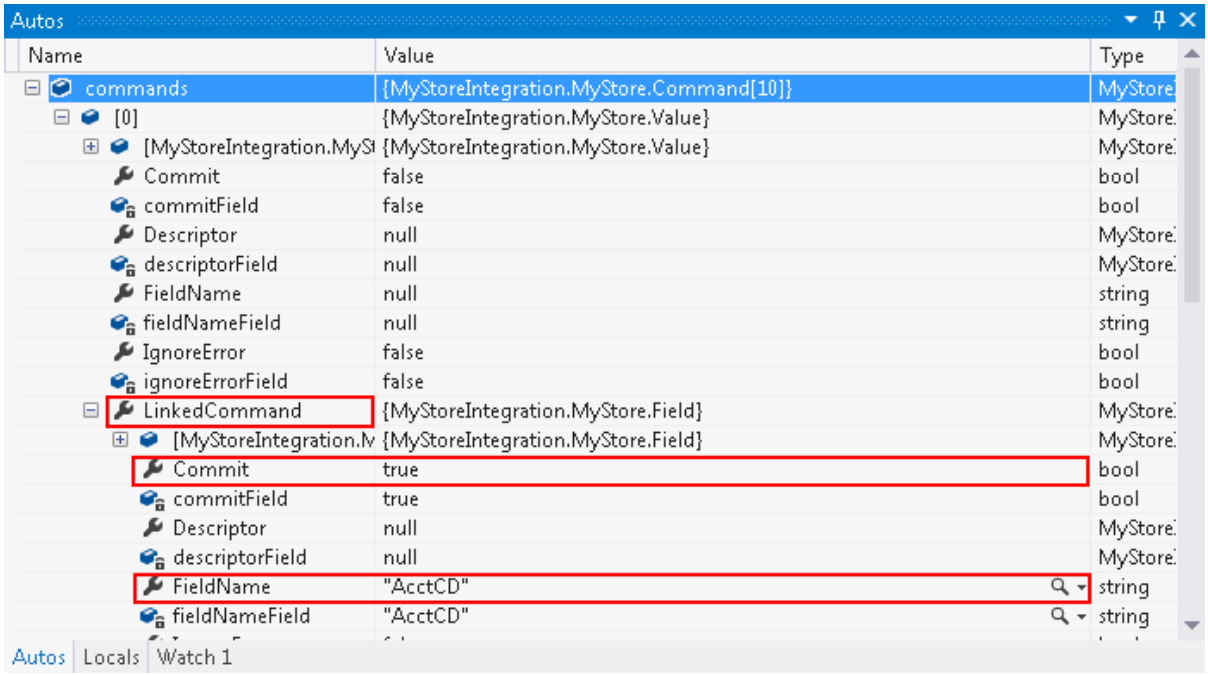

Figure: Commit property

If the `Commit` property is `false`, the element is filled in with data but no update of the form is invoked. If the `Commit` property is `true`, after the element is filled in, the commit is invoked on the server and the form is updated.

For example, when you select a customer class on the *Customers* form, the system assigns values to the elements related to customer class settings, such as **Statement Cycle ID** and **Country**. When you configure a sequence of commands for setting the values of elements on the *Customers* form, for elements such as **Customer**, the system automatically sets the `Commit` property of `LinkedCommand` to `true`.

In most cases, when you compose the sequence of commands, you can use the values of the `Commit` property that are set by default. However, you may need to force the system to invoke a commit to the server—for example, when you need to commit the value of the field before entering another field value. In such cases, you should set the `Commit` property to `true` for the command or action.

Commands for Working with Attachments

In Acumatica ERP, you can attach files to records on Acumatica ERP forms and to detail lines on master-detail forms.

To obtain a file attached to the selected record or detail line through the web services API, you specify the `Value` command in the array of `Command` objects as follows:

- In the `FieldName` property, you specify the name of the file that should be obtained.
- In the `LinkedCommand` property, you specify the needed `Attachment` service command.

  To work with a file attached to a form, you use the `Attachment` service command of the object that corresponds to the Summary object. For example, to obtain a file attached to a stock item record on the *Stock Items* (IN202500) form, you use the `Attachment` service command of the `IN202500StockItemSummary` object.

  To work with the file attached to a detail line of a master-detail form, you use the `Attachment` service command of the object that corresponds to the Details object. For example, to obtain the file attached to a warehouse detail line of a stock item on the *Stock Items* form, you use the `Attachment` service command of the `IN202500WarehouseDetails` object.

The following code shows how to retrieve the file attached to a stock item record on the *Stock Items* form.

```
//stockItemsSchema is an IN202500Content object
var commands = new Command[]
{
  new Value
  {
    Value = "AAMACHINE1",
    LinkedCommand = stockItemSchema.StockItemSummary.InventoryID
  },
  new Value
  {
    FieldName = "T2MCRO.jpg",
    LinkedCommand =
     stockItemSchema.StockItemSummary.ServiceCommands.Attachment
  }
};
//context is a Screen object
var stockItemAttachment =
 context.IN202500Export(commands, null, 1, false, true);
```

Commands for Working with Multilingual Fields

For some text boxes on Acumatica ERP forms, users can type values in multiple languages if multiple locales are configured in Acumatica ERP. For example, if your Acumatica ERP instance has the English and French locales activated and multilingual user input configured, you can specify the value of the **Description** box on the *Stock Items* (IN202500) form in English or French.

Specifying Localized Values of a Multilingual Field

When you need to specify localized values of a text box by using the screen-based SOAP API, you specify the value of the field that corresponds to the box as a string in JSON format with the localized values. In this string, you use the two-letter ISO code of the language with which the value should be associated.

In the example that is mentioned at the beginning of the topic, if you need to specify values in English and French in the **Description** box on the *Stock Items* form, you specify the value of the `Description` field of the `StockItem` entity in the following format: `[{en:English description},{fr:French description}]`, as shown in the following code fragment.

```
//stockItemSchema is an IN202500Content object
new Value
{
    Value = "[{en:Item},{fr:Pièce}]",
    LinkedCommand = stockItemSchema.StockItemSummary.Description
}
```

In the string, you should specify the actual values of the field in all languages that are configured for multilingual user input. If you specify the values of the field in particular languages, the values of the field in other languages configured for multilingual user input become empty. For example, suppose that in your instance of Acumatica ERP, multilingual fields can have values in English and French. If you pass the value of a field in the following format `[{en:English description}]`, the French value of the field becomes empty.

If you specify the value of a multilingual field as plain text, this text is saved as the value of the corresponding box in the current language of Acumatica ERP (that is, either the default language of the instance or the language that you have specified by using the `SetLocaleName()` method). For details on how to specify the locale through the screen-based SOAP API, see *SetLocaleName() Method*.

Retrieving Localized Values of a Multilingual Field

If you need to retrieve localized values of a text box that supports multiple input languages, you retrieve the value of an internal field that contains all localized values of the text box and has the *Translations* suffix in its field name.

To specify the field name and the object name of the needed internal field with localized values, you specify the field name and the object name of the multilingual text box and append *Translations* to the field name. For example, the following code shows the command that you should use to retrieve the localized values of the **Description** element of the *Stock Items* form.

```
//stockItemSchema is an IN202500Content object
new Field
{
    ObjectName = stockItemsSchema.StockItemSummary.Description.ObjectName,
    FieldName = stockItemsSchema.StockItemSummary.Description.FieldName +
      "Translations"
}
```

The returned value of a `Translations` field is a string in JSON format with the available localized values of the field. The language to which the value belongs is identified by the two-letter ISO code of the language. For example, suppose that the **Description** element of the *Stock Items* form has the value *Item* in English and *Pièce* in French. In this case, the value of the `DescrTranslations` field, which corresponds to the **Description** element, is the following string: `[{en:Item},{fr:Pièce}]`.

Related Links

- *Locales and Languages*
- *Boxes That Have Multilanguage Support*

Screen-Based SOAP API Reference

In this chapter, you will find the reference information for the main methods of the screen-based web services API; these methods are used to transfer data to and from Acumatica ERP.

This chapter covers the following objects, and the following methods, which are exposed by the `Screen` class:

- *Login() Method*
- *Logout() Method*
- *SetLocaleName() Method*
- *SetBusinessDate() Method*
- *GetScenario() Method*
- *GetSchema() Method*
- *SetSchema() Method*
- *Export() Method*
- *Submit() Method*
- *Import() Method*
- *Clear() Method*
- *GetProcessStatus() Method*

Login() Method

You use the `Login()` method to make the client application sign in to Acumatica ERP.

Instead of directly signing in to Acumatica ERP, your application can implement the OAuth 2.0 or OpenID Connect (OIDC) mechanism of authorization. For details about OAuth 2.0 and OIDC, see *Authorizing Client Applications to Work with Acumatica ERP*.

Syntax

```
public LoginResult Login(string name, string password)
```

Parameters

- `name`: The username that the application should use to sign in to Acumatica ERP, such as `"admin"`.

  To sign in to a specific Acumatica ERP tenant, specify the `name` parameter as follows: `UserName@TenantName`, where you should specify the username instead of `UserName` and the tenant name instead of `TenantName`. For example, if you sign in to the tenant with the name *Dollar* as the user with the name *admin*, you should specify the parameter as `admin@Dollar`. You can view the name that should be used for the tenant in the **Login Name** box of the *Tenants* (SM203520) form.

  To sign in to a certain branch in the tenant, specify the parameter as follows: `UserName@TenantName:BranchName`, where you should specify the username instead of `UserName`, the tenant name instead of `TenantName`, and the branch ID instead of `BranchName`. You can view the ID of the branch in the **Branch ID** box of the *Branches* (CS102000) form. For example, if you sign in to the *East* branch of the *Dollar* tenant as the user with the name *admin*, you should specify the parameter as `admin@Dollar:East`.
- `password`: The password for the username, such as `"123"`.

Return Value

The method returns the `LoginResult` object, which contains the description of errors that occurred during signing in, if any.

Example

The following code signs in to Acumatica ERP by using the parameters that are specified in the application settings.

```
Screen context = new Screen();
context.CookieContainer = new System.Net.CookieContainer();
context.Url = "https://localhost/WebServiceAPITest/Soap/MYSTORE.asmx";
context.Login("admin@MyTenant:MYSTORE", "123");
```

Usage Notes

Before you sign in to Acumatica ERP by using the `Login()` method, do the following:

1. Initialize the `CookieContainer` property of the object with a new `System.Net.CookieContainer()`. The `CookieContainer` property is a standard property

of an object of the `HttpWebClientProtocol` system type. (The `Screen` class is derived from the `HttpWebClientProtocol` class.) This property is used to maintain the session state for a client.

2. Specify the URL of the web service in the `URL` property of the object. This is the same URL that you specify when you add a web reference to the Acumatica ERP web service. You can change the URL of the service dynamically in your application if you need to switch between multiple Acumatica ERP web services.

For each call of the `Login()` method, you must call the `Logout()` method after you finish your work with Acumatica ERP to close the session. Therefore, when you are working with the web services API, we recommend that you use the pattern that is shown in the following code.

```
using
(
  //Connect to the web services and log in to Acumatica ERP
  Screen context = new Screen();
  ...
)
{
  try
  {
    //Import, export, or submit data
    ...
  }
  finally
  {
    //Log out from Acumatica ERP
    context.Logout();
  }
}
```

You should take into account Acumatica ERP license API limits when using the `Login()` method. For details, see *License Restrictions for API Users*.

Logout() Method

You use the `Logout()` method to make the client application sign out from Acumatica ERP.

Syntax

```
public void Logout()
```

Usage Notes

For each call of the `Login()` method, you must call the `Logout()` method after you finish your work with Acumatica ERP to close the session. Therefore, when you are working with the web services API, we recommend that you use the pattern that is shown in the following code.

```
using
(
  //Connect to the web services and sign in to Acumatica ERP
  Screen context = new Screen();
  ...
)
{
  try
```

```
  {
    //Import, export, or submit data
    ...
  }
  finally
  {
    //Sign out from Acumatica ERP
    context.Logout();
  }
}
```

SetLocaleName() Method

You use the `SetLocaleName()` method to specify the locale for Acumatica ERP to correctly recognize the format of dates, numbers, and other country-specific data that is passed by using the web services API. By default, Acumatica ERP uses the invariant locale, which is similar to the English (United States) locale.

Syntax

```
public void SetLocaleName(string localeName)
```

Parameter

- `localeName`: The locale that should be used in Acumatica ERP. You should specify the locale in the `System.Globalization.CultureInfo` format converted to `string`, such as `"EN-US"`.

Example

The following code shows how to specify the appropriate locale with the `SetLocaleName()` method of the `Screen` object.

```
...
using System.Threading;
...
Screen context = new Screen();
context.SetLocaleName(Thread.CurrentThread.CurrentCulture.ToString());
```

SetBusinessDate() Method

You use the `SetBusinessDate()` method to specify the business date in Acumatica ERP. You can set the business date to any date to make the system insert this date into the date fields by default. The business date is inserted into any new document that you create and is used in the default selection parameters that appear on processing and inquiry screens.

Syntax

```
public void SetBusinessDate(System.DateTime businessDate)
```

Parameter

- `businessDate`: The business date that should be used in Acumatica ERP.

Usage Notes

The business date resets to the current date of your computer after each login. Therefore, if you need to specify a business date in your application, you should call the `SetBusinessDate()` method after each client application login.

GetScenario() Method

You use the `GetScenario()` method to retrieve the list of commands of an integration scenario that is configured in the system.

Syntax

```
public Command[] GetScenario(string scenario)
```

Parameter

- `scenario`: The name of the import or export scenario for which the list of commands should be retrieved.

Return Value

The method returns the list of commands of the specified integration scenario.

GetSchema() Method

You use the `GetSchema()` method of the `Screen` object to get the description of the structure (schema) of a form. This method is specific for each Acumatica ERP form, and you should use the method with the ID of the needed form in the prefix of the method name.

To prevent application failures because of UI changes in Acumatica ERP, you can use the `GetSchema()` method of the screen-based API wrapper instead of the `GetSchema()` method of the `Screen` object. For more information on the screen-based API wrapper, see *Screen-Based API Wrapper*.

Syntax

```
public Content GetSchema()
```

Return Value

The method returns the schema of the form as the corresponding `Content` object, which is specific for each form.

Example

To get the schema of the *Stock Items* (IN202500) form, you should call the `IN202500GetSchema()` method of the `Screen` object. You will receive the result as a `IN202500Content` object, as the following code shows.

```
Screen context = new Screen();
...
IN202500Content stockItemsSchema = context.IN202500GetSchema();
```

SetSchema() Method

You use the `SetSchema()` method of the `Screen` object to change the description of the structure (schema) of a form to the one specified in the method. This method is useful when you need to work with the description of the form that was used in previous versions of Acumatica ERP. This method is specific for each Acumatica ERP form, and you should use the method with the ID of the needed form in the prefix of the method name.

Syntax

```
public void SetSchema(Content schema)
```

Parameter

- `schema`: The schema of an Acumatica ERP form.

Export() Method

You use the proper `Export()` method of a `Screen` object to export data from Acumatica ERP. You select the needed `Export()` method by using in the name of the method the prefix that contains the ID of the Acumatica ERP form from which the method exports data.

Syntax

```
public string[][] Export(
  Command[] commands,
  Filter[] filters,
  int topCount,
  bool includeHeaders,
  bool breakOnError
)
```

Parameters

- `commands`: In this parameter, you specify the UI elements of the Acumatica ERP form whose values you need to export. In the array of commands that you pass to the `commands` parameter, you can also use the `EveryValue` service command, which makes the system export all records of the specific type.
- `filters`: In this parameter, you specify any restrictions on the data to be exported. For example, you can configure the system to export only the records that have a particular status.

- `topCount`: In this parameter, you can restrict the number of records to be exported. If this parameter is set to 0, the system exports all records that are specified by the `commands` and `filters` parameters of the `Export()` method.
- `includeHeaders`: In this parameter, you specify whether the result of the export should include column headers. If this parameter is set to `true`, the result of the export includes the names of exported elements in the first row of the exported data.
- `breakOnError`: In this parameter, you specify whether the system should stop the export if an error occurs during this process. If this parameter is set to `true`, the system stops exporting data when the first error occurs during the export.

Return Value

The result of the data export is a two-dimensional string array, which represents the exported data in a table format. Thus, if an exported record contains detail lines, the values of the detail lines are translated to multiple rows of this table. The number of rows is equal to the number of detail lines of the source record. The table rows that belong to one record have the same values of the elements of the summary area specified.

For example, suppose that on the *Invoices* (SO303000) form, an invoice has three detail lines. If you export this invoice with detail lines, the data prepared for export will include three records for this invoice—one record for each detail line. These records will include identical values of the elements in the summary area of the invoice, such as the type and reference number, and different values of the detail line elements.

Submit() Method

You use the proper `Submit()` method of a `Screen` object to submit data to Acumatica ERP. You select the needed `Submit()` method by using in the name of the method the prefix that contains the ID of the Acumatica ERP form with which the method works.

Syntax

```
public Content[] Submit(Command[] commands)
```

Parameter

- `commands`: You use this parameter to specify the data that you are going to submit. In this parameter, you pass to the web service an array of `Command` objects in which you can specify commands that do the following:
  - Set the values of elements on the form by using `Value` commands.
  - Click buttons on the form (for example, the **Save** button) by using `Action` commands.
  - Get the result of data processing by using `Field` commands.

Return value

The result of processing of the data submitted by using the `Submit()` method is returned as a `Content` object specific to the form to which the data has been submitted. This object contains the values of the elements that you specified by using `Field` commands in the array of `Command` objects, which you pass to the `Submit()` method. The values of the elements that were not specified by using `Field` commands are `null`.

If you want only to submit data to an Acumatica ERP form and do not need to obtain any values of elements after submitting, do not specify any `Field` commands in the array of the `Command` object that you pass to a `Submit()` method. In this case, the `Submit()` method does not return any value.

Example 1: Submitting Data and Obtaining the Result of Processing

Suppose that you want to submit a customer record to the *Customers* (AR303000) form and obtain as a result of processing the values of the **Customer Name** and **Customer Class** elements. You pass the list of commands, which includes `Field` commands for the `CustomerName` and `CustomerClass` fields, to the `AR303000Submit()` method, as the following code shows. In this example, the `AR303000Submit()` method returns a `AR303000Content` object that has non-null values of the `CustomerName` and `CustomerClass` fields. The values of the other elements of the returned `AR303000Content` object are `null`.

```
AR303000Content custSchema = context.AR303000GetSchema();
var commands = new Command[]
{
    ...
    custSchema.Actions.Save,
    custSchema.CustomerSummary.CustomerName,
    custSchema.GeneralInfoFinancialSettings.CustomerClass
};
AR303000Content customer = context.AR303000Submit(commands)[0];
```

Example 2: Submitting Data without Obtaining the Result of Processing

Suppose that you want to create a customer record on the *Customers* form and do not need to get the values of any element on the form after the customer record is created. You pass the list of commands, which sets the needed values and saves the changes on the form (the list of commands does not include any `Field` commands), to the `AR303000Submit()` method, as the following code shows. In this example, the `AR303000Submit()` method does not return any value.

```
AR303000Content custSchema = context.AR303000GetSchema();
var commands = new Command[]
{
    new Value
    {
        ...
    },
    custSchema.Actions.Save
};
context.AR303000Submit(commands);
```

Import() Method

To import data to Acumatica ERP by using the web services API, you should use the proper `Import()` method of a `Screen` object. You select the needed `Import()` method by using in the name of the method a prefix that contains the ID of the Acumatica ERP form to which the method imports data.

Syntax

```
public ImportResults[] Import(
  Command[] commands,
  Filter[] filters,
  string[][] data,
  bool includedHeaders,
  bool breakOnError,
```

```
  bool breakOnIncorrectTarget
)
```

Parameters

- `commands`: In this parameter, you specify the UI elements of the Acumatica ERP form to which you need to import data by using `Field` commands. You can click buttons on the form (for example, the **Save** button) by using `Action` commands.
- `filters`: In this parameter, you specify any restrictions on the data to be imported. For example, you can configure the system to import only the records that have the *CUST* prefix in the customer ID field.
- `data`: In this parameter, you specify the data that should be imported in a two-dimensional string array. Each row of the array should contain the values of the fields in the order that you specified in the `commands` parameter.
- `includeHeaders`: In this parameter, you specify whether the imported data include column headers. If this parameter is set to `true`, this signals to the system that the data, which is imported, includes the names of the imported elements in the first row.
- `breakOnError`: In this parameter, you specify whether the system should stop the data import if an error occurs during this process. If this parameter is set to `true`, the system stops importing data when the first error occurs during the import.
- `breakOnIncorrectTarget`: In this parameter, you specify whether the system should stop the data import if the record does not meet the condition specified for the imported record. If this parameter is set to `true`, the system stops processing records and displays an error.

Return Value

The result of the processing of the data imported by using the `Import()` method is returned as a `ImportResults` object specific to the form to which the data has been imported. This object contains the result of the processing.

Clear() Method

You use the `Clear()` method to clear all changes on the form. The method works in the same way as the **Cancel** button on the toolbar of an Acumatica ERP form does. This method is specific to the particular Acumatica ERP form, and you should use the method with the ID of the needed form in the prefix of the method name.

Syntax

```
public void Clear()
```

GetProcessStatus() Method

You use the `GetProcessStatus()` method to monitor the status of a long-running operation (such as the release or confirmation operation).

Syntax

```
public ProcessResult GetProcessStatus()
```

Return Value

The method returns a `ProcessResult` object. You should use the `Status` property of this object to get the status of the processing operation. When the status of the operation is `Completed`, you can get the result of processing.