## FREE eBook

## LEARNING acumatica

Free unaffiliated eBook created from **Stack Overflow contributors.** 

## #acumatica

### **Table of Contents**

About
Chapter 1: Getting started with acumatica
Remarks2
Examples2
Installation or Setup
Chapter 2: Acumatica BQL Reference
Examples
BQL Parse and Verify
Parse
Verify
Conclusion 4
Chapter 3: Acumatica Platform Attributes Reference
Examples
PXFormula Attribute
General Description
Modes of Usage
PXFormulaAttribute Properties and Constructor Parameters
Usage
Order of Fields
Formula Context and Its Modifiers
Current <trecord.field> and Current2<trecord.field>8</trecord.field></trecord.field>
Parent <tparent.field></tparent.field>
IsTableEmpty <trecord>9</trecord>
Selector <keyfield, foreignoperand="">9</keyfield,>
Fetches a PXSelectorAttribute defined on the foreign key field (KeyField) of the current D9
Fetches the foreign data record currently referenced by the selector
Calculates and returns an expression on that data record as defined by ForeignOperand9
Using Formulas on Unbound Fields
List of Built-In Common Formulas

Direct and Mediated Circular References in Formulas	
Control Flow in Conditional Formulas	10
Using Multiple Formulas on One Field	10
PXRestrictor Attribute	
Introduction	
Details	11
Options	11
Overriding Inherited Restrictors	12
Global Caching	12
Recommendations for Using	
Use Restrictor Conditions Only	12
Chapter 4: Adding Attribute Support to out-of-box Sales Order Entity	14
Introduction	
Remarks	14
Examples	14
This article provides how-to guide to add Acumatica ERP Attribute support to out-of-box Sa	14
Chapter 5: Changing caption dynamically using readonly DAC fields.	17
Introduction	17
Examples	17
Ном-То	17
Chapter 6: Changing Size of Selector Drop-Down Window	20
Introduction	
Examples	
Changing default size ranges for selector drop-down window	
To expand drop-down window width of the Customer selector	
Chapter 7: Conditionally Hiding Tabs	
Introduction	
Examples	
VisibleExp Property of the PXTab Control in Aspx	
To hide Activities tab for Leads with New status	

To hide Cross-Reference tab for Stock Items that can not be sold       26         To hide Attributes tab for inactive Stock Items       26         Chapter 8: Creating Data and Time Fields in Asymptica       27
To hide Attributes tab for inactive Stock Items
Chapter 8: Creating Date and Time Fields in Asymptics
Introduction
Examples
The PX(DB)DateAndTime Attribute
The PXDBTime Attribute
The PX(DB)DateAttribute Attribute
The PXDBTimeSpan Attribute
The PXTimeList Attribute
Chapter 9: Customization Mechanisms
Examples
Using CacheAttached to Override DAC Attributes in the Graph
Replacing All Attributes
Appending a New Attribute to the DAC Field
Overriding a Single Property of an Attribute
Replacing an Attribute with Another Attribute
Application Order of the Attribute-Customizing Attributes
Chapter 10: Displaying an Error Requiring to Enter Entity Data40
Examples40
Displaying an Error Requiring the User to Enter Entity Data4
Chapter 11: Downloading Files Attached to a Detail Entity Using Contract-Based API42
Introduction
Remarks42
Examples
HTTP Cookie Header from a SOAP Response Shared by SOAP and REST Clients
Chapter 12: Exporting Records via REST Contract-Based API
Introduction 4
Remarks
Examples4

Data Export in a Single REST Call	
To export all stock items in a single REST call:	
To export all sales order of the IN type in a single REST call:	
Implementing Pagination on Multiple REST Requests	
To export stock items in batches of 10 records with multiple REST calls:	48
To export all sales orders in batches of 100 records with multiple REST calls:	
Chapter 13: Exporting Records via Screen-Based API	
Introduction	
Remarks	51
Examples	51
Data Export from an Entry Form with a Single Primary Key	51
To export all stock items in a single web service call:	
To export stock items in batches of 10 records:	
Data Export from an Entry Form with a Composite Primary Key	
To request all types of existing orders:	
To export records of each type independently in batches:	
To export records of each type independently in batches: To export records of a specific type:	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities Introduction Examples	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities Introduction Examples Adding Test Work Orders to the Related Entity Description Field.	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities Introduction Examples Adding Test Work Orders to the Related Entity Description Field Chapter 15: Filtering with multiple value with only one selector	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities Introduction Examples. Adding Test Work Orders to the Related Entity Description Field. Chapter 15: Filtering with multiple value with only one selector Introduction.	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities Introduction. Examples. Adding Test Work Orders to the Related Entity Description Field. Chapter 15: Filtering with multiple value with only one selector Introduction. Examples.	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities Introduction. Examples. Adding Test Work Orders to the Related Entity Description Field. Chapter 15: Filtering with multiple value with only one selector. Introduction. Examples. Retrieving Sales Order for multilple customer.	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities Introduction Examples Adding Test Work Orders to the Related Entity Description Field. Chapter 15: Filtering with multiple value with only one selector Introduction Examples Retrieving Sales Order for multilple customer. Chapter 16: Freight Calculation	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities Introduction Examples Adding Test Work Orders to the Related Entity Description Field. Chapter 15: Filtering with multiple value with only one selector Introduction Examples. Retrieving Sales Order for multilple customer. Chapter 16: Freight Calculation Introduction	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities. Introduction. Examples. Adding Test Work Orders to the Related Entity Description Field. Chapter 15: Filtering with multiple value with only one selector. Introduction. Examples. Retrieving Sales Order for multilple customer. Chapter 16: Freight Calculation. Introduction. Examples.	
To export records of each type independently in batches: To export records of a specific type: Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities. Introduction. Examples. Adding Test Work Orders to the Related Entity Description Field. Chapter 15: Filtering with multiple value with only one selector. Introduction. Examples. Retrieving Sales Order for multilple customer. Chapter 16: Freight Calculation. Introduction. Examples. Overriding Freight Amount in Shipment and Invoice.	

Sales Orders	
Shipments	
Overriding Freight Amount	69
Understanding implementation of the FreightCalculatorCst class in the sample above	
Chapter 17: Modifications to Base Data Views	72
Introduction	72
Examples	72
APInvoiceEntry BLC: add additional restriction to poReceiptLinesSelection data view	72
Chapter 18: Modifications to Contact and Address Info through Code	75
Introduction	75
Examples	75
Specify Contact and Address information for a new Employee	75
Override Bill-To Contact and Bill-To Address Info for a Customer	
Override Bill-To Contact and Bill-To Address Info for a Sales Order	77
Chapter 19: Modifying Items in a Dropdown List	79
Introduction	79
Remarks	79
Examples	
Modifying Marital Statuses	
To add new items to the PXStringListAttribute successor	80
To remove items declared in the PXStringListAttribute successor	81
To replace items declared in the PXStringListAttribute successor	82
Chapter 20: Populating report with data through code	
Examples	
This article covers example showing how to create report using memory records:	85
Chapter 21: Publishing skipped already applied customization content	91
Introduction	91
Examples	
Publish with cleanup from the customization screen	91
Publish with clean up from inside a customization project	92
Chapter 22: Replacing Images on the Login Page	94

Introduction	
Examples	
Using customization to replace images on the login page	94
Chapter 23: Significant API Changes Between Versions	
Examples	
PXSelectGroupBy and Bit Values in Acumatica 5.1 and 5.2+	
Acumatica Framework 5.2 and Later	
Acumatica Framework 5.1 and Earlier	
Explanation	
Chapter 24: User Interface Techniques	
Examples	101
Creating a Dropdown Menu for a Screen	101
Option 1: Creating a Dropdown Menu in ASPX	
Option 2: Creating a Menu in the Graph	102
Chapter 25: Using Customization Plug-In to Make Changes in Multiple Companies	
Introduction	104
Examples	104
Implementation of a customization plug-in to update multiple companies	
Credits	



You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: acumatica

It is an unofficial and free acumatica ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official acumatica.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

## Chapter 1: Getting started with acumatica

#### Remarks

This section provides an overview of what acumatica is, and why a developer might want to use it.

It should also mention any large subjects within acumatica, and link out to the related topics. Since the Documentation for acumatica is new, you may need to create initial versions of those related topics.

### Examples

Installation or Setup

Detailed instructions on getting acumatica set up or installed.

Read Getting started with acumatica online: https://riptutorial.com/acumatica/topic/7152/getting-started-with-acumatica

## **Chapter 2: Acumatica BQL Reference**

#### **Examples**

**BQL** Parse and Verify

Any Acumatica application developer spends a great deal of their time writing BQL code. At the same time, not everybody knows the underlying details of how BQL types work under the hood.

At the heart of BQL lay two key methods: Parse() and Verify(), declared by the IBqlCreator interface. Most of the commonly used BQL types, such as Where<>, And<>, Or<> etc., derive from this interface.

It should be admitted that the names these methods historically stuck with are not very descriptive. Arguably better alternative names for them would be PrepareCommandText and Evaluate.

### Parse

```
public void Parse(
    PXGraph graph,
    List<IBqlParameter> pars,
    List<Type> tables,
    List<Type> fields,
    List<IBqlSortColumn> sortColumns,
    StringBuilder text,
    BqlCommand.Selection selection)
```

The only purpose of Parse() is to translate BQL into an SQL command to be sent into DBMS. Therefore, this method accepts a StringBuilder parameter representing the SQL command currently being constructed, to which the BQL creator appends the SQL text representation of itself.

For example, the And<> predicate's Parse() method will append " AND " to the command text, and recursively request translation of all nested BQL creators.

In particular, And<ARRegister.docType, Equal<ARDocType.invoice>> will translate into something like "AND "ARRegister.DocType = 'AR'".



```
public void Verify(
    PXCache cache,
    object item,
    List<object> pars,
    ref bool? result,
    ref object value)
```

In contrast to Parse(), Verify() operates purely at the application level.

Given a record (e.g. an ARRegister object), it can be used to calculate expressions on it, including calculating formulas and evaluating conditions.

The result parameter is used to store the boolean condition evaluation result. It is mostly used by *predicate* BQL creators such as Where<>.

The value parameter is used to store the expression calculation result. For example, the value of a BQL Constant<string> is the string representation of that constant.

Most of the time, BQL creators will either affect the result or the value, but rarely both of them.

One notable usage of the <code>Verify()</code> method is in the static <code>BqlCommand.Meet()</code> method, used by <code>PxCache</code> to determine if a given item satisfies the BQL command:

```
public bool Meet(PXCache cache, object item, params object[] parameters)
{
    List<object> pars = new List<object>(parameters);
    bool? result = null;
    object value = null;
    try {
        Verify(cache, item, pars, ref result, ref value);
    }
    catch (SystemException ex) {
        throw new PXException(String.Format("BQL verification failed! {0}", this.ToString()),
ex);
    }
    return result == null || result == true;
}
```

## Conclusion

The real power and beauty of BQL creators lies in that most of them can be used at both the database and application level, enabling Acumatica's cache merging mechanism and providing a great opportunity for code reusability.

For instance, when you select records from the database, the Where<> clause of the BQL command:

- Will provide Parse() to translate itself into SQL text during command preparation.
- Will provide Verify() during cache merging to determine which items already residing in the cache Meet() the Where<><> clause conditions so as to include such cached items into the result set.

Read Acumatica BQL Reference online: https://riptutorial.com/acumatica/topic/9690/acumatica-bql-reference

## Chapter 3: Acumatica Platform Attributes Reference

#### **Examples**

**PXFormula Attribute** 

## **General Description**

A formula in Acumatica is a DAC field that is calculated based on the values of other object fields.

To calculate a formula, Acumatica framework provides a set of various operations and functions (such as arithmetical, logical, and comparison operations and string processing functions; see *List of Built-In Common Formulas*). In addition to the field values, a formula can use various constants provided by both the core of Acumatica and the application solutions. Moreover, a formula can obtain values for the calculation not only from the current record but also from other sources (see *Formula Context and Its Modifiers*).

The beauty of the formulas is that they will automatically recalculate the value at the right time:

- On field defaulting (inserting a new row; FieldDefaulting event handler of formula field)
- On updating of dependent fields (FieldUpdated event handler of each dependent field)
- On database selection (only for unbound fields; RowSelecting event handler)
- On database persisting if needed (developer should specify it explicitly; RowPersisted event handler)

Recalculation of a formula field value on the update of a dependent field raises a FieldUpdated event for formula field. This allows you to make a chain of dependent formulas (see Direct and Mediated Circular References in Formulas).

Application developers can write their own application-side formulas.

## Modes of Usage

A formula can be used in three main modes:

- Simply calculating the value and assigning it to formula field (see Basic Usage)
- Calculating the aggregate value from existing values of formula fields and assigning it to specified field in the parent object (see *Aggregate Usage*)
- Mixed mode: Calculating the formula value, assigning it to the formula field, calculating the aggregate value, and assigning it to the field in the parent object (see *Combined Usage*)

There is another auxiliary mode, unbound formulas, that is very similar to mixed mode, but the

calculated values of the formula are not assigned to the formula field. The aggregated value is calculated immediately and assigned to the field of the parent object. See *Usage of Unbound Formulas* for more information.

## PXFormulaAttribute Properties and Constructor Parameters

The formula functionality is implemented by PXFormulaAttribute. The constructor of PXFormulaAttribute has the following signatures:

```
public PXFormulaAttribute(Type formulaType)
{
    // ...
}
```

The single parameter formulaType is a type of formula expression to calculate the field value from other fields of the same data record. This parameter must meet one of the following conditions:

- Must implement the IBqlField interface
- Must be a BQL constant
- Must implement the IBqlCreator interface (see List of Built-In Common Formulas)

```
public PXFormulaAttribute(Type formulaType, Type aggregateType)
{
    // ...
}
```

The first parameter, formulaType, is the same as in the first constructor. The second parameter, aggregateType, is a type of aggregation formula to calculate the parent data record field from the child data record fields. An aggregation function can be used, such as SumCalc, CountCalc, MinCalc, and MaxCalc. Application developers can create their own aggregation formulas.

An aggregate formula type must be a generic type and must implement IBqlAggregateCalculator interface. The first generic parameter of the aggregate formula type must implement the IBqlField interface and must have the field type of the parent object.

```
public virtual bool Persistent { get; set; }
```

The PXFormulaAttribute.Persistent property indicates whether the attribute recalculates the formula after changes are saved to the database. You may need recalculation if the fields the formula depends on are updated on the RowPersisting event. By default, the property equals false.

## Usage

In most cases, formulas are used for direct calculation of the value of the formula field from other

fields of the same data record.

The simplest example of formula usage:

```
[PXDBDate]
[PXFormula(typeof(FADetails.receiptDate))]
[PXDefault]
[PXUIField(DisplayName = Messages.PlacedInServiceDate)]
public virtual DateTime? DepreciateFromDate { get; set; }
```

In this example, the value of the ReceiptDate field is assigned to the DepreciateFromDate field on the insertion of a new record and on the update of the ReceiptDate field.

#### A slightly more complex example:

```
[PXCurrency(typeof(APPayment.curyInfoID), typeof(APPayment.unappliedBal))]
[PXUIField(DisplayName = "Unapplied Balance", Visibility = PXUIVisibility.Visible, Enabled =
false)]
[PXFormula(typeof(Sub<APPayment.curyDocBal, APPayment.curyApplAmt>))]
public virtual Decimal? CuryUnappliedBal { get; set; }
```

Here, the unapplied balance of the document is calculated as the difference between the balance of the document and the applied amount.

Example of multiple choice with a default value:

```
[PXUIField(DisplayName = "Class Icon", IsReadOnly = true)]
[PXImage]
[PXFormula(typeof(Switch<
    Case<Where<EPActivity.classID, Equal<CRActivityClass.task>>, EPActivity.classIcon.task,
    Case<Where<EPActivity.classID, Equal<CRActivityClass.events>>,
EPActivity.classIcon.events,
    Case<Where<EPActivity.classID, Equal<CRActivityClass.email>,
    And<EPActivity.isIncome, NotEqual<True>>>, EPActivity.classIcon.email,
    Case<Where<EPActivity.classID, Equal<CRActivityClass.email>,
    And<EPActivity.isIncome, Equal<True>>>, EPActivity.classIcon.emailResponse,
    Case<Where<EPActivity.classID, Equal<CRActivityClass.history>>,
EPActivity.classIcon.history>>>>,
    Selector<Current2<EPActivity.type>, EPActivityType.imageUrl>>))]
public virtual string ClassIcon { get; set; }
```

## **Order of Fields**

The order of fields in the DAC is important to correct formula calculation. All source fields (from which the formula is calculated) including other formulas must be defined in the DAC before the formula field. Otherwise, the field can be calculated incorrectly or can cause a runtime error.

### **Formula Context and Its Modifiers**

By default, the context of the formula calculation is restricted by the current object (record) of the

class containing the formula declaration. It is also allowed to use constants (descendants of the Constant<> class).

A formula that uses the fields of its object only:

```
public partial class Contract : IBqlTable, IAttributeSupport
{
    //...
    [PXDecimal(4)]
    [PXDefault(TypeCode.Decimal, "0.0", PersistingCheck = PXPersistingCheck.Nothing)]
    [PXFormula(typeof(Add<Contract.pendingRecurring, Add<Contract.pendingRenewal,
    Contract.pendingSetup>>))]
    [PXUIField(DisplayName = "Total Pending", Enabled=false)]
    public virtual decimal? TotalPending { get; set; }
    //...
}
```

However, it is possible to obtain input values for the formula calculation from other sources:

- A current record of any cache in the BLC (if assigned).
- A foreign record specified by PXSelectorAttribute.
- A parent record specified by PXParentAttribute.

The formula supports the following context modifiers.

Current<TRecord.field> and Current2<TRecord.field>

Fetches the field value from the record stored in the current property of the TRecord cache.

If the cache's Current property or the field itself contains null:

- Current<> forces field defaulting and returns the default field value.
- Current2<> returns null.

Example:

```
[PXFormula(typeof(Switch<
    Case<Where<
    ARAdjust.adjgDocType, Equal<Current<ARPayment.docType>>,
    And<ARAdjust.adjgRefNbr, Equal<Current<ARPayment.refNbr>>>>,
    ARAdjust.classIcon.outgoing>,
    ARAdjust.classIcon.incoming>))]
protected virtual void ARAdjust_ClassIcon_CacheAttached(PXCache sender)
```

Parent<TParent.field>

Fetches the field value from the parent data record as defined by PXParentAttribute residing on the current DAC.

```
public class INTran : IBqlTable
{
    [PXParent(typeof(Select<
        INRegister,</pre>
```

```
Where<
        INRegister.docType, Equal<Current<INTran.docType>>,
        And<INRegister.refNbr,Equal<Current<INTran.refNbr>>>>))]
public virtual String RefNbr { ... }
    [PXFormula(typeof(Parent<INRegister.origModule>))]
    public virtual String OrigModule { ... }
}
```

IsTableEmpty<TRecord>

Returns true if the DB table corresponding to the specified DAC contains no records, false otherwise.

```
public class APRegister : IBqlTable
{
    [PXFormula(typeof(Switch<
        Case<Where<
            IsTableEmpty<APSetupApproval>, Equal<True>>,
            True,
            Case<Where<
                APRegister.requestApproval, Equal<True>>,
                False>>,
                True>))]
    public virtual bool? DontApprove { get; set; }
}
```

Selector<KeyField, ForeignOperand>

# Fetches a PXSelectorAttribute defined on the foreign key field (KeyField) of the current DAC.

## Fetches the foreign data record currently referenced by the selector.

## Calculates and returns an expression on that data record as defined by ForeignOperand.

```
public class APVendorPrice : IBqlTable
{
    // Note: inventory attribute is an
    // aggregate containing a PXSelectorAttribute
    // inside, which is also valid for Selector<>.
```

```
// -
[Inventory(DisplayName = "Inventory ID")]
public virtual int? InventoryID
[PXFormula(typeof(Selector<
    APVendorPrice.inventoryID,
    InventoryItem.purchaseUnit>))]
public virtual string UOM { get; set; }
```

## **Using Formulas on Unbound Fields**

If the formula field is an unbound field marked with one of the PXFieldAttribute descendants (such as PXIntAttribute or PXStringAttribute), then its calculation is additionally triggered during RowSelecting event.

## List of Built-In Common Formulas

TBD

}

## Direct and Mediated Circular References in Formulas

TBD

## **Control Flow in Conditional Formulas**

TBD

## Using Multiple Formulas on One Field

TBD

**PXRestrictor Attribute** 

## Introduction

The PXSelectorAttribute attribute (also referred to as the selector), while vital and frequently used, has however two major drawbacks:

• It gives an uninformative message "<object\_name> cannot be found in the system" if no items

are found to satisfy the selector condition.

• The produces the same error message if you update *other* fields of the record but the object referenced by the selector has already changed and no longer meets its condition. This behaviour is clearly wrong because the law must not be retroactive.

The **PXRestrictorAttribute** (also referred to as the restrictor) can be used to solve these problems.

### Details

PXRestrictorAttribute does not work alone; it should always be paired with a PXSelectorAttribute. Using the restrictor without the selector will have no effect.

The restrictor finds the selector on the same field, injecting into it an additional condition and the corresponding error message. The restrictor condition is appended to the selector condition via a boolean AND, and an appropriate error message is generated if the referenced object violates the restrictor constraint. Also, if the referenced object has changed and no longer meets the restrictor condition, no error messages are produced when you change **any other** fields of the referencing object.

#### General usage:

```
[PXDBInt]
[PXSelector(typeof(Search<FAClass.assetID, Where<FAClass.recordType,
Equal<FARecordType.classType>>>),
    typeof(FAClass.assetCD), typeof(FAClass.assetTypeID), typeof(FAClass.description),
typeof(FAClass.usefulLife),
    SubstituteKey = typeof(FAClass.assetCD),
    DescriptionField = typeof(FAClass.description), CacheGlobal = true)]
[PXRestrictor(typeof(Where<FAClass.active, Equal<True>>), Messages.InactiveFAClass,
typeof(FAClass.assetCD))]
[PXUIField(DisplayName = "Asset Class", Visibility = PXUIVisibility.Visible)]
public virtual int? ClassID { get; set; }
```

Multiple restrictors can be used with one selector attribute. In this case, all additional restrictor conditions are applied in a non-determined order. Once any condition is violated, the appropriate error message is generated.

The Where<> condition of the selector itself is applied after all restrictor conditions.

```
[PXDefault]
// An aggregate attribute containing the selector inside.
// -
[ContractTemplate(Required = true)]
[PXRestrictor(typeof(Where<ContractTemplate.status, Equal<Contract.status.active>>),
Messages.TemplateIsNotActivated, typeof(ContractTemplate.contractCD))]
[PXRestrictor(typeof(Where<ContractTemplate.effectiveFrom,
LessEqual<Current<AccessInfo.businessDate>>,
Or<ContractTemplate.effectiveFrom, IsNull>>), Messages.TemplateIsNotStarted)]
[PXRestrictor(typeof(Where<ContractTemplate.discontinueAfter,
GreaterEqual<Current<AccessInfo.businessDate>>,
Or<ContractTemplate.discontinueAfter, IsNull>>), Messages.TemplateIsExpired)]
public virtual int? TemplateID { get; set; }
```



The constructor of PXRestrictorAttribute takes three parameters:

- 1. The restrictor's additional condition. This BQL type must implement the IBqlWhere interface.
- 2. The appropriate error message. The message can contain format elements (curly brackets) to show context. The message must be a string constant defined in a localizable static class (such as PX.Objects.GL.Messages).
- 3. An array of field types. These fields must belong to the current object and must implement the <code>IBqlField</code> interface. The values of the fields will be used for error message formatting.

Also, there are several options that specify the restrictor behavior.

### **Overriding Inherited Restrictors**

The ReplaceInherited property indicates whether the current restrictor should override the inherited restrictors. If this property is set to true, then all inherited restrictors (placed on any aggregate attributes or base attribute) will be replaced.

Replacing inherited restrictors:

```
[CustomerActive(Visibility = PXUIVisibility.SelectorVisible, Filterable = true, TabOrder =
2)]
[PXRestrictor(typeof(Where<Customer.status, Equal<CR.BAccount.status.active>,
        Or<Customer.status, Equal<CR.BAccount.status.oneTime>,
        Or<Customer.status, Equal<CR.BAccount.status.hold>,
        Or<Customer.status, Equal<CR.BAccount.status.creditHold>>>>),
Messages.CustomerIsInStatus, typeof(Customer.status),
        ReplaceInherited = true)] // Replaced all restrictors from CustomerActiveAttribute
[PXUIField(DisplayName = "Customer")]
[PXDefault()]
public override int? CustomerID { get; set; }
```

Please note that we do not advise that you use the ReplaceInherited property in application code when reasonable alternatives exist. This property is primarily intended to be used in customizations.

### **Global Caching**

CacheGlobal supports global dictionary functionality in the same way as in PXSelectorAttribute.

## **Recommendations for Using**

#### **Use Restrictor Conditions Only**

When restrictors and a selector are used together, the latter should not contain the  ${\tt IBqlWhere}$ 

clause. Ideally, all conditions should be moved into restrictors. This approach provides more userfriendly error messages and eliminates unnecessary retroactive errors.

An ideal example:

```
[PXDBString(5, IsFixed = true, IsUnicode = false)]
[PXUIField(DisplayName = "Type", Required = true)]
[PXSelector(typeof(EPActivityType.type), DescriptionField =
typeof(EPActivityType.description))]
[PXRestrictor(typeof(Where<EPActivityType.active, Equal<True>>),
Messages.InactiveActivityType, typeof(EPActivityType.type))]
[PXRestrictor(typeof(Where<EPActivityType.isInternal, Equal<True>>),
Messages.ExternalActivityType, typeof(EPActivityType.type))]
public virtual string Type { get; set; }
```

#### Possible retroactive errors:

```
[PXDBInt]
[PXUIField(DisplayName = "Contract")]
[PXSelector(typeof(Search2<Contract.contractID,
   LeftJoin<ContractBillingSchedule, On<Contract.contractID,
Equal<ContractBillingSchedule.contractID>>>,
    Where<Contract.isTemplate, NotEqual<True>,
        And<Contract.baseType, Equal<Contract.ContractBaseType>,
        And<Where<Current<CRCase.customerID>, IsNull,
            Or2<Where<Contract.customerID, Equal<Current<CRCase.customerID>>,
                And<Current<CRCase.locationID>, IsNull>>,
            Or2<Where<ContractBillingSchedule.accountID, Equal<Current<CRCase.customerID>>,
                And<Current<CRCase.locationID>, IsNull>>,
            Or2<Where<Contract.customerID, Equal<Current<CRCase.customerID>>,
                And<Contract.locationID, Equal<Current<CRCase.locationID>>>>,
            Or<Where<ContractBillingSchedule.accountID, Equal<Current<CRCase.customerID>>,
                And<ContractBillingSchedule.locationID,
Equal<Current<CRCase.locationID>>>>>>,
    OrderBy<Desc<Contract.contractCD>>>),
    DescriptionField = typeof(Contract.description),
    SubstituteKey = typeof(Contract.contractCD), Filterable = true)]
[PXRestrictor(typeof(Where<Contract.status, Equal<Contract.status.active>>),
Messages.ContractIsNotActive)]
[PXRestrictor(typeof(Where<Current<AccessInfo.businessDate>, LessEqual<Contract.graceDate>,
Or<Contract.expireDate, IsNull>>), Messages.ContractExpired)]
[PXRestrictor(typeof(Where<Current<AccessInfo.businessDate>,
GreaterEqual<Contract.startDate>>), Messages.ContractActivationDateInFuture,
typeof(Contract.startDate))]
[PXFormula(typeof(Default<CRCase.customerID>))]
[PXDefault (PersistingCheck = PXPersistingCheck.Nothing)]
public virtual int? ContractID { get; set; }
```

Read Acumatica Platform Attributes Reference online: https://riptutorial.com/acumatica/topic/8853/acumatica-platform-attributes-reference

## Chapter 4: Adding Attribute Support to outof-box Sales Order Entity

#### Introduction

Acumatica ERP lets you define attributes for flexible, meaningful classification of an Entity (Lead, Stock/Non-Stock Items Etc.) as required for your company's specific needs. An attribute is a property that enables you to specify additional information for objects in the system. Attributes are defined in the context of a class which is a grouping of the business accounts (including leads, opportunities, customers, and cases), Stock and Non-Stock items by one or more of their properties.

### Remarks

This example is applicable to Acumatica 6.0 series

### Examples

This article provides how-to guide to add Acumatica ERP Attribute support to out-of-box Sales Order Entity

At the very core, your entity main DAC must have GUID column (NoteID) to reference CSAnswers table and must have field that identify the class of the Entity.

We will make use of order Type to define list of attributes to gather particular order type-specific information.

Create a Graph Extension for <code>soOrderTypeMaint</code> Graph and declare data view to define list of attributes for a particular order type. We will be using out-of-box <code>CSAttributeGroupList<TEntityClass</code>, <code>TEntity></code>

```
public class SOOrderTypeMaintPXExt : PXGraphExtension<SOOrderTypeMaint>
{
     [PXViewName(PX.Objects.CR.Messages.Attributes)]
     public CSAttributeGroupList<SOOrderType, SOOrder> Mapping;
}
```

Create a Graph Extension for *soorderEntry* Graph and declare data view for attributes specific to current order type.

```
public class SOOrderEntryPXExt : PXGraphExtension<SOOrderEntry>
{
    public CRAttributeList<SOOrder> Answers;
}
```

Create DAC Extension for soorder DAC and declare user defined field decorated with CRAttributesField attribute and specify the ClassID field – in our case it is OrderType.

```
public class SOOrderPXExt : PXCacheExtension<SOOrder>
{
    #region UsrAttributes
    public abstract class usrAttributes : IEqlField { }
    [CRAttributesField(typeof(SOOrder.orderType))]
    public virtual string[] UsrAttributes { get; set; }
    #endregion
}
```

Modify Order Types page (S0201000) as below using Customization Engine

```
<px:PXTabItem Text="Attributes">
 <Template>
   <px:PXGrid runat="server" BorderWidth="0px" Height="150px" SkinID="Details" Width="100%"
ID="AttributesGrid"
                MatrixMode="True" DataSourceID="ds">
        <AutoSize Enabled="True" Container="Window" MinHeight="150" />
        <Levels>
            <px:PXGridLevel DataMember="Mapping">
                <RowTemplate>
                    <px:PXSelector runat="server" DataField="AttributeID"
FilterByAllFields="True" AllowEdit="True"
                                    CommitChanges="True" ID="edAttributeID" /></RowTemplate>
                <Columns>
                    <px:PXGridColumn DataField="AttributeID" Width="81px" AutoCallBack="True"
LinkCommand="ShowDetails" />
                    <px:PXGridColumn DataField="Description" Width="351px" AllowNull="False"
/>
                    <px:PXGridColumn DataField="SortOrder" TextAlign="Right" Width="81px" />
                    <px:PXGridColumn DataField="Required" Type="CheckBox" TextAlign="Center"
AllowNull="False" />
                    <px:PXGridColumn DataField="CSAttribute__IsInternal" Type="CheckBox"
TextAlign="Center" AllowNull="True" />
                    <px:PXGridColumn DataField="ControlType" Type="DropDownList" Width="81px"
AllowNull="False" />
                    <px:PXGridColumn DataField="DefaultValue" RenderEditorText="False"
Width="100px" AllowNull="True" />
                </Columns>
            </px:PXGridLevel>
        </Levels>
    </px:PXGrid>
  </Template>
</px:PXTabItem>
```

Modify sales orders page (so301000) as below using Customization Engine

```
<Actions>
                <Search Enabled="False" />
            </Actions>
        </ActionBar>
        <Mode AllowAddNew="False" AllowDelete="False" AllowColMoving="False" />
        <Levels>
            <px:PXGridLevel DataMember="Answers">
                <Columns>
                    <px:PXGridColumn TextAlign="Left" DataField="AttributeID"</pre>
TextField="AttributeID_description"
                                         Width="250px" AllowShowHide="False" />
                    <px:PXGridColumn Type="CheckBox" TextAlign="Center" DataField="isRequired"
Width="80px" />
                    <px:PXGridColumn DataField="Value" Width="300px" AllowSort="False"</pre>
AllowShowHide="False" />
                </Columns>
            </px:PXGridLevel>
        </Levels>
    </px:PXGrid>
  </Template>
</px:PXTabItem>
```

Download deployment package

Read Adding Attribute Support to out-of-box Sales Order Entity online: https://riptutorial.com/acumatica/topic/8666/adding-attribute-support-to-out-of-box-sales-orderentity

## Chapter 5: Changing caption dynamically using readonly DAC fields.

#### Introduction

This example shows how to change dynamically the Caption/Label of Customer Name field on Customer ScreenID AR303000 on Acumatica ERP, depending on current Customer ID selected on the same form. We could:

### Examples

How-To

#### Add new unbound field to the DAC. (as readonly)

```
[PXString(60, IsUnicode = true)]
[PXUIField(Enabled = false, IsReadOnly = true)]
public virtual string UsrReadOnlyAcctName{get;set;}
public abstract class usrReadOnlyAcctName : IBqlField{}
```

Modify its value depending on conditions using handlers. (On Customer cycle ID Selected)

```
public class CustomerMaint_Extension:PXGraphExtension<CustomerMaint>
{
    protected void Customer_RowSelected(PXCache sender, PXRowSelectedEventArgs e)
    {
        var customer = (BAccount)e.Row;
        var customerExt = customer.GetExtension<BAccountExt>();
        if (customerExt != null)
        {
            customerExt != null)
        {
            customerExt.UsrReadOnlyAcctName = customer.AcctName;
        }
    }
}
```

SuppressLabel(true) for both new unbound fields and existing fields whose label will be replace.

### Layout Editor: AR303000 (Customers)

#### PREVIEW CHANGES ACTIONS -

с 📋 🔻	Ρ	roper	ties	Attributes	Events	Add
DataSource: CustomerMaint	¢	2	↔	Y		•
→ Porm: BAccount		Ove	rride	Property	/	
✓	-			Base P	roperties	5
I Customer ID				Commi	tChanges	
I UsrReadOnlyAcctName			~	DataFie	eld	
✓			✓	ID		
abi Status				Size		
I Customer Name				SkinID		
▶ 🗊 Column	Ŧ			Ext Pro	perties	
▶ 🗊 Tab: CurrentCustomer	►			AutoCa	llBack	
▶ 📾 Dialogs	Þ			AutoSiz	e	
				Disable	Spellchee	ck
			✓	Enable	d	
				Height		
				LabelW	/idth	
	Þ			LinkCo	mmand	
			✓	Suppre	ssLabel	
				SyncSt	ateWithC	omma
				TextAlig	<u>j</u> n	
				TextMo	de	

#### Place the added unbound field before the existing field.

#### Results:

Customer ID:	ACTIVESTAF	Q	Status:	Active
Active Staffing Service			* Active Sta	affing

Read Changing caption dynamically using readonly DAC fields. online: https://riptutorial.com/acumatica/topic/8858/changing-caption-dynamically-using-readonly-dac-fields-

## Chapter 6: Changing Size of Selector Drop-Down Window

#### Introduction

In this topic you will learn how to change size of the selector drop-down window. Each selector control in Acumatica has a button indicated with a magnifier icon. By clicking this button, users can open a drop-down window showing a list of objects available for selection.

### Examples

Changing default size ranges for selector drop-down window

The following 4 properties are available for **PXSelector** and **PXSegmentMask** input controls to define size range for a drop-down window:

- MinDropWidth: gets or sets the minimum drop-down control width
- MinDropHeight: gets or sets the minimum drop-down control height
- MaxDropWidth: gets or sets the maximum drop-down control width
- MaxDropHeight: gets or sets the maximum drop-down control height

Please be advised, the 4 properties listed above are hidden from the Properties window and won't be suggested to you by IntelliSense while editing Aspx pages in Visual Studio.

## To expand drop-down window width of the Customer selector

Default 13-column layout defined for the **Customer** selector on the **Sales Orders** screen (SO.30.10.00) doesn't quite fit the default size range specified for selector drop-down window. To help users explore as much information as possible and save their time on scrolling horizontally to see all of the columns, you need to increase the maximum drop-down control width by assigning a bigger number to the **MaxDropWidth** property of for the **Customer** selector.

To set value for the **MaxDropWidth** property in Layout Editor, uncheck **Hide Advanced Properties** radio button as shown on the screenshot below:

#### Layout Editor: SO301000 (Sales Orders)

PREVIEW CHANGES	ACTIONS -						
С 📋 🔻		Properties /	Attributes	Events	Add Controls	Add Da	ata Fields
C Truction of the control of the con		Properties       /         C       I→       /         Override       /         I→       I→       I→         I→       I→       I→       I→         I→       I→       I→       I→         I→       I→       I→       I→         I→       I→       I→       I→       I→         I→       I→       I→       I→       I→         I→       I→       I→       I→       I→         I→       I→       I→       I→       I→       I→         I→       I→       I→       I→       I→       I→       I→         I→       I→       I→       I→       I→       I→       I→       I→       I→       I→       I→       I→       I→       I→       I→       I→       I→	Attributes	Events  v geUrl ClientScr Theming ViewStat lor n terKey ld pelID ostfix	Add Controls	Add Da	Ita Fields Value Value
			MaxDro MaxDro Menulm	pHeight		2	2000
		•	MenuSt MinDro	tyles pHeight			

After publishing the customization, users can enjoy the new layout of **Customer** selector, now expanded upon entire working frame:

		+	Ì D.▼	K < >	>I AC	TIONS - REPOR	TS -	
* Order Ty	pe:		S0 P	* Customer:	ACTIVES	TAF	ہ م	🖉 Ordered Qty.:
Order Ni	or.:	Se	elect - Customer					
Status:		\$	SELECT C	⊷ + ▼				1
* Date:		8	Customer ID †	Customer Name		Address Line 1		Address Line 2
* Request	ed		ABARTENDE	USA Bartending S	chool	203 Lower Notch R	d)	
Custom	er (		ABCHOLDING	ABC Holdings Inc		65 Broadway		
External	Re		ABCSTUDIOS	ABC Studios Inc		77 W 66th St # 13		
Description			ABCVENTURE	ABC Capital Ventu	ires	601 W Girard Ave		
Document	De	>	ACTIVESTAF	Active Staffing Ser	vice	460 W 34th St		
c +			ALPHABETLD	Alphabetland Sch	ool Center	17575 Newbridge	Rd	
Ba P *	Bra		AMROBANK	AMRO Bank N.V.		351th Fl., Atago Gre	een Hills	55-12, Atago 26-chome, Min
	UIA		ANTUNSWEST	Antun's of Westch	ester	15 S Central Ave		
			APOSTELSCH	Church of The Ap	ostles	406 Willet Dr		
			ARTCAGES	Artcages		22112 Clay Spring	Loop	
			ASAHISUNTR	Asahi Sun Tours		45-87, Shiba Daim	on 17-ch	Hamamatsucho Seiwa Bldg.
			ASBLBAR	Nautilus Bar SABI	L	1216 Rue Lamartir	ne	
			AVACUST1	Avalara Customer	r	101 E Front St		
			BEAUTYSCH	New York Internat	ional Beau	143-53 South Dr		
			BESTYPEIMG	Bestype Image		4580 Broadway		
			BIBIMBAB	Bibimbab Korean	Restaurant	153 Lower Notch R	d	
			BORDERSHOP	Borders Books, M	usic & Cafe	3111 McCommas E	Blvd	
			BOULDERCR	Boulder Couriers	Denver	1899 Wynkoop St		Suite 700
			BRASSKEY	Brass Key Bar		11749 Livernois Av	e	

Read Changing Size of Selector Drop-Down Window online:

C Revision Two HQ - Sales Orders 🔺

https://riptutorial.com/acumatica/topic/9524/changing-size-of-selector-drop-down-window

## **Chapter 7: Conditionally Hiding Tabs**

#### Introduction

In this topic you will explore two approaches to conditionally hiding tabs on data entry screens in Acumatica.

#### **Examples**

VisibleExp Property of the PXTab Control in Aspx

The **VisibleExp** property is a boolean expression, that determines if given tab is visible (when logical expression is TRUE) or hidden. You specify **VisibleExp** property for PXTab controls in Aspx page:

```
<px:PXTabItem Text="Credit Card Processing Info" BindingContext="form"
    VisibleExp="DataControls[&quot;chkIsCCPayment&quot;].Value = 1">
```

**VisibleExp** is composed of input controls placed within the container with ID specified in the **BindingContext** property of PXTab control. You are not allowed to use input controls from more than one container. Access to a specific input control is provided through the DataControls dictionary by its ID, not the name of a DAC field.

Usually **VisibleExp** property is used to compose fairly simple boolean expressions with hardcoded input control values, that are unlikely to change with time. For instance, the following expression is used on the **Sales Orders** screen (SO.30.10.00) to hide **Payment Setting** tab for orders of the *Transfer* type:

```
<px:PXTabItem Text="Payment Settings"
VisibleExp="DataControls["edOrderType"].Value!=TR" BindingContext="form">
```

## To hide Activities tab for Leads with New status

To hide **Activities** tab from the **Leads** screen (CR.30.10.00), set **BindingContext** property to **form** (top-level *Lead Summary* form holds **form** ID) and define **VisibleExp** to return FALSE if lead status is Open (*Status* dropdown holds **edStatus** ID):

```
<pr:PXTabItem Text="Activities" LoadOnDemand="True"
BindingContext="form" VisibleExp="DataControls["edStatus"].Value != H">
```

C Revision Two H	Q - Leads ★ New ×		
SAVE & CLOSE	🗟 🗠 + 🔋 🗘 - H	< < > > AC	TIONS -
Lead ID:	Boyd, Ellis	Workgroup:	
* Status:	New 👻	Owner:	
Reason:	Assign 👻		
Details Attributes Relation	ns Campaigns Marketing Lists		
SUMMARY	Activities tab used to be	here CRM	
First Name:	- Ellis	Lead Class:	LEADBUS - Sales
* Last Name:	Boyd	Source:	Web
Position:	Developer	Campaign ID:	
Business Account:	م	Contact Method:	Any
Company Name:	Officemax North America Inc		Do Not Call
Parent Business Accou	م	e.	Do Not Email
CONTACT			□ No Mass Mail
Email:	eu.tempor.erat@velitPellentesqueultric 🖂	Last Incoming Activity:	
Web:		Last Outgoing Activity:	
Phone 1:	Business 1 👻	ADDRESS	
Phone 2:	Business 2 🔻		Same As In Acco
Phone 3:	Home 👻	Address Line 1:	
Fax:	Business Fa 👻	Address Line 2:	
		City:	Kansas City
		* Country:	US - UNITED STAT
		State:	MO - MISSOURI
		Postal Code:	95216

#### **AllowSelect Property on Data Views**

Unlike the **VisibleExp** property, defined in Aspx, you manipulate **AllowSelect** property of a data view though BLC or BLC extension code. The **AllowSelect** property makes it possible to use more complex boolean expressions (in comparison to the **VisibleExp** property) and, if necessary, retrieve additional information from database or other sources not available on a web page.

Below are 3 most common scenarios to work with the AllowSelect property:

 RowSelected event handler for top-level entity to hide Applications tab for invoices of Cash Sale and Cash Return types:

```
public class SOInvoiceEntry : ARInvoiceEntry
{
    ...
    protected override void ARInvoice_RowSelected(PXCache cache, PXRowSelectedEventArgs
e)
    {
        Adjustments.AllowSelect =
            doc.DocType != ARDocType.CashSale &&
            doc.DocType != ARDocType.CashReturn;
        }
        ...
}
```

• BLC constructor to show **Subitem Replenishment Info** tab on the **Item warehouse Details** screen only when both *Inventory Replenishment* and *Inventory Subitems* features are activated:

```
public class INItemSiteMaint : PXGraph<INItemSiteMaint, INItemSite>
{
    ...
    public INItemSiteMaint()
    {
        ...
        bool enableSubItemReplenishment =
PXAccess.FeatureInstalled<FeaturesSet.replenishment>() &&
PXAccess.FeatureInstalled<FeaturesSet.subItem>();
        subitemrecords.AllowSelect = enableSubItemReplenishment;
    }
    ...
}
```

 RowSelected handler for top-level entity to hide Depreciation History tab unless current asset is depreciable and Depreciation History View is set to Side by Side in the Fixed Assets Preferences:

```
public class AssetMaint : PXGraph<AssetMaint, FixedAsset>
{
    ...
    protected virtual void FixedAsset_RowSelected(PXCache sender, PXRowSelectedEventArgs
e)
    {
        ...
        AssetHistory.AllowSelect = asset.Depreciable == true &&
        fasetup.Current.DeprHistoryView == FASetup.deprHistoryView.SideBySide;
        }
        ...
    }
```

Every time **AllowSelect** property is used to conditionally change tab visibility though BLC or BLC extension code, you must set **RepaintOnDemand** property to *false* in Aspx for the corresponding PXTab container:

The **RepaintOnDemand** property is *true* by default. This property controls the initialization of PXTab container: when set to *true*, PXTab will not be initialized until it was selected by a user. Obviously you need **RepaintOnDemand** set to *false* to guarantee proper behavior of the given PXTab container despite whether it was selected or not.

## To hide Cross-Reference tab for Stock Items that can not be sold

To hide **Cross-Reference** tab from the **Stock Items** screen (IN.20.25.00) for items with **No Sales** status, proceed as follows:

1. implement **InventoryItem\_RowSelected** handler in the InventoryItemMaint BLC extension to set **AllowSelect** property to *false* for the *itemxrefrecords* data view if **Item Status** was set to **No Sales**:

```
public class InventoryItemMaintExt : PXGraphExtension<InventoryItemMaint>
{
    protected void InventoryItem_RowSelected(PXCache sender, PXRowSelectedEventArgs e)
    {
        InventoryItem item = (InventoryItem)e.Row;
        if (item == null) return;
        Base.itemxrefrecords.AllowSelect = (item.ItemStatus !=
InventoryItemStatus.NoSales);
    }
}
```

2. in Customization manager, set **RepaintOnDemand** property to *false* for the **Cross-Reference** tab and publish customization:

#### Layout Editor: IN202500 (Stock Items)

PREVIEW CHANGES ACTIONS -

C ∎ ▼	Pr	operties	Attributes	Events	Add Controls	Add Data
<ul> <li>DataSource: InventoryItemMaint</li> </ul>	с	;  ↔	T			
Form: Item		Override	Propert	tv		
	-		Base F	 Propertie	s	
General Settings			Binding	- gContext		
▶ 🗇 Subitems	•		Conten	- ntLayout		
► Price/Cost Info			LoadO	nDemand		
▶ 🗇 Warehouse Details			Repain	tOnDema	and	
▶ 🗇 Vendor Details			Text			
► 🗇 Attributes			Visible			
▶ 🗇 Packaging			Visible	Exp		
► 🗇 Cross-Reference	-		Ext Pr	operties		
▶ 📾 Replenishment Info	►		AutoCa	allBack		
▶						
► 📾 GL Accounts						
▶						
▶ 📾 Description						
Dialogs						
	Indi	cates whe	ether the tab	b item rep	aints its content	t from the s

After you completed 2 quite simple steps above, the **Cross-Reference** tab should not be accessible for Stock Items with **No Sales** status:

	🖹 🏫 🕂 🗗 -	∎ K < > >I	AC	OITO	NS -	INQUIRIE	ES 🕶			
	* Inventory ID:	AACOMPUT01	Q		Prod	uct Workgro	up:			
	Item Status:	No Sales 👻	Product Manager:							
	Description:	Acer Laptop Computer								
	General Settings Price/Cos	st Info Warehouse Details Vendor De	tails	Att	ributes	Packaging	Replen	ishment Info	Def	ern
	ITEM DEFAULTS				OF MEASUF	RE				
	Item Class:	ELECCOMP - Electronics & Compute	Q	R	★ Base Unit:			EA		đ
	Type:	Finished Good	-		★ Sales Unit:			EA	Q	đ
		⊡ls a Kit			* Purc	hase Unit:		EA	Q	đ
	Valuation Method:	Average	-		c + ×					
	* Tax Category:	TAXABLE - Taxable Goods and Service P         ELE - Electronics & Computers		R	+ From Multin					
	* Posting Class:			R	Unit		Μυπριγ/υνιαε		0	
	* Lot/Serial Class:	NOTTRACKED - Not Tracked	Q	R						
	Auto-Incremental Value:									
	WAREHOUSE DEFAULTS									
	Default Warehouse:	WHOLESALE - HQ Wholesale Wareh	ρ	R						
Default Issue From: Default Receipt To:		R1S1 - Row 1 Shelf 1PRECEIVING - ReceivingP		R	PHYSICAL INVENTORY					
				R	PI Cycle:					
					ABC	Code:				
								Fixed AB	C Co	de
					Movement Class:					
								Fixed Mov	/eme	nt

C Revision Two HQ - Stock Items 🔺

## To hide Attributes tab for inactive Stock Items

To conditionally hide \*\* Attributes\*\* tab from the **Stock Items** screen (IN.20.25.00), proceed as follows:

1. implement InventoryItem\_RowSelected handler in the InventoryItemMaint BLC extension to set AllowSelect property to *false* for the Answers and Category data views if Item Status was set to *Inactive*. Also notice Visible property set to *false* for PXUIFieldAttribute added on the InventoryItem.ImageUrl field by CacheAttached handler:

```
public class InventoryItemMaintExt : PXGraphExtension<InventoryItemMaint>
{
   protected void InventoryItem_RowSelected(PXCache sender, PXRowSelectedEventArgs e)
    {
        InventoryItem item = (InventoryItem)e.Row;
       if (item == null) return;
       bool showAttributesTab = item.ItemStatus != InventoryItemStatus.Inactive;
       Base.Answers.AllowSelect = Base.Category.AllowSelect = showAttributesTab;
       PXUIFieldAttribute.SetVisible<InventoryItem.imageUrl>(sender, item,
showAttributesTab);
   }
    [PXMergeAttributes (Method = MergeMethod.Append)]
    [PXUIField(DisplayName = "Image")]
   protected void InventoryItem_ImageURL_CacheAttached(PXCache sender)
    { }
}
```

2. in Customization manager, set **RepaintOnDemand** property to *false* for the **Attributes** tab and publish customization:
#### Layout Editor: IN202500 (Stock Items)

PREVIEW CHANGES ACTIONS -

C 📋 🔻	Pr	operties	Attributes	Events	Add Controls	Add Data
<ul> <li>DataSource: InventoryItemMaint</li> </ul>	с	:  ↔	Ŧ			
<ul> <li>Form: Item</li> </ul>		Override	Propert	v		
	+		Base F	, Propertie	s	
▶ 🗇 General Settings			Binding	Context		
▶ 🗊 Subitems	•		Conten	tLayout		
► 🗊 Price/Cost Info			LoadO	nDemand		
▶ 🗇 Warehouse Details		✓	Repain	tOnDema	and	
▶ 🗇 Vendor Details			Text			
► 🗇 Attributes			Visible			
▶ 🗇 Packaging			Visible	Exp		
Gross-Reference	-		Ext Pr	operties		
► 🗊 Replenishment Info	•		AutoCa	allBack		
▶ 🗇 Deferral Settings						
► 🗇 GL Accounts						
Restriction Groups						
<ul> <li>Description</li> </ul>						
▶ 🗊 Dialogs						
	Indi	cates whe	ether the tab	o item rep	aints its conten	t from the s

After you completed 2 steps above, the **Attributes** tab should not be accessible for Stock Items with *Inactive* status:

* Inventory ID: AALEGO500		ALEGO500	Q		Produ	ict Workgrou	up:			
Item Status:	In	Inactive 👻			Produ	ict Manager	-			
Description:	L	Lego 500 piece set		_						
General Settings	Price/Cost Ir	nfo Warehouse Details	Vendor Details	Pa	ckaging	Cross-Refe	erence	Replenis	hment Info	,
ITEM DEFAULTS	;			-	UNIT O	F MEASUR	E —			
Item Class:	С	ONSUMER - Consumer	Goods 🖉 🔎	æ	* Base	Unit:		EA		1
Type:	F	inished Good	<b>.</b>		* Sales	Unit:		EA	و	6
		ls a Kit			* Purch	nase Unit:		EA	Q	1
Valuation Meth	od: A	verage	Ŧ		с	+	×			
* Tax Category:	T/	AXABLE - Taxable Goods	and Service P	R	* F	rom	Multiph	/Divide		••
* Posting Class:	С	ON - Consumer Goods	Q	R	i	Jnit	manapi	Divide		
* Lot/Serial Clas	s: N	OTTRACKED - Not Track	ked 🔎	R						
Auto-Increment	tal Value:			_						
WAREHOUSE D	)efaults —			-						
Default Wareho	ouse: V	HOLESALE - HQ Whole	esale Wareh 🔎	æ						
Default Issue F	rom: R	1S1 - Row 1 Shelf 1	Q	æ	PHYSI	CAL INVEN	TORY -			
Default Receipt	t To: R	ECEIVING - Receiving	Q	æ	PI Cy	cle:				
					ABC	Code:				
								Fixed	ABC Cod	е
					Move	ment Class:				
								Fixed	Movemen	t

C Revision Two HQ - Stock Items 🔺

Read Conditionally Hiding Tabs online: https://riptutorial.com/acumatica/topic/9506/conditionallyhiding-tabs

# Chapter 8: Creating Date and Time Fields in Acumatica

#### Introduction

This topic will walk you through different options available in the Acumatica Framework to create date and time fields in a data access class (DAC).

#### **Examples**

```
The PX(DB)DateAndTime Attribute
```

The **PXDBDateAndTime** attribute and the **PXDateAndTime** attribute are designed to work with a DAC field of the Nullable<DateTime> (DateTime?) type and store both date and time value parts inside a single field:

```
#region UsrDateAndTime
public abstract class usrDateAndTimeAttribute : IBqlField
{ }
[PXDBDateAndTime(
    DisplayNameDate = "Date Value Part",
    DisplayNameTime = "Time Value Part")]
public DateTime? UsrDateAndTime { get; set; }
#endregion
```

From the UI perspective, for a field decorated with **PXDBDateAndTimeAttribute** or **PXDateAndTimeAttribute**, one is expected to create either separate input controls for date and time value parts:

DATE AND TIME FIEL	.D			
Date Value Part:	7/12/2017 👻	Time Value Part:	2:30 PM 👻	
<px:pxdatetimeedit <px:pxdatetimeedit TimeMode="True" /&gt;</px:pxdatetimeedit </px:pxdatetimeedit 	runat="server" runat="server"	ID="edUsrDate" ID="edUsrTime"	DataField="UsrDateA DataField="UsrDateA	ndTime_Date" /> ndTime_Time"

or separate grid columns to enter and display date and time values:

Date Value Part	Time Value Part			
7/14/2017	9:30 AM			

```
<Columns>
...
<px:PXGridColumn DataField="UsrDateAndTime_Date" Width="90px" />
```

```
<pr:PXGridColumn DataField="UsrDateAndTime_Time" Width="90px" TimeMode="True" />
...
</Columns>
```

#### The PXDBTime Attribute

The **PXDBTime** attribute is designed to work with a DAC field of the Nullable<DateTime> (DateTime?) type and store only the time part without date inside a DAC field:

```
#region UsrTime
public abstract class usrTime : IBqlField
{ }
 [PXDBTime(DisplayMask = "t", InputMask = "t")]
 [PXUIField(DisplayName = "Time Only Value")]
 public DateTime? UsrTime { get; set; }
#endregion
```

In the UI, for a field decorated with **PXDBTimeAttribute** the system creates an input control accepting only time values both on a form:

Time Only Value:	2:00 PM	-
	2:30 AM	^
	3:00 AM	
	3:30 AM	
Longiture 1	4:00 AM	
increase a	4:30 AM	
	5:00 AM	

iii

<px:PXDateTimeEdit runat="server" ID="edUsrTime" DataField="UsrTime" TimeMode="True" />

#### and within a grid cell:

**Time Only Value** 

2:3	0 PM	•
1:00	PM	^
1:30	PM	
2:00	PM	
2:30	PM	
3:00	PM	
3:30	PM	
<col< td=""><td>.umns&gt;</td><td></td></col<>	.umns>	
	<px:pxgridco< td=""><td>olun</td></px:pxgridco<>	olun
CC</td <td>unins/</td> <td></td>	unins/	

#### The PX(DB)DateAttribute Attribute

The **PXDBDate** attribute and the **PXDate** attribute are designed to work with a DAC field of the Nullable<DateTime> (DateTime?) type and store date value with an optional time part inside a single field. Wheather **PX(DB)DateAttribute** should save time in addition to date in a DAC field is defined by the **PreserveTime** property: when **PreserveTime** is set to **True**, the time part of a field value is preserved, otherwise only the date part is saved in a DAC field:

```
#region UsrDateTime
public abstract class usrDateTime : IBqlField
{ }
[PXDBDate(PreserveTime = true, InputMask = "g")]
[PXUIField(DisplayName = "DateTime Value")]
public DateTime? UsrDateTime { get; set; }
#endregion
#region UsrDate
public abstract class usrDate : IBqlField
{ }
[PXDBDate]
[PXUIField(DisplayName = "Date Value")]
public DateTime? UsrDate { get; set; }
#endregion
```

In the UI, for a field decorated with **PXDBDateAttribute** or **PXDateAttribute** the system creates an input control accepting either only date values or both date and time values depending on the value of **PreserveTime** property. This concept works exactly the same on a form:

DATE AND OPTIONAL	TIME FIELD			
DateTime Value:	7/12/2017 3:15 PM 🔹	Date Value:	7/18/2017 👻	
<px:pxdatetimeedit <px:pxdatetimeedit< td=""><td>runat="server" ID="e runat="server" ID="e</td><td>dUsrDateTime" DataE dUsrDate" DataFielo</td><td>ield="UsrDateT: d="UsrDate" /&gt;</td><td>ime" Size="SM" /&gt;</td></px:pxdatetimeedit<></px:pxdatetimeedit 	runat="server" ID="e runat="server" ID="e	dUsrDateTime" DataE dUsrDate" DataFielo	ield="UsrDateT: d="UsrDate" />	ime" Size="SM" />

and within a grid cell:



```
<Columns>

...

<px:PXGridColumn DataField="UsrDateTime" Width="130px" />

<px:PXGridColumn DataField="UsrDate" Width="90px" />

...

</Columns>
```

#### The PXDBTimeSpan Attribute

The **PXDBTimeSpan** attribute is designed to work with a DAC field of the Nullable<int> (int?) type and store time value inside a DAC field as the number of minutes passed since midnight:

```
#region UsrTimeInt
public abstract class usrTimeInt : IBqlField
{ }
[PXDBTimeSpan(DisplayMask = "t", InputMask = "t")]
[PXUIField(DisplayName = "Time Value")]
public int? UsrTimeInt { get; set; }
#endregion
```

In the UI, for a field decorated with **PXDBTimeSpanAttribute** the system creates a drop-down with half hour interval values both on a form:

Time Value:	7:15 AM	•
	12:00 AM	^
	12:30 AM	
	1:00 AM	
ay to state the	1:30 AM	
80 - A 190	2:00 AM	
	2:30 AM	

and within a grid cell:

```
<pr:PXDateTimeEdit runat="server" ID="edUsrTimeInt" DataField="UsrTimeInt" TimeMode="true" />
Time Value
7:15 AM
7:15 AM
1:2:00 AM
1:2:30 AM
1:30 AM
2:30 AM
2:30 AM
```

### The PXTimeList Attribute

The **PXTimeList** attribute is designed to work with a DAC field of the Nullable<int> (int?) type and store time span value inside a DAC field as a number of minutes:

```
#region UsrTimeSpan
public abstract class usrTimeSpan : IBqlField
{ }
 [PXDBInt]
 [PXTimeList]
```

https://riptutorial.com/

```
[PXUIField(DisplayName = "Time Span")]
public int? UsrTimeSpan { get; set; }
#endregion
```

In the UI, for a field decorated with **PXTimeListAttribute** the system creates a drop-down with 30minute interval values both on a form:

Time Span:	01:40	-
	00:00	^
	00:30	
Construction of the	01:00	
Constituent (	01:30	
factory .	02:00	

<px:PXTimeSpan ID="edUsrTimeSpan" runat="server" DataField="UsrTimeSpan" InputMask="hh:mm" />

#### and within a grid cell:

Tir	1	1		
_8:45			+	1
00:00	^	1		
00:30				
01:00				
01:30				
02:00				

```
<RowTemplate>
...
<px:PXTimeSpan ID="edgUsrTimeSpan" runat="server" DataField="UsrTimeSpan"
InputMask="hh:mm" />
...
</RowTemplate>
<Columns>
...
<px:PXGridColumn DataField="UsrTimeSpan" Width="90px" RenderEditorText="True" />
...
</Columns>
```

Read Creating Date and Time Fields in Acumatica online: https://riptutorial.com/acumatica/topic/10783/creating-date-and-time-fields-in-acumatica

## **Chapter 9: Customization Mechanisms**

#### **Examples**

Using CacheAttached to Override DAC Attributes in the Graph

Sometimes, you need to override one or more attributes of a particular Data Access Class (DAC) field just for a particular screen, without changing the existing behavior for other screens.

## **Replacing All Attributes**

Suppose the original DAC field attributes are declared as shown below:

```
public class ARInvoice : IBqlTable
{
    [PXDBDecimal(4)]
    [PXDefault(TypeCode.Decimal, "0.0")]
    [PXUIField(DisplayName = "Commission Amount")]
    public virtual Decimal? CommnAmt
    {
        get;
        set;
    }
}
```

The basic way to override the field's attributes in the graph is to declare a *cacheAttached* event handler in the graph that follows the standard convention for naming graph events (note the absence of the EventArgs argument). The body of the event handler will not be executed, but any *attributes* that you place on the handler will replace the attributes on the corresponding DAC field:

```
[PXDBDecimal(4)]
[PXDefault(TypeCode.Decimal, "0.0")]
[PXUIField(DisplayName = "Commission Amount")]
[PXAdditionalAttribute(NecessaryProperty = true)]
protected virtual void ARInvoice_CommnAmt_CacheAttached(PXCache sender) { }
```

## Appending a New Attribute to the DAC Field

The set of attributes placed on the CacheAttached handler will **redefine the whole set of the attributes** placed on the field in the DAC. This is almost always overkill; note how in the previous example, in order to add just a single attribute to the field, you had to copy all other attribute declarations from the DAC. This leads to undesired code duplication, as well as the possibility of DAC and the graph going out of sync. It is very easy to imagine a situation when someone changes the defaulting logic of, for instance, <code>PXDefaultAttribute</code> in the DAC, but forgets to update all the corresponding attributes placed on the <code>CacheAttached</code> handlers in various graphs.

To remedy this problem, the Acumatica Framework provides a special attribute called PXMergeAttributesAttribute. When this attribute is placed on a CacheAttached handler, you can reuse the existing attributes defined in the DAC.

Appending an attribute using PXMergeAttributesAttribute:

```
[PXMergeAttributes(Method = MergeMethod.Append)]
[PXAdditionalAttribute(NecessaryProperty = true)]
protected virtual void ARInvoice_CommnAmt_CacheAttached(PXCache sender) { }
```

In the above example, the whole set of attributes from the original DAC will be reused, appended by any attributes that you have declared on the CacheAttached event handler.

PXMergeAttributesAttribute has other merge behaviours, according to the following possible values for the Method property:

- MergeMethod.Replace replaces the DAC's attributes completely (equivalent to the absence of PXMergeAttributesAttribute).
- MergeMethod.Append appends the attributes from the CacheAttached handler to the original DAC attributes.
- MergeMethod.Merge is similar to Append; however, it also checks whether there are any conflicting attributes between the handler attributes and the DAC field attributes. If there is a conflict, the CacheAttached attribute takes precedence and the corresponding DAC attribute is discarded.

## **Overriding a Single Property of an Attribute**

A very common application development scenario occurs when you have to redefine just a single property of a DAC's attribute for a particular screen; consider the situation when you have to define the DisplayName property of the PXUIFieldAttribute.

For that purpose, you can use yet another special attribute provided by the Acumatica Framework: PXCustomizeBaseAttributeAttribute. Its constructor accepts three values:

- The type of the DAC attribute whose property needs to be overridden
- The name of the attribute's property to override (use the nameof operator in C# 6.0 for code maintainability)
- The new value for the specified property.

Suppose that there is a requirement to change the UI display name from *Commission Amount* to *Base Currency Commission* for only one screen. The following code example demonstrates how to implement the desired behavior.

```
[PXMergeAttributes(Method = MergeMethod.Append)]
[PXCustomizeBaseAttribute(typeof(PXUIFieldAttribute), nameof(PXUIFieldAttribute.DisplayName),
"Base Currency Commission")]
protected virtual void ARInvoice_CommnAmt_CacheAttached(PXCache sender) { }
```

In this example, PXMergeAttributes ensures that the original DAC attributes are preserved, and PXCustomizeBaseAttribute allows the software engineer to override the UI field's display name for the graph in question.

## **Replacing an Attribute with Another Attribute**

Suppose that there is a requirement to replace a DAC field's PXDefaultAttribute with PXDBDefaultAttribute for only one screen.

This can be achieved in the following manner:

```
[PXMergeAttributes(Method = MergeMethod.Append)]
[PXRemoveBaseAttribute(typeof(PXDefaultAttribute))]
[PXDBDefault(typeof(SOShipment.siteID), PersistingCheck = PXPersistingCheck.Nothing)]
protected void SOOrderShipment_SiteID_CacheAttached(PXCache sender) { }
```

## Application Order of the Attribute-Customizing Attributes

- 1. PXCustomizeBaseAttribute
- 2. PXRemoveBaseAttribute
- 3. PXMergeAttributes

Read Customization Mechanisms online: https://riptutorial.com/acumatica/topic/9751/customization-mechanisms

# Chapter 10: Displaying an Error Requiring to Enter Entity Data

### Examples

Displaying an Error Requiring the User to Enter Entity Data

Users often turn up in a situation when a business process cannot be finished because the user has not entered all the necessary information.

An example of this situation is when a user tries to create a drop-ship order with missing customer address.

According to UX best practices, the system should be friendly to the user and not only inform the user about the situation, but also guide him to the resolution of his issue. As we know, the system already has a similar mechanism activated by PXSetup<TSetup>.Current when there are no records in the TSetup table. It is internally implemented by throwing a PXSetupNotEnteredException.

Recently, a new functionality has been added to this exception, which allows an application developer to throw an error with a link to the entity which must be re-configured:

```
INSite erroneousSite = PXSelect<
    INSite,
    Where<
        INSite.siteID, Equal<Current<SOCreateFilter.siteID>>,
        And<INSite.active, Equal<True>,
        And<Where<INSite.addressID, IsNull, Or<INSite.contactID, IsNull>>>>>>
    .SelectSingleBound(this, new object[] { e.Row });

if (erroneousSite != null)
{
    throw new PXSetupNotEnteredException<INSite, INSite.siteCD>(
        Messages.WarehouseWithoutAddressAndContact,
        erroneousSite.SiteCDInk,
        erroneousSite.SiteCDinf);
}
```

The result is displayed to the user like this:

#### Error #81

The requested resource is not available. The Multiple Warehouses feature and the Transfer or

#### Next step:

Navigate to the Warehouse form and enter the required configuration data.

- As a first type parameter, PXSetupNotEnteredException accepts the type of the entity to which the default graph link will be generated.
- The second type parameter denotes the key field of the record to be used to generate the link. In the above example, navigation to the warehouse entity is made by the CD key.
- The first constructor argument is the format string for the error message. The numbering of its internal placeholders should start with 1: i.e. The Multiple Warehouses feature and the Transfer order type are activated in the system, in this case an address and a contact must be configured for the '{1}' warehouse.
- The second constructor argument is the value of the key field specified as the second generic parameter. In the example, the link that would be generated is /IN204000.aspx?siteCD=erroneousSite.SiteCDlnk.
- The third constructor argument is the human-readable value to be displayed in the error message: ...in this case an address and a contact must be configured for the 'erroneousSite.SiteCDinf' warehouse.

Read Displaying an Error Requiring to Enter Entity Data online: https://riptutorial.com/acumatica/topic/9274/displaying-an-error-requiring-to-enter-entity-data

# Chapter 11: Downloading Files Attached to a Detail Entity Using Contract-Based API

### Introduction

This topic will demonstrate how to download files attached to a detail entity inside Acumatica ERP by using the Contract-Based API.

### Remarks

The code snippet above was created using the Json.NET framework (Newtonsoft.Json.dll).

To obtain HTTP cookie header from a SOAP response, add a reference to the .Net framework **System.ServiceModel** and **System.ServiceModel.Web** assemblies and the following 2 using directives in your code file:

```
using System.ServiceModel;
using System.ServiceModel.Web;
```

### Examples

HTTP Cookie Header from a SOAP Response Shared by SOAP and REST Clients

There is a limitation in Acumatica's SOAP Contract-Based API allowing to download attachments only for a top-level entity. Any attempt to use the GetFiles() method to get the attachments of a detail entity will, unfortunately, result in the error "Entity without screen binding cannot be used as top level entity." telling us it can only be used with a top-level entity defined in the web service endpoint.

Another limitation with the GetFiles() method is that it always returns the content of all files attached to an entity. There is no option to first retrieve only file names and then decide what particular file(s) to download from Acumatica.

Thankfully, there is a better and more controllable way to work with attachments provided with the Contract-Based REST API. The files array returned as part of every entity exported by the Contract-Based REST API contains only:

- file names (the filename property)
- file identifiers (the id property)
- hypertext references (the href property), which can be used later to download file content

For an example of obtaining a list of files attached to any entity from the web service endpoint and retrieving particular file content though the Contract-Based REST API, please check Acumatica

#### **Product Help**

How can one download the files attached to a detail entity if the entire integration project was developed with the SOAP Contract-Based API? As shown in the code snippet below, it is possible to pass HTTP cookie header from a SOAP response into the REST API client exclusively used to work with the attachments:

```
using (var soapClient = new DefaultSoapClient())
{
   var address = new Uri("http://localhost/AcumaticaERP/entity/Default/6.00.001/");
   CookieContainer cookieContainer;
   using (new OperationContextScope(soapClient.InnerChannel))
        soapClient.Login(login, password, null, null, null);
       string sharedCookie = WebOperationContext.Current.IncomingResponse.Headers["Set-
Cookie"];
       cookieContainer = new CookieContainer();
       cookieContainer.SetCookies(address, sharedCookie);
    }
    try
    {
        var shipment = new Shipment()
        {
            ShipmentNbr = new StringSearch { Value = "001301" },
            ReturnBehavior = ReturnBehavior.OnlySpecified
        };
        shipment = soapClient.Get(shipment) as Shipment;
        var restClient = new HttpClient(
           new HttpClientHandler
            {
                UseCookies = true,
                CookieContainer = cookieContainer
            });
        restClient.BaseAddress = address;// new
Uri("http://localhost/059678/entity/Default/6.00.001/");
        var res = restClient.GetAsync("Shipment/" + shipment.ID + "?$expand=Packages")
            .Result.EnsureSuccessStatusCode();
        var shipmentWithPackages = res.Content.ReadAsStringAsync().Result;
        JObject jShipment = JObject.Parse(shipmentWithPackages);
        JArray jPackages = jShipment.Value<JArray>("Packages");
        foreach (var jPackage in jPackages)
        {
            JArray jFiles = jPackage.Value<JArray>("files");
            string outputDirectory = ".\\Output\\";
            if (!Directory.Exists(outputDirectory))
                Directory.CreateDirectory(outputDirectory);
            }
            foreach (var jFile in jFiles)
            {
                string fullFileName = jFile.Value<string>("filename");
                string fileName = Path.GetFileName(fullFileName);
                string href = jFile.Value<string>("href");
                res = restClient.GetAsync(href).Result.EnsureSuccessStatusCode();
```

```
byte[] file = res.Content.ReadAsByteArrayAsync().Result;
System.IO.File.WriteAllBytes(outputDirectory + fileName, file);
}
finally
{
soapClient.Logout();
}
```

Read Downloading Files Attached to a Detail Entity Using Contract-Based API online: https://riptutorial.com/acumatica/topic/10692/downloading-files-attached-to-a-detail-entity-usingcontract-based-api

## Chapter 12: Exporting Records via REST Contract-Based API

### Introduction

This topic will demonstrate how to export records from Acumatica ERP via the REST Contract-Based API. In contrast to the Screen-Based API of Acumatica ERP, the Contract-Based API provides both SOAP and REST interfaces. For more information on the Contract-Based API, see Acumatica ERP Documentation

### Remarks

To communicate with the REST Contract-Based API of Acumatica ERP your client application must always perform the following 3 steps:

- 1. log into Acumatica ERP instance and get cookie with user session information
- 2. interact with one of Contract-Based API endpoints available on Acumatica ERP instance
- 3. log out from Acumatica ERP to close user session

All samples provided in this topic were built with the **Default** endpoint, always deployed as part of the standard Acumatica ERP installation process. On the **Web Service Endpoints** screen (SM.20.70.60) you can view the details of existing endpoints or configure your custom endpoints of the Acumatica ERP contract-based web services:

#### C Revision Two HQ - Web Service Endpoints 🔺

F

+

VIEW ENDPOINT SERVICE VIEW MAINTEI

VIEW MAINTENANCE SERVICE EXTEND ENDP

* Endpoint Name: Default		Q	<ul> <li>Endpoint Version</li> </ul>	c (	6.00.001
		Endpoint Properties			
	^	* Endpoint Name:	Default		Base Endpo
⊕ — — Adjustment		* Endpoint Version:	6.00.001		Base Endpo
		System Contract:	2		
⊕ <sup></sup> ⊖ Contact					
🖶 🗁 Customer					
⊕ " 🛅 CustomerPaymentMethod					
⊕ — Employee					
InventoryAllocationInquiry					
InventorySummaryInquiry					
🗊 🛅 ItemClass					
⊕ " 🛅 ItemSalesCategory	•				

For your reference, below is implementation of the **RestService** class used in all samples above to interact with the Contract-Based web service of Acumatica ERP:

```
public class RestService : IDisposable
{
   private readonly HttpClient _httpClient;
   private readonly string _acumaticaBaseUrl;
   private readonly string _acumaticaEndpointUrl;
   public RestService(string acumaticaBaseUrl, string endpoint,
       string userName, string password,
       string company, string branch)
    {
       _acumaticaBaseUrl = acumaticaBaseUrl;
       _acumaticaEndpointUrl = _acumaticaBaseUrl + "/entity/" + endpoint + "/";
       _httpClient = new HttpClient(
           new HttpClientHandler
            {
                UseCookies = true,
                CookieContainer = new CookieContainer()
            })
        {
            BaseAddress = new Uri(_acumaticaEndpointUrl),
            DefaultRequestHeaders =
            {
                Accept = {MediaTypeWithQualityHeaderValue.Parse("text/json")}
            }
        };
```

```
var str = new StringContent(
       new JavaScriptSerializer()
            .Serialize(
                new
                {
                    name = userName,
                    password = password,
                    company = company,
                    branch = branch
                }),
                Encoding.UTF8, "application/json");
    _httpClient.PostAsync(acumaticaBaseUrl + "/entity/auth/login", str)
        .Result.EnsureSuccessStatusCode();
}
void IDisposable.Dispose()
{
    _httpClient.PostAsync(_acumaticaBaseUrl + "/entity/auth/logout",
       new ByteArrayContent(new byte[0])).Wait();
   _httpClient.Dispose();
}
public string GetList(string entityName)
{
    var res = _httpClient.GetAsync(_acumaticaEndpointUrl + entityName)
        .Result.EnsureSuccessStatusCode();
   return res.Content.ReadAsStringAsync().Result;
}
public string GetList(string entityName, string parameters)
{
   var res = _httpClient.GetAsync(_acumaticaEndpointUrl + entityName + "?" + parameters)
        .Result.EnsureSuccessStatusCode();
    return res.Content.ReadAsStringAsync().Result;
}
```

#### **Examples**

Data Export in a Single REST Call

In this example you will explore how to export the following data from Acumatica ERP in a single call via the REST Contract-Based API:

- · all stock items existing in the application
- all sales order of the IN type

If you need to export records from Acumatica ERP, use the following URL: http://<Acumatica ERP instance URL>/entity/<Endpoint name>/<Endpoint version>/<Top-level entity>

<Top-level entity> is the name of the entity which you are going to export

# To export all stock items in a single REST call:

To export stock item records from a local AcumaticaERP instance by using the **Default** endpoint of version **6.00.001**, you should use the following URL:

http://localhost/AcumaticaERP/entity/Default/6.00.001/StockItem

Below is the sample code written in C# to export all stock items by sending a single REST call to the **Default** endpoint of version **6.00.001**:

```
using (RestService rs = new RestService(
    @"http://localhost/AcumaticaERP/", "Default/6.00.001",
    username, password, company, branch))
{
    string stockItems = rs.GetList("StockItem");
}
```

# To export all sales order of the IN type in a single REST call:

To export sales orders of the *IN* type from a local AcumaticaERP instance by using the *Default* endpoint of version *6.00.001*, you should use the following URL: http://localhost/AcumaticaERP/entity/Default/6.00.001/SalesOrder?\$filter=OrderType eq 'IN'

Below is the sample code written in C# to export all sales orders of the *IN* type by sending a single REST call to the *Default* endpoint of version *6.00.001*:

```
using (RestService rs = new RestService(
    @"http://localhost/StackOverflow/", "Default/6.00.001",
    username, password, company, branch))
{
    var parameters = "$filter=OrderType eq 'IN'";
    string inSalesOrders = rs.GetList("SalesOrder", parameters);
}
```

**Implementing Pagination on Multiple REST Requests** 

In this example you will explore how to export the following data from Acumatica ERP in batches via the REST Contract-Based API:

- · stock items existing in the application in batches of 10 records
- all sales orders in batches of 100 records

## To export stock items in batches of 10

## records with multiple REST calls:

To export first 10 stock items from a local AcumaticaERP instance by using the **Default** endpoint of version **6.00.001**, you should use the following URL:

http://localhost/AcumaticaERP/entity/Default/6.00.001/StockItem?\$top=10

Accordingly, to request stock items from 10 to 20, you simply extend the URL above with *filter* parameter:

http://localhost/AcumaticaERP/entity/Default/6.00.001/StockItem?\$top=10&\$filter=InventoryID gt
'<InventoryID>'

<InventoryID> is the ID of the last stock item exported with a previous REST call

Below is the sample code written in C# to export all stock items in batches of 10 records by sending multiple REST calls to the *Default* endpoint of version *6.00.001*:

```
using (RestService rs = new RestService(
   @"http://localhost/StackOverflow/", "Default/6.00.001",
   username, password, company, branch))
{
   var json = new JavaScriptSerializer();
    string parameters = "$top=10";
   string items = rs.GetList("StockItem", parameters);
   var records = json.Deserialize<List<Dictionary<string, object>>>(items);
   while (records.Count == 10)
    {
        var inventoryID = records[records.Count - 1]["InventoryID"] as Dictionary<string,</pre>
object>;
        var inventoryIDValue = inventoryID.Values.First();
        string nextParameters = parameters + "&" +
            "$filter=" + string.Format("InventoryID gt '{0}'", inventoryIDValue);
       items = rs.GetList("StockItem", nextParameters);
        records = json.Deserialize<List<Dictionary<string, object>>>(items);
    }
}
```

# To export all sales orders in batches of 100 records with multiple REST calls:

To export first 100 sales orders from a local AcumaticaERP instance by using the **Default** endpoint of version **6.00.001**, you should use the following URL: http://localhost/AcumaticaERP/entity/Default/6.00.001/SalesOrder?\$top=100

Since the primary key of the **Sales Order** entity is composed by the **Order Type** and the **Order Number**, in this example you will be using a combination of **filter** parameters for the **Order Type** and **Order Number** fields:

• to request sales orders from 100 to 200 of the **SO** type, you should use the following URL: http://localhost/AcumaticaERP/entity/Default/6.00.001/SalesOrder?\$top=100&\$filter=OrderType eq 'SO' and OrderNbr gt '<OrderNbr>' <OrderNbr> is the number of the last sales order exported with a previous REST call

 accordingly, to request first 100 sales orders of the next to SO type, you should use the following URL:

```
http://localhost/AcumaticaERP/entity/Default/6.00.001/SalesOrder?$top=100&$filter=OrderType
gt 'SO' and OrderNbr gt ''
```

Below is the sample code written in C# to export all sales orders in batches of 100 records with multiple REST calls to the *Default* endpoint of version *6.00.001*:

```
using (RestService rs = new RestService(
   @"http://localhost/StackOverflow/", "Default/6.00.001",
   username, password, company, branch))
{
   var json = new JavaScriptSerializer();
    string parameters = "$top=100";
    string items = rs.GetList("SalesOrder", parameters);
   var records = json.Deserialize<List<Dictionary<string, object>>>(items);
   bool sameOrderType = true;
   while (records.Count > 0 && (records.Count == 100 || !sameOrderType))
    {
       var orderType = records[records.Count - 1]["OrderType"] as Dictionary<string, object>;
       var orderTypeValue = orderType.Values.First();
        var orderNbr = records[records.Count - 1]["OrderNbr"] as Dictionary<string, object>;
        var orderNbrValue = orderNbr.Values.First();
        string nextParameters = parameters + "&" + "$filter=" +
           string.Format("OrderType {0} '{1}'", sameOrderType ? "eq" : "gt", orderTypeValue)
+ " and " +
           string.Format("OrderNbr gt '{0}'", sameOrderType ? orderNbrValue : "''");
        items = rs.GetList("SalesOrder", nextParameters);
        records = json.Deserialize<List<Dictionary<string, object>>>(items);
       sameOrderType = records.Count == 100;
   }
}
```

Read Exporting Records via REST Contract-Based API online: https://riptutorial.com/acumatica/topic/9298/exporting-records-via-rest-contract-based-api

## Chapter 13: Exporting Records via Screen-Based API

### Introduction

This topic will demonstrate how to export records from Acumatica ERP via the Screen-Based API. The Screen-Based API of Acumatica ERP provides only the SOAP interface. If your development platform has limited support for SOAP web services, consider the Contract-Based API providing both SOAP and REST interfaces. For more information on the Screen-Based API, see Acumatica ERP Documentation

### Remarks

All sample provided in this topic were created with the Screen-Based API Wrapper. If you want your client application to not depend on the UI changes in the Acumatica ERP application, you should use the screen-based API wrapper, which is described in Acumatica ERP Documentation

### Examples

Data Export from an Entry Form with a Single Primary Key

The **Stock Items** screen (IN.20.25.00) is one of the most often used data entry forms of Acumatica ERP to export data. *Inventory ID* is the only primary key on the **Stock Items** screen:

* Inventory ID:		AAC	OMPUT01	Q		Prod	uct Workgro	up:			
Item Status:		Activ	e 🔻			Prod	uct Manager	:			
Description:	[	Acer	Laptop Computer								_
General Settings	Price/Cost	t Info	Warehouse Details	Vendor Details	Att	ributes	Packaging	Cross-F	Reference	Reple	nis
ITEM DEFAULTS	s					UNIT (	OF MEASUR	E			
Item Class:		ELEC	COMP - Electronics	& Computer D	R	* Base	e Unit:		EA		đ
Type:		Finis	hed Good	-		* Sale	s Unit:		EA	Q	đ
	[	ls a	Kit			* Purc	hase Unit:		EA	Q	đ
Valuation Meth	od:	Avera	ige	-		с	+	×			
* Tax Category:	[	TAXA	BLE - Taxable Goods	s and Servici D	R	*	From	Multiply/	Divide		6
* Posting Class:		ELE	- Electronics & Comp	outers D	R		Unit	munipiy/	DIVICE		
* Lot/Serial Clas	s:	NOT	RACKED - Not Trac	ked 🔎	R						
Auto-Increment	tal Value:										
WAREHOUSE D	EFAULTS										
Default Wareho	ouse:	WHO	LESALE - HQ Whole	esale Wareh 🔎	R						
Default Issue F	rom:	R1S1	- Row 1 Shelf 1	Q	R	PHYS	ICAL INVEN	TORY -			
Default Receip	t To:	REC	EIVING - Receiving	Q	R	PI C	ycle:				
						ABC	Code:				
									Fixed A	BC Co	de
						Move	ement Class				

To export records from a data entry form, your SOAP request must always begin with the ServiceCommands.Every[Key] command, where [Key] is to be replaced with primary key name.

# To export all stock items in a single web service call:

```
Screen context = new Screen();
context.CookieContainer = new System.Net.CookieContainer();
context.Url = "http://localhost/AcumaticaERP/Soap/IN202500.asmx";
context.Login(username, password);
try
```

C Revision Two HQ - Stock Items +

```
{
   Content stockItemsSchema = PX.Soap.Helper.GetSchema<Content>(context);
   Field lastModifiedField = new Field
    {
        ObjectName = stockItemsSchema.StockItemSummary.InventoryID.ObjectName,
        FieldName = "LastModifiedDateTime"
    };
    var commands = new Command[]
    {
        stockItemsSchema.StockItemSummary.ServiceCommands.EveryInventoryID,
       stockItemsSchema.StockItemSummary.InventoryID,
        stockItemsSchema.StockItemSummary.Description,
        stockItemsSchema.GeneralSettingsItemDefaults.ItemClass,
        stockItemsSchema.GeneralSettingsUnitOfMeasureBaseUnit.BaseUnit,
       lastModifiedField
   };
   var items = context.Export(commands, null, 0, false, false);
}
finally
{
   context.Logout();
}
```

With time amount of data in any ERP application tends to grow in size. If you will be exporting all records from your Acumatica ERP instance in a single web service call, very soon you might notice timeout errors. Increasing timeout is a possible one-time, but not very good long-term solution. Your best option to address this challenge is to export stock items in batches of several records.

# To export stock items in batches of 10 records:

```
Screen context = new Screen();
context.CookieContainer = new System.Net.CookieContainer();
context.Url = "http://localhost/AcumaticaERP/Soap/IN202500.asmx";
context.Login(username, password);
try
{
    Content stockItemsSchema = PX.Soap.Helper.GetSchema<Content>(context);
    Field lastModifiedField = new Field
    {
       ObjectName = stockItemsSchema.StockItemSummary.InventoryID.ObjectName,
       FieldName = "LastModifiedDateTime"
    };
    var commands = new Command[]
    {
        stockItemsSchema.StockItemSummary.ServiceCommands.EveryInventoryID,
        stockItemsSchema.StockItemSummary.InventoryID,
       stockItemsSchema.StockItemSummary.Description,
       stockItemsSchema.GeneralSettingsItemDefaults.ItemClass,
       stockItemsSchema.GeneralSettingsUnitOfMeasureBaseUnit.BaseUnit,
       lastModifiedField
    };
    var items = context.Export(commands, null, 10, false, false);
```

```
while (items.Length == 10)
    {
        var filters = new Filter[]
        {
            new Filter
            {
                Field = stockItemsSchema.StockItemSummary.InventoryID,
                Condition = FilterCondition.Greater,
                Value = items[items.Length - 1][0]
            }
        };
        items = context.Export(commands, filters, 10, false, false);
    }
}
finally
{
   context.Logout();
}
```

There are 2 main differences between the single call approach and the export in batches:

- topCount parameter of the Export command was always set to o in the single call approach
- when exporting records in batches, size of a batch is configured though the **topCount** parameter supplemented by the **Filter** array to request the next result set

Data Export from an Entry Form with a Composite Primary Key

The **Sales Orders** screen (SO.30.10.00) is a perfect example of a data entry form with a composite primary key. The primary key on the **Sales Orders** screen is composed by the **Order** *Type* and the **Order Number**.

C Revision	Two HQ	<ul> <li>Sales O</li> </ul>	rders ★							
• • +	T C	р <del>-</del> к	<	>	ACTIONS - R	EPO	RTS -			
★ Order Type: Order Nbr.:	SO SO003724	ァ ・Cu ァ・Lo	istomer: cation:	FDIT MAI	AMPA - Tampa Bay N - Primary Location	Food	Distributio	or e⊄	Order VAT E	red Qty Exempt
Status: * Date:	Completed	Cu * Pri	irrency: oject:	USD Cre X - N	) 1.00 • edit Hold Ion-Project Code.		EW BASE	E	VAT 1 Tax T Order	Faxable 'otal: r Total:
* Requested On: Customer Order:	1/1/2017 FDITAMPA	De 201	scription:	Foo	d distribution					
External Refer										
Document Details	Tax Details	Commissions	Financial Set	ttings	Payment Settings	Ship	oing Setti	ngs (	Discount	Details
c + 🖊	$\times$ A	LLOCATIONS	ADD INVOI	CE	ADD STOCK ITEM	P	D LINK	INVE	ENTORY	SUMN
B 🛈 🗋 *Branch	* Invento ID	ny Free Item	Warehous	Line	e Description		* UOM	(	Quantity	C Ship
> 0 🗅 VA	AAPOV	<u>v</u>	RETAIL	Po	weraid 32 Oz - lot nu	mb	EA	3,	880.00	3,8

The recommended 2-step strategy to export data from the **Sales Orders** screen or any other data entry form with a composite primary key via the Screen-Based API:

- on step 1 you request all types of orders previously created in your Acumatica ERP application
- 2nd step is to export orders of each type independently either in a single call or in batches

## To request all types of existing orders:

```
Screen context = new Screen();
context.CookieContainer = new System.Net.CookieContainer();
context.Url = "http://localhost/AcumaticaERP/Soap/SO301000.asmx";
context.Login(username, password);
try
{
    Content orderSchema = PX.Soap.Helper.GetSchema<Content>(context);
    var commands = new Command[]
    {
        orderSchema.OrderSummary.ServiceCommands.EveryOrderType,
        orderSchema.OrderSummary.OrderType,
    };
    var types = context.Export(commands, null, 1, false, false);
}
```

```
finally
{
    context.Logout();
}
```

In the SOAP call above, notice **topCount** parameter of the **Export** command set to 1. The purpose of this request is only to get all types of orders previously created in your Acumatica ERP application, not to export data.

# To export records of each type independently in batches:

```
Screen context = new Screen();
context.CookieContainer = new System.Net.CookieContainer();
context.Url = "http://localhost/AcumaticaERP/Soap/SO301000.asmx";
context.Login(username, password);
try
{
    Content orderSchema = PX.Soap.Helper.GetSchema<Content>(context);
    var commands = new Command[]
    {
       orderSchema.OrderSummary.ServiceCommands.EveryOrderType,
       orderSchema.OrderSummary.OrderType,
    };
    var types = context.Export(commands, null, 1, false, false);
    for (int i = 0; i < types.Length; i++)</pre>
    {
        commands = new Command[]
        {
            new Value
            {
                LinkedCommand = orderSchema.OrderSummary.OrderType,
                Value = types[i][0]
            },
            orderSchema.OrderSummary.ServiceCommands.EveryOrderNbr,
            orderSchema.OrderSummary.OrderType,
            orderSchema.OrderSummary.OrderNbr,
            orderSchema.OrderSummary.Customer,
            orderSchema.OrderSummary.CustomerOrder,
            orderSchema.OrderSummary.Date,
            orderSchema.OrderSummary.OrderedQty,
            orderSchema.OrderSummary.OrderTotal
        };
        var orders = context.Export(commands, null, 100, false, false);
        while (orders.Length == 100)
        {
            var filters = new Filter[]
            {
                new Filter
                {
                    Field = orderSchema.OrderSummary.OrderNbr,
                    Condition = FilterCondition.Greater,
                    Value = orders[orders.Length - 1][1]
                }
```

```
};
orders = context.Export(commands, filters, 100, false, false);
}
finally
{
    context.Logout();
}
```

The sample above demonstrates how to export all sales orders from Acumatica ERP in batches of 100 records. To export sales order of each type independently, your SOAP request must always begin with the value command, which determines the type of orders to be exported. After the Value command used to set first key value goes the serviceCommands.Every[Key] command, where [Key] is to be replaced with name of the second key.

## To export records of a specific type:

In case you need to export sales orders of a specific type, it's possible to explicitly define the type of orders with the value command in the beginning of your SOAP request followed by the single call approach or the export in batches.

To export all sales order of the *IN* type in one call:

```
Screen context = new Screen();
context.CookieContainer = new System.Net.CookieContainer();
context.Url = "http://localhost/AcumaticaERP/Soap/SO301000.asmx";
context.Login(username, password);
try
{
   Content orderSchema = PX.Soap.Helper.GetSchema<Content>(context);
   var commands = new Command[]
    {
       new Value
        {
           LinkedCommand = orderSchema.OrderSummary.OrderType,
            Value = "IN"
        },
        orderSchema.OrderSummary.ServiceCommands.EveryOrderNbr,
       orderSchema.OrderSummary.OrderType,
       orderSchema.OrderSummary.OrderNbr,
       orderSchema.OrderSummary.Customer,
       orderSchema.OrderSummary.CustomerOrder,
        orderSchema.OrderSummary.Date,
        orderSchema.OrderSummary.OrderedQty,
       orderSchema.OrderSummary.OrderTotal
   };
   var orders = context.Export(commands, null, 0, false, false);
}
finally
{
   context.Logout();
}
```

Read Exporting Records via Screen-Based API online: https://riptutorial.com/acumatica/topic/9288/exporting-records-via-screen-based-api

## Chapter 14: Extending List of Entities Supported by Tasks, Events and Activities

#### Introduction

In this topic you will learn how to extend the Related Entity Description field with a custom entity for Tasks, Events and Activities.

### Examples

Adding Test Work Orders to the Related Entity Description Field

Let's say you have already created the custom **Test Work Orders** screen to manage test work orders in your Acumatica ERP application:

	O Revisio	on Two HQ 🚽	Test V	Vork O	rder 🔺		ACTIVITIES F	ILES CUSTO	OMIZAT
	• • •	+ 🖬 🗘	+ K	<	> >	I			
	* ITWO Nbr.:	000003		Q	St	tatus:	Closed	Ŧ	
	* Purchase Ord	ler: 000102		Q	C	reated By:	admin		
	* Order Date:	2/6/2017	-		La	ast Modified On:	3/3/2017 12:30:48 F	PM	
	c + >	×  ↔  ×							
Ē	Status	Item ID	Descriptio	n		Manufacturer	Received Date	Received	Qty
	Closed	AALEGO500	Lego 500	piece set		DSFD-2324	1/18/2017	5.	00
	Cancelled	CONCRIB02	Graco Sty	ylus Class	sic Travel S	S 3243-FDEW5	56 1/16/2017	2.	00 3
>	Closed Short	ELEBOSE1	Bose Qui	et Comfor	t Noise Ca	BDS-432456	1/20/2017	20.	00 1

There is already NoteID field declared in the TestWorkOrder DAC, managed on the Test Work Orders screen:

```
[Serializable]
public class TestWorkOrder : IBqlTable
{
    ...
    #region NoteID
    public abstract class noteID : IBqlField { }
    [PXNote]
    public virtual Guid? NoteID { get; set; }
    #endregion
```

```
} ...
```

and ActivityIndicator property is set to *True* for the top-level PXForm container:

```
<px:PXFormView ID="form" runat="server" ActivityIndicator="true" DataSourceID="ds" Style="z-
index: 100" DataMember="ITWO" Width="100%" >
```

However, when new task, event or activity is created for a test work order, the **Related Entity Description** control is always empty:

	0 Revision	n Two HQ - 1	Fest Work O	order ★ [ >>>	] NOTES	ACTIVITIES	FILES	CUSTOMIZ	ZATION
	* ITWO Nbr.: * Purchase Orde * Order Date:	000003 r: 000102 2/6/2017	م م T	) Status: Created Last M Tasks & Activiti	d By: odified On: <b>es</b>	Closed admin 3/3/2017 12:4	43:43 PM	<b>•</b>	
ē >	Status Closed	Item ID De AALEGO500 Le	escription ego 500 piece s	C ADD T	ASK ADD	EVENT AI	DD ACTIVIT	Υ •	
	Task - Mozilla Fi localhost/Sta Revision Tw SAVE & CLOS	refox ickOverflow/(W(10 ro HQ Task E 🖬 🔊	000))/pages/cr/c	cr306020.aspx? C PLETE COI	timeStamp= ) NOTES MPLETE & I	5bb8b94af373 FILES CUS FOLLOW-UP	- 3d81d31a9 TOMIZATION	D Pac8d1825 HELP	× 5164
	Details Related * Summary: Start Date: Due Date: Completion (%) Workgroup: Owner: Remind at (Dat	Activities Related	I Tasks Tasks Paker Maxwell P 3	Internal Select En Type: Entity:	Priority tity * * 000003	r: Nor	rmal C/	,∽ ∧ ANCEL	
	Related Entity * Project: Project Task:	X - Non-Project	م Code. ک	2	Billable	e Overti 00:	00		

To add the **Test Work Order** entity to the **Related Entity Description** selector, you should complete the following steps:

1. For the PXNoteAttribute on TestWorkOrder.NoteID field, set ShowInReferenceSelector property to *True* and define BQL expression to select data records displayed in the Entity lookup:

[PXNote(

```
ShowInReferenceSelector = true,
Selector = typeof(Search<TestWorkOrder.orderNbr>))]
public virtual Guid? NoteID { get; set; }
```

2. Decorate the TestWorkOrder DAC with the PXCacheNameAttribute and the

PXPrimaryGraphAttribute:

```
[PXLocalizable]
public static class Messages
{
    public const string Opportunity = "Test Work Order";
}
[Serializable]
[PXCacheName(Messages.Opportunity)]
[PXPrimaryGraph(typeof(TestWorkOrderEntry))]
public class TestWorkOrder : IBqlTable
{
    ...
}
```

The PXCacheName attribute defines user-friendly name for the TestWorkOrder DAC (*Test Work Order* in this case), which will be available in the **Type** dropdown. The PXPrimaryGraph attribute determines the entry page where a user is redirected for editing a test work order, which is the **Test Work Orders** screen in the given example.

3. Decorate some TestWorkOrder fields with the PXFieldDescriptionAttribute. Those field values will be concatenated into a single text label, representing the referenced test work order inside the Related Entity Description field:

```
...
[PXFieldDescription]
public virtual string OrderNbr { get; set; }
...
[PXFieldDescription]
public virtual String Status { get; set; }
...
[PXFieldDescription]
public virtual string POOrderNbr { get; set; }
```

- 4. Define the list of columns displayed in the **Entity** lookup by choosing one of the approaches below:
  - a. Use the PXNoteAttribute.FieldList property (gets the highest priority):

```
public abstract class noteID : IBqlField { }
[PXNote(
    ShowInReferenceSelector = true,
    Selector = typeof(Search<TestWorkOrder.orderNbr>),
    FieldList = new Type[]
    {
        typeof(TestWorkOrder.orderNbr),
```

```
typeof(TestWorkOrder.orderDate),
    typeof(TestWorkOrder.status),
    typeof(TestWorkOrder.poOrderNbr)
})]
public virtual Guid? NoteID { get; set; }
```

b. Borrow the list of columns defined for the OrderNbr lookup:

```
public abstract class orderNbr : IBqlField { }
[PXDBString(15, IsKey = true, IsUnicode = true, InputMask = ">CCCCCCCCCCCCC")]
[PXDefault()]
[PXUIField(DisplayName = "ITWO Nbr.", Visibility = PXUIVisibility.SelectorVisible)]
[PXSelector(typeof(Search<TestWorkOrder.orderNbr>),
    typeof(TestWorkOrder.orderNbr),
    typeof(TestWorkOrder.orderDate),
    typeof(TestWorkOrder.status),
    typeof(TestWorkOrder.poOrderNbr))]
[PXFieldDescription]
public virtual string OrderNbr { get; set; }
```

c. Show all TestWorkOrder fields with Visibility set to PXUIVisibility.SelectorVisible:

```
...
[PXUIField(DisplayName = "ITWO Nbr.", Visibility = PXUIVisibility.SelectorVisible)]
public virtual string OrderNbr { get; set; }
...
[PXUIField(DisplayName = "Order Date", Visibility = PXUIVisibility.SelectorVisible)]
public virtual DateTime? OrderDate { get; set; }
...
[PXUIField(DisplayName = "Status", Visibility = PXUIVisibility.SelectorVisible)]
public virtual String Status { get; set; }
...
[PXUIField(DisplayName = "Purchase Order", Visibility = PXUIVisibility.SelectorVisible)]
public virtual string POOrderNbr { get; set; }
```

After you completed the 4 steps above, **Test Work Orders** should be fully supported by the **Related Entity Description** field on Tasks, Events and Activities

G Revision	TwoHQ - TestWo ∎ D - K	ork Order 🔺		ACTIVITIES	FILES CUS	TOMIZATION
<ul> <li>ITWO Nbr.:</li> <li>Purchase Order:</li> <li>Order Date:</li> </ul>	000003 000102 2/6/2017 -	א א ב ב ב	tatus: reated By: ast Modified On:	Closed admin 3/3/2017 12:4	▼ 3:43 PM	
C + X	I←I     I       Im ID     Description       ALEGO500     Lego 500 pin		D TASK ADD	EVENT ADD		
Task - Mozilla Firef          Iocalhost/Stack         Revision Two	ox :Overflow/(W(10000))/pag HQ Task	es/cr/cr306020.a	aspx?timeStamp=	=5bb8b94af373 🗋 NG	d81d31a9ac8d DTES FILES	175642213 CUSTOM
SAVE & CLOSE Details Related A	ctivities Related Tasks	COMPLETE	COMPLETE &	FOLLOW-UP	CANCEL	
* Summary: Start Date: Due Date: Completion (%):	3/3/2017		nal Priorit ntity	y: Norr	nal >	- -
Workgroup: Owner:	EP00000002 - Baker Maxv	א Type: vell א Entity:	* Test Work Or * 000003 Select - Entity	der 🔻	م	
Remind at (Dat Related Entity	000003, Closed, 000102 X - Non-Project Code	<i>₽ €</i>		C  ↔	Statue	Purchase O
VISUAL -	▲ Paragraph ▼	в <u>и</u>	<ul> <li>&gt; 000003</li> <li>&gt; 000004</li> <li>&gt; 000005</li> <li>&gt; 000006</li> </ul>	2/6/2017 2/6/2017 2/15/2017 2/21/2017	Closed Locked In Progress Open	000102 000146 000111 000154
					к	< >

Read Extending List of Entities Supported by Tasks, Events and Activities online: https://riptutorial.com/acumatica/topic/9342/extending-list-of-entities-supported-by-tasks--eventsand-activities

# Chapter 15: Filtering with multiple value with only one selector

#### Introduction

Here is a way of having multiple value inside of a selector in order to filter a grid.

### **Examples**

**Retrieving Sales Order for multilple customer** 

When trying to filter some record using multiple value in a selector. First you must use the px:PXMultiSelector in the aspx page instead of the normal px:PXSelector. Then after you must create yourself a graph containing at least three views and a view delegate. you will also need at least a basic unbound DAC.

Here is an sample page with the px:PXMultiSelector:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPages/FormDetail.master" AutoEventWireup="true"
ValidateRequest="false" CodeFile="TT000000.aspx.cs" Inherits="Page_TT000000" Title="Untitled
Page" %>
<%@ MasterType VirtualPath="~/MasterPages/FormDetail.master" %>
<asp:Content ID="cont1" ContentPlaceHolderID="phDS" runat="Server">
<px:PXDataSource ID="ds" runat="server" Visible="True" Width="100%"
    TypeName="MultiSelector.MultiInquiry"
    PrimaryView="MasterView">
    <CallbackCommands>
    </CallbackCommands>
</px:PXDataSource>
</asp:Content>
<asp:Content ID="cont2" ContentPlaceHolderID="phF" runat="Server">
<pr:PXFormView ID="form" runat="server" DataSourceID="ds" DataMember="MasterView" Width="100%"
Height="100px" AllowAutoHide="false">
    <Template>
        <px:PXMultiSelector ID="edInventoryID" runat="server" Width="100%" DataSourceID="ds"
DataField="Customer" CommitChanges="True"></px:PXMultiSelector>
    </Template>
</px:PXFormView>
</asp:Content>
<asp:Content ID="cont3" ContentPlaceHolderID="phG" runat="Server">
<pr:PXGrid ID="grid" runat="server" DataSourceID="ds" Width="100%" Height="150px"
SkinID="Details" AllowAutoHide="false">
    <Levels>
        <px:PXGridLevel DataMember="DetailsView">
            <Columns>
                <pr:PXGridColumn DataField="OrderType" Width="70"></pr:PXGridColumn>
                <px:PXGridColumn DataField="OrderNbr" Width="200"></px:PXGridColumn>
                <px:PXGridColumn DataField="OrderDesc" Width="100"></px:PXGridColumn>
                <px:PXGridColumn DataField="CustomerOrderNbr" Width="100"></px:PXGridColumn>
```
Here is the sample graph with the views and the delegate.

```
public class MultiInquiry : PXGraph<MultiInquiry>
{
   public PXCancel<MasterTable> Cancel;
   public PXFilter<MasterTable> MasterView;
   public PXSelect<SOOrder> DetailsView;
   public PXSelectJoin<SOOrder, LeftJoin<BAccount, On<SOOrder.customerID,
Equal<BAccount.bAccountID>>>, Where<BAccount.acctCD, In<Required<BAccount.acctCD>>>> Orders2;
   protected virtual IEnumerable detailsView()
    {
       var list = new List<SOOrder>();
       var customers = MasterView.Current.Customer;
        if (customers != null)
            List<string> customerList = new List<string>();
            customerList.AddRange(customers.Split(new string[] { "; " },
StringSplitOptions.None));
            object[] val = new object[] { customerList.ToArray() };
            foreach (PXResult<SOOrder> res in Orders2.Select(val))
                SOOrder order = res;
                list.Add(order);
            }
        }
       return list;
    }
}
```

To this we add the DAC containing the definition for the field used in the MultiSelector and the constant for only selecting customer accounts.

```
[Serializable]
public class MasterTable : IBqlTable
{
    #region InventoryID
    public abstract class customer : IBqlField { }
    [PXString()]
    [PXUIField(DisplayName = "Customer")]
    [PXSelector(typeof(Search<BAccount.acctCD, Where<BAccount.type,
Equal<CustomerType>>>), ValidateValue = false)]
    public virtual string Customer { get; set; }
```

```
#endregion
}
public class CustomerType : Constant<string> { public CustomerType() : base("CU") { } }
```

And the result for this example could be something like this :

С	usto	omer	-	ABCHOLDING ×	ABCSTUDIOS ×	ABARTEN	NDE ×	
( 8	C		• X *Order	↔  🗵 Order Nbr.	Desc	ription	Customer	Status
			Туре				Order	
>	0		CS	001558	War	rehouse		Complete
	U		CS	001559	War	rehouse		Complete
	0		CS	001569	Wa	rehouse		Complete
	0		CS	001570	War	rehouse		Complete
	0		CS	001580	Wa	rehouse		Complete
	0		CS	001584	Wa	rehouse		Complete

Read Filtering with multiple value with only one selector online: https://riptutorial.com/acumatica/topic/10707/filtering-with-multiple-value-with-only-one-selector

# **Chapter 16: Freight Calculation**

#### Introduction

Acumatica ERP enables you to manage freight to better control any additional costs and revenues on sales transactions. The freight amount you charge your customers may include not only the freight your company is charged by carriers, but also insurance, handling and packaging fees defined by your shipping terms and premium freight.

#### Examples

#### **Overriding Freight Amount in Shipment and Invoice**

Out of the box Acumatica allows to create and maintain the list of shipping terms in the system. Shipping terms are used to define the shipping, packaging and handling costs, depending on the shipment amount.

In this example I will show how to calculate freight amount for a shipment based on sales order amount, which would allow users to create multiple shipments per sales order with same shipping terms automatically applied to all shipments.

## **FreightCalculator**

The FreightCalculator class is responsible for calculation of Freight Cost and Freight Terms. For the purpose of this example, our interest will be only around the GetFreightTerms method:

```
public class FreightCalculator
{
          ...
          protected virtual ShipTermsDetail GetFreightTerms(string shipTermsID, decimal? lineTotal)
          {
                return PXSelect<ShipTermsDetail,
                Where<ShipTermsDetail.shipTermsID, Equal<Required<SOOrder.shipTermsID>>,
                And<ShipTermsDetail.breakAmount, LessEqual<Required<SOOrder.lineTotal>>>>,
                OrderBy<Desc<ShipTermsDetail.breakAmount>>>.Select(graph, shipTermsID, lineTotal);
        }
        ...
    }
```

Both the **Sales Orders** and the **Shipments** screens utilize FreightCalculator class to calculate freight amount based on sales order's and shipment's amount respectively:

#### **Sales Orders**

```
public class SOOrderEntry : PXGraph<SOOrderEntry, SOOrder>, PXImportAttribute.IPXPrepareItems
{
    . . .
    public virtual FreightCalculator CreateFreightCalculator()
    {
       return new FreightCalculator(this);
    }
    . . .
   protected virtual void SOOrder_RowUpdated(PXCache sender, PXRowUpdatedEventArgs e)
    {
        . . .
       PXResultset<SOLine> res = Transactions.Select();
       FreightCalculator fc = CreateFreightCalculator();
        fc.CalcFreight<SOOrder, SOOrder.curyFreightCost, SOOrder.curyFreightAmt>(sender,
(SOOrder)e.Row, res.Count);
        . . .
    }
    . . .
}
```

#### Shipments

```
public class SOShipmentEntry : PXGraph<SOShipmentEntry, SOShipment>
{
    • • •
   protected virtual FreightCalculator CreateFreightCalculator()
    {
       return new FreightCalculator(this);
    }
    . . .
    protected virtual void SOShipment_RowUpdated(PXCache sender, PXRowUpdatedEventArgs e)
    {
        . . .
        PXResultset<SOShipLine> res = Transactions.Select();
        . . .
        FreightCalculator fc = CreateFreightCalculator();
        fc.CalcFreight<SOShipment, SOShipment.curyFreightCost,</pre>
SOShipment.curyFreightAmt>(sender, (SOShipment)e.Row, res.Count);
        . . .
    }
    . . .
}
```

### **Overriding Freight Amount**

https://riptutorial.com/

To customize how Acumatica calculates freight amount on the **Shipments** screen I will declare FreightCalculatorCst class inherited from FreightCalculator and override GetFreightTerms method:

```
public class FreightCalculatorCst : FreightCalculator
{
   public FreightCalculatorCst(PXGraph graph)
        : base(graph)
    {
    }
   protected override ShipTermsDetail GetFreightTerms(string shipTermsID, decimal? lineTotal)
    {
        if (graph is SOShipmentEntry)
        {
            var shipmentEntry = graph as SOShipmentEntry;
           int orderCount = 0;
           decimal? lineTotalTemp = null;
           foreach (PXResult<SOOrderShipment, SOOrder, CurrencyInfo, SOAddress, SOContact,
SOOrderType> orderRec in
                shipmentEntry.OrderList.SelectWindowed(0, 2))
            {
                orderCount++;
                SOOrder order = (SOOrder)orderRec;
                if (orderCount == 1)
                   lineTotalTemp = order.LineTotal;
                else
                   break;
            }
            if (orderCount == 1)
            {
               lineTotal = lineTotalTemp;
            }
        }
       return base.GetFreightTerms(shipTermsID, lineTotal);
   }
}
```

After that I will implement an extension for the <code>soshipmentEntry</code> BLC and override CreateFreightCalculator method to replace <code>FreightCalculator</code> with my custom <code>FreightCalculatorCst</code> class on the Shipments screen:

```
public class SOShipmentEntryExt : PXGraphExtension<SOShipmentEntry>
{
    [PXOverride]
    public FreightCalculator CreateFreightCalculator()
    {
        return new FreightCalculatorCst(Base);
    }
}
```

### Understanding implementation of the

# FreightCalculatorCst class in the sample above

In the overridden GetFreightTerms method I will use amount from sales order instead of shipment amount to invoke base GetFreightTerms method and receive shipping terms:

```
foreach (PXResult<S00rderShipment, S00rder, CurrencyInfo, S0Address, S0Contact, S00rderType>
orderRec in
    shipmentEntry.OrderList.SelectWindowed(0, 2))
{
    orderCount++;
    S00rder order = (S00rder)orderRec;
    if (orderCount == 1)
        lineTotalTemp = order.LineTotal;
    else
        break;
}
if (orderCount == 1)
{
    lineTotal = lineTotalTemp;
}
```

Obviously, it's only possible to use sales order amount to calculate freight amount for shipments, which fulfill only 1 order. If one shipment fulfills several orders, we'd have to follow base product behavior and calculate freight amount based on shipment amount. To check the number of orders shipment fulfills, I used SelectWindowed method on the OrderList data view and requested first 2 orders fulfilled by the current shipment. I could have requested all orders fulfilled by the shipment, but this would take significantly more time to execute and return way to many records than needed to verify if sales order amount can be used instead of shipment amount to calculate freight.

Read Freight Calculation online: https://riptutorial.com/acumatica/topic/9044/freight-calculation

## **Chapter 17: Modifications to Base Data Views**

#### Introduction

This topic is intended to demonstrate various patterns and practices available to modify base data views in Acumatica.

#### **Examples**

APInvoiceEntry BLC: add additional restriction to poReceiptLinesSelection data view

To add additional restriction to the **poReceiptLinesSelection** data view, you should invoke select method on base view and inspect each item in returned <code>PXResultSet</code> independently to decide if it complies with additional restriction(s):

```
public class APInvoiceEntryExt : PXGraphExtension<APInvoiceEntry>
{
    [PXCopyPasteHiddenView]
    public PXSelect<APInvoiceEntry.POReceiptLineAdd> poReceiptLinesSelection;
    public virtual IEnumerable POReceiptLinesSelection()
    {
        foreach (var record in Base.poReceiptLinesSelection.Select())
        {
            // Additional restriction goes here
            if (true == true)
            {
                yield return record;
            }
       }
   }
}
```

This approach perfectly works with the **poReceiptLinesSelection** data view, due to lack of paging and aggregation in the implementation of base view. To compose result set, **poReceiptLinesSelection** view requests necessary data from database and performs all calculations on the application side.

```
if (doc.DocType != APDocType.Invoice && doc.DocType != APDocType.DebitAdj)
            yield break;
        string poReceiptType = (doc.DocType == APDocType.Invoice) ? POReceiptType.POReceipt :
POReceiptType.POReturn;
        HashSet<APTran> usedRecceiptLine = new HashSet<APTran>(new POReceiptLineComparer());
        HashSet<APTran> unusedReceiptLine = new HashSet<APTran> (new POReceiptLineComparer());
        foreach (APTran aPTran in Transactions.Cache.Inserted)
            if (aPTran.ReceiptNbr != null && aPTran.ReceiptType != null &&
aPTran.ReceiptLineNbr != null)
               usedRecceiptLine.Add(aPTran);
        }
        foreach (APTran aPTran in Transactions.Cache.Deleted)
            if (aPTran.ReceiptNbr != null && aPTran.ReceiptType != null &&
aPTran.ReceiptLineNbr != null && Transactions.Cache.GetStatus(aPTran) !=
PXEntryStatus.InsertedDeleted)
                if (!usedRecceiptLine.Remove(aPTran))
                    unusedReceiptLine.Add(aPTran);
        }
        foreach (APTran aPTran in Transactions.Cache.Updated)
            APTran originAPTran = new APTran();
            originAPTran.ReceiptNbr =
(String)Transactions.Cache.GetValueOriginal<APTran.receiptNbr>(aPTran);
           originAPTran.ReceiptType =
(String)Transactions.Cache.GetValueOriginal<APTran.receiptType>(aPTran);
           originAPTran.ReceiptLineNbr =
(Int32?)Transactions.Cache.GetValueOriginal<APTran.receiptLineNbr>(aPTran);
            if (originAPTran.ReceiptNbr != null && originAPTran.ReceiptType != null &&
originAPTran.ReceiptLineNbr != null)
            {
                if (!usedRecceiptLine.Remove(originAPTran))
                    unusedReceiptLine.Add(originAPTran);
            if (aPTran.ReceiptNbr != null && aPTran.ReceiptType != null &&
aPTran.ReceiptLineNbr != null)
            {
                if (!unusedReceiptLine.Remove(aPTran))
                   usedRecceiptLine.Add(aPTran);
            }
        }
        foreach (LinkLineReceipt item in PXSelect<LinkLineReceipt,
            Where<LinkLineReceipt.vendorID, Equal<Current<APInvoice.vendorID>>,
            And<LinkLineReceipt.vendorLocationID, Equal<Current<APInvoice.vendorLocationID>>,
            And<LinkLineReceipt.receiptCuryID, Equal<Current<APInvoice.curyID>>,
            And<LinkLineReceipt.receiptType, Equal<Required<POReceipt.receiptType>>,
           And<Where<LinkLineReceipt.orderNbr, Equal<Current<POReceiptFilter.orderNbr>>,
Or<Current<POReceiptFilter.orderNbr>, IsNull>>>
           >>>>.SelectMultiBound(this, new object[] { doc }, poReceiptType))
        {
           APTran aPTran = new APTran();
            aPTran.ReceiptType = item.ReceiptType;
```

```
aPTran.ReceiptNbr = item.ReceiptNbr;
aPTran.ReceiptLineNbr = item.ReceiptLineNbr;
if (!usedRecceiptLine.Contains(aPTran))
yield return (PXResult<POReceiptLineAdd,
POReceipt>)ReceiptLineAdd.Select(item.ReceiptNbr, item.ReceiptType, item.ReceiptLineNbr);
}
foreach (APTran item in unusedReceiptLine)
{
yield return (PXResult<POReceiptLineAdd,
POReceipt>)ReceiptLineAdd.Select(item.ReceiptNbr, item.ReceiptType, item.ReceiptLineNbr);
}
}
....
```

Read Modifications to Base Data Views online: https://riptutorial.com/acumatica/topic/9101/modifications-to-base-data-views

# Chapter 18: Modifications to Contact and Address Info through Code

#### Introduction

In this topic, you will learn how to modify Contact and Address information through code on different screens inside Acumatica.

#### Examples

Specify Contact and Address information for a new Employee

To specify Contact and Address info for an Employee, you should always invoke select() method
on the **Contact** and **Address** data views prior to assigning any field values. It is also
recommended to assign the result of select() method to the **Contact** and **Address** data views' *Current* property to guarantee that your code modifies the current record in **Contact** and **Address**PXCache respectively.

```
EmployeeMaint employeeMaintGraph = PXGraph.CreateInstance<EmployeeMaint>();
EPEmployee epEmployeeRow = new EPEmployee();
epEmployeeRow.AcctCD = "EMPLOYEE1";
epEmployeeRow = employeeMaintGraph.Employee.Insert(epEmployeeRow);
Contact contactRow = employeeMaintGraph.Contact.Current = employeeMaintGraph.Contact.Select();
contactRow.FirstName = "John";
contactRow.LastName = "Green";
employeeMaintGraph.Contact.Update(contactRow);
Address addressRow = employeeMaintGraph.Address.Current = employeeMaintGraph.Address.Select();
addressRow.CountryID = "US";
addressRow = employeeMaintGraph.Address.Update(addressRow);
addressRow.State = "DC";
employeeMaintGraph.Address.Update(addressRow);
epEmployeeRow.VendorClassID = "EMPSTAND";
epEmployeeRow.DepartmentID = "FINANCE";
employeeMaintGraph.Employee.Update(epEmployeeRow);
employeeMaintGraph.Actions.PressSave();
```

When inserting a new Employee, employeeMaintGraph.Contact.Current will always return the main contact record as the contact record gets automatically inserted into the cache and therefore becomes available via the *Current* property of PXCache/Data View. The use of select() method is a little more generic since it will work in all possible scenarios, whether you need to insert new Employee or update an existing one.

**Override Bill-To Contact and Bill-To Address Info for a Customer** 

When you need to override Bill-To Contact and Bill-To Address info for a Customer, the very first step is to set correct values for the *IsBillContSameAsMain* and *IsBillSameAsMain* properties of the **Customer** DAC. Don't forget to invoke <code>Update()</code> method on the **Customer** cache right after you updated *IsBillContSameAsMain* or *IsBillSameAsMain* property to commit the current *Same as Main* field value into the cache.

Your next step is to invoke <code>Select()</code> method on the **BillContact** and **BillAddress** data views prior to assigning any field values. It is also recommended to assign the result of <code>Select()</code> method to the **BillContact** and **BillAddress** data views' *Current* property to guarantee that your code modifies the current record in **Contact** and **Address** PXCache respectively.

```
public class CustomerMaintExt : PXGraphExtension<CustomerMaint>
{
   public PXAction<Customer> UpdateBillingAddress;
   [PXButton(CommitChanges = true)]
    [PXUIField(DisplayName = "Update Bill-To Info")]
   protected void updateBillingAddress()
    {
        Customer currentCustomer = Base.BAccount.Current;
        if (currentCustomer.IsBillContSameAsMain != true)
        {
            currentCustomer.IsBillContSameAsMain = true;
           Base.BAccount.Update(currentCustomer);
        }
        else
        {
            currentCustomer.IsBillContSameAsMain = false;
            Base.BAccount.Update(currentCustomer);
            Contact billContact = Base.BillContact.Current = Base.BillContact.Select();
           billContact.FullName = "ABC Holdings Inc";
           billContact.Phone1 = "+1 (212) 532-9574";
           Base.BillContact.Update(billContact);
        }
        if (currentCustomer.IsBillSameAsMain != true)
        {
           currentCustomer.IsBillSameAsMain = true;
           Base.CurrentCustomer.Update(currentCustomer);
        }
        else
            currentCustomer.IsBillSameAsMain = false;
           Base.CurrentCustomer.Update(currentCustomer);
           Address billAddress = Base.BillAddress.Current = Base.BillAddress.Select();
           billAddress.AddressLine1 = "65 Broadway";
           billAddress.AddressLine2 = "Office Suite 187";
           billAddress.City = "New York";
            billAddress.CountryID = "US";
           billAddress = Base.BillAddress.Update(billAddress);
           billAddress.State = "NY";
           billAddress.PostalCode = "10004";
           Base.BillAddress.Update(billAddress);
        }
        Base.Actions.PressSave();
```

### }

#### Override Bill-To Contact and Bill-To Address Info for a Sales Order

To specify Bill-To Contact and Bill-To Address info for a Sales Order, you should always first invoke select() method on the Billing\_Contact and Billing\_Address data views prior to assigning any field values. It is also recommended to assign the result of select() method to the Billing\_Contact and Billing\_Address data views' Current property to guarantee that your code modifies the current record in SOBillingContact and SOBillingAddress PXCache respectively.

When you need to override Bill-To Contact and Address info for a Sales Order, set correct values for the *OverrideContact* and *OverrideAddress* properties on the **SOBillingContact** DAC and the **SOBillingAddress** DAC. Don't forget to invoke <code>Update()</code> method on the **SOBillingContact** and **SOBillingAddress** caches right after you updated *OverrideContact* and *OverrideAddress* properties to commit the changes. Once that's done, you can go ahead and specify Bill-To Contact and Address info for a Sales Order.

```
public class SOOrderEntryExt : PXGraphExtension<SOOrderEntry>
{
   public PXAction<SOOrder> UpdateBillingAddress;
    [PXButton(CommitChanges = true)]
    [PXUIField(DisplayName = "Update Bill-To Info")]
   protected void updateBillingAddress()
    {
        SOBillingContact contact = Base.Billing_Contact.Current =
Base.Billing_Contact.Select();
       if (contact.OverrideContact == true)
        {
            contact.OverrideContact = false;
           Base.Billing_Contact.Update(contact);
        }
        else
        {
           contact.OverrideContact = true;
            contact = Base.Billing_Contact.Update(contact);
            if (contact == null)
            {
                contact = Base.Billing_Contact.Current;
            }
            contact.Phone1 = "+1 (908) 643-0281";
            contact.Email = "sales@usabartend.con";
            Base.Billing_Contact.Update(contact);
        }
        SOBillingAddress address = Base.Billing_Address.Current =
Base.Billing_Address.Select();
        if (address.OverrideAddress == true)
        {
            address.OverrideAddress = false;
           Base.Billing_Address.Update(address);
        }
        else
        {
            address.OverrideAddress = true;
```

```
address = Base.Billing_Address.Update(address);
           if (address == null)
            {
               address = Base.Billing_Address.Current;
            }
            address.AddressLine1 = "201 Lower Notch Rd";
           address.AddressLine2 = "Office Suite 1936";
           address.City = "Little Falls";
           address.CountryID = "US";
           address = Base.Billing_Address.Update(address);
           address.State = "NJ";
           address.PostalCode = "07425";
           Base.Billing_Address.Update(address);
        }
       Base.Actions.PressSave();
  }
}
```

Read Modifications to Contact and Address Info through Code online:

https://riptutorial.com/acumatica/topic/10617/modifications-to-contact-and-address-info-through-code

# Chapter 19: Modifying Items in a Dropdown List

#### Introduction

In this topic you will learn how to modify field attributes inherited from the PXStringList or PXIntList attributes. The demonstrated approach will make sure to not break functionality of the base Acumatica ERP product and require minimal maintenance, if any, while upgrading your customizations to a newer version of Acumatica.

#### Remarks

In all samples above, you made changes to both the \_AllowedValues and \_AllowedLabels arrays. If your task is to modify only label (external value) of a drop-down item, consider using Translation Dictionaries. For more information on Translation Dictionaries see Acumatica ERP Documentation

#### Examples

**Modifying Marital Statuses** 

In this example you will be modifying the **Marital Status** drop-down list found on the **Contacts** form (CR302000):

C Revision Two H	IQ - Contact	s ★ M	y Contacts	×				
SAVE & CLOSE		+ 🖬	¢-к	<	> >I	ACTI	ONS -	
Contact ID:	Air Jane		۵	Wo	rkaroup:		Execut	ive
Type:	Contact		-	0	nor:		Baker	Maxwell Mr
Type.				000	ner.		Dater	
Details Additional Info At	tributes Activities	Relations	Opportunities	Cases	Campaigns	Marketir	ng Lists	Notification
COMMON				PHO	то			
Gender:			-	Se	lect an image	to uploa	d.	
Marital Status:			+					
Spouse/Partner Name:	Single							
LEAD HISTORY	Married					1		
Source:	Widowed							
Campaign ID:	Indened		_					
Status							4	
Reason:								
Converted By:								
Qualification Date:								
SYNCHRONIZATION					г	)rac and (	drop the	image here
	Synchronize				L	and and	arop the	inage nere

#### To add new items to the PXStringListAttribute successor

The best way to extend drop-down items hard-coded inside an attribute inherited from the PXStringList or PXIntList attribute is by increasing size of the \_AllowedValues and \_AllowedLabels arrays in the constructor of your custom field attribute:

```
[PXLocalizable(Messages.Prefix)]
public static class MaritalStatusesMessages
{
    public const string CommonLaw = "Living common law";
    public const string Separated = "Separated (not living common law)";
    public const string DivorcedNoCommonLaw = "Divorced (not living common law)";
    public const string NeverMarried = "Never Married";
}
public class MaritalStatusesCstlAttribute : MaritalStatusesAttribute
{
    public const string CommonLaw = "L";
    public const string Separated = "P";
    public const string NeverMarried = "N";
```

```
public MaritalStatusesCstlAttribute()
{
    Array.Resize(ref _AllowedValues, _AllowedValues.Length + 3);
    _AllowedValues[_AllowedValues.Length - 3] = CommonLaw;
    _AllowedValues[_AllowedValues.Length - 2] = Separated;
    _AllowedValues[_AllowedValues.Length - 1] = NeverMarried;
    Array.Resize(ref _AllowedLabels, _AllowedLabels.Length + 3);
    _AllowedLabels[_AllowedLabels.Length - 3] = MaritalStatusesMessages.CommonLaw;
    _AllowedLabels[_AllowedLabels.Length - 2] = MaritalStatusesMessages.Separated;
    _AllowedLabels[_AllowedLabels.Length - 1] = MaritalStatusesMessages.NeverMarried;
}
```

In the sample above, you increased size of the \_AllowedValues and \_AllowedLabels arrays to add 3 additional items to the **Marital Status** drop-down list. Internal values, stored in the \_AllowedValues array, will be assigned to DAC fields and saved in database, and external values from the \_AllowedValues array represent internal values in the UI.

To test the results, in the Contact DAC extension, decorate MaritalStatus field with the

MaritalStatusesCst1Attribute:

```
public class ContactExt : PXCacheExtension<Contact>
{
     [PXRemoveBaseAttribute(typeof(MaritalStatusesAttribute))]
     [PXMergeAttributes(Method = MergeMethod.Append)]
     [MaritalStatusesCst1]
     public string MaritalStatus { get; set; }
}
```

Now there are 7 items in the Marital Status drop-down list:

0 F	SAVE & CLOS	oHQ ≁ ( ⊫ ⊟	Contact:	s ★ ► ∎	р• к				
Cont	act ID:	Air, Ja	ine		Q				
Туре		Conta	Contact						
		Activ	/0						
Details	Additional Info	Attributes	Activities	Relations	Opportunities				
Geno	fon	-			*				
Spou	ise/Partner Nam	e Single	<u>(</u>		- 1				
LEAD	HISTORY	Divorc	Married Divorced						
Sour	ce:	Widow	ved		_				
Cam	paign ID:	Living	common la	W					
Statu	15:	Separa	Separated (not living common law)						
Reas	ion:	- TOYON	- Contraction		_				

# To remove items declared in the PXStringListAttribute successor

https://riptutorial.com/

To remove specific drop-down item, that was hard-coded inside an attribute inherited from the PXStringList or PXIntList attribute, you need to decrease size of the \_AllowedValues and \_AllowedLabels arrays in the constructor of your custom field attribute:

```
public class MaritalStatusesCst2Attribute : MaritalStatusesCst1Attribute
{
    public MaritalStatusesCst2Attribute()
    {
        string[] allowedValues = new string[_AllowedValues.Length - 1];
        string[] allowedLabels = new string[_AllowedLabels.Length - 1];
        Array.Copy(_AllowedValues, 1, allowedValues, 0, _AllowedValues.Length - 1);
        Array.Copy(_AllowedLabels, 1, allowedLabels, 0, _AllowedValues.Length - 1);
        _AllowedValues = allowedValues;
        _AllowedLabels = allowedLabels;
    }
}
```

In the sample above, you decreased size of the \_AllowedValues and \_AllowedLabels arrays to remove *Single* item from the Marital Status drop-down list.

To test the results, in the Contact DAC extension, decorate MaritalStatus field with the

MaritalStatusesCst2Attribute:

```
public class ContactExt : PXCacheExtension<Contact>
{
     [PXRemoveBaseAttribute(typeof(MaritalStatusesAttribute))]
     [PXMergeAttributes(Method = MergeMethod.Append)]
     [MaritalStatusesCst2]
     public string MaritalStatus { get; set; }
}
```

Now there are only 6 items: 3 original and 3 custom - in the Marital Status drop-down list:



#### To replace items declared in the PXStringListAttribute

#### successor

To replace specific drop-down item, originally hard-coded inside an attribute inherited from the PXStringList or PXIntList attribute, you need to update appropriate value in the \_AllowedValues and \_AllowedLabels arrays in the constructor of your custom field attribute:

```
public class MaritalStatusesCst3Attribute : MaritalStatusesCst2Attribute
{
    public const string DivorcedNoCommonLaw = "V";
    public MaritalStatusesCst3Attribute()
    {
        __AllowedValues[Array.IndexOf(_AllowedValues, Divorced)] = DivorcedNoCommonLaw;
        __AllowedLabels[Array.IndexOf(_AllowedLabels, Messages.Divorced)] =
MaritalStatusesMessages.DivorcedNoCommonLaw;
    }
}
```

In the sample above, you replaced *D* - *Divorced* item with *V* - *Divorced* (not living common law) in the \_AllowedValues and \_AllowedLabels arrays respectively. By doing so, we replace both internal and external values of a drop-down item.

To test the results, in the Contact DAC extension, decorate MaritalStatus field with the

MaritalStatusesCst3Attribute:

```
public class ContactExt : PXCacheExtension<Contact>
{
     [PXRemoveBaseAttribute(typeof(MaritalStatusesAttribute))]
     [PXMergeAttributes(Method = MergeMethod.Append)]
     [MaritalStatusesCst3]
     public string MaritalStatus { get; set; }
}
```

Now there are only 6 items: 2 original, 3 custom and 1 replaced - in the **Marital Status** drop-down list:

#### O Revision Two HQ - Contacts \*

Cont	act ID:	Air, Ja	ine		Q
Туре		Conta	ct		
		🗹 Activ	ve		
Details	Additional Info	Attributes	Activities	Relations	Opportunities
Gend	ION				÷
COMN Geno Marit	ION				*
Geno Marit Spou	ION	Marrie	d		×
Geno Marit Spou LEAD	ION	Marrie Divorc	d ed (not living	g common I	* * aw)
COMM Genc Marit Spou LEAD Sour	ION al Status: ise/Partner Nami HISTORY ce:	Marrie Divorc Widow	d ed (not living ved	g common l	▼ ▼ aw)
COMN Geno Marit Spou LEAD Sour Cam	ION al Status: ise/Partner Nami HISTORY ce: paign ID:	Marrie Divorc Widow Living Separa	d ed (not livin ved common lav	g common l w ing commor	aw)

Read Modifying Items in a Dropdown List online:

https://riptutorial.com/acumatica/topic/9392/modifying-items-in-a-dropdown-list

# Chapter 20: Populating report with data through code

#### Examples

This article covers example showing how to create report using memory records:

This example shows how to populate report with data returned by a data view delegate. During the exercise, we will develop an inquiry screen showing list of Sales Orders between two dates. Data view delegate will be used to populate Sales Order information.

#### **Prerequisites:**

1. We start with declaration of the SOOrderFilter DAC:

```
[Serializable]
public class SOOrderFilter : IBqlTable
{
   public abstract class dateFrom : IBqlField
   {
   }
   [PXDate()]
   [PXUIField(DisplayName = "Date From")]
   public DateTime? DateFrom { get; set; }
   public abstract class dateTo : IBqlField
    {
   }
   [PXDate()]
   [PXUIField(DisplayName = "Date To")]
   public DateTime? DateTo { get; set; }
}
```

2. Continue with declaration of the SOOrderData DAC:

```
[Serializable]
public class SOOrderData : IBqlTable
{
    #region OrderType
    public abstract class orderType : PX.Data.IBqlField
    {
        }
        [PXString(2, IsKey = true, IsFixed = true)]
        [PXUIField(DisplayName = "Type")]
        public virtual string OrderType { get; set; }
        #endregion
        #region OrderNbr
        public abstract class orderNbr : PX.Data.IBqlField
        {
        }
    }
```

```
[PXString(15, IsKey = true, IsUnicode = true, InputMask = ">CCCCCCCCCCCC")]
[PXUIField(DisplayName = "Order Nbr.")]
public virtual string OrderNbr { get; set; }
#endregion
#region OrderDate
public abstract class orderDate : PX.Data.IBqlField
{
}
[PXDate]
[PXUIField(DisplayName = "Date")]
public virtual DateTime? OrderDate { get; set; }
#endregion
#region Status
public abstract class status : PX.Data.IBqlField
{
}
[PXString(1, IsFixed = true)]
[PXUIField(DisplayName = "Status")]
[SOOrderStatus.List()]
public virtual string Status { get; set; }
#endregion
#region OrderDesc
public abstract class orderDesc : PX.Data.IBqlField
}
[PXString(60, IsUnicode = true)]
[PXUIField(DisplayName = "Description", Visibility = PXUIVisibility.SelectorVisible)]
public virtual string OrderDesc { get; set; }
#endregion
#region OrderTotal
public abstract class orderTotal : PX.Data.IBqlField
{
}
[PXDecimal(4)]
[PXDefault(TypeCode.Decimal, "0.0")]
public virtual decimal? OrderTotal { get; set; }
#endregion
#region DueDate
public abstract class dueDate : PX.Data.IBqlField
}
[PXDate]
[PXUIField(DisplayName = "Due Date")]
public virtual DateTime? DueDate { get; set; }
#endregion
```

3. In PX.Documentation namespace create your SOOrderInq BLC using the code snippet below to declare Results data view delegate, which will later use to populate report with data:

```
public class SOOrderInq : PXGraph<SOOrderInq>
{
    public PXCancel<SOOrderFilter> Cancel;
    public PXFilter<SOOrderFilter> Filter;
    [PXFilterable]
    public PXSelectOrderBy<SOOrderData,
        OrderBy<Desc<SOOrderData.orderNbr>>> Result;
    protected virtual IEnumerable result()
```

}

```
{
        BqlCommand cmd = PXSelect<SOOrder,</pre>
            Where<SOOrder.orderDate,
                Between<Current<SOOrderFilter.dateFrom>,
                    Current<SOOrderFilter.dateTo>>>>.GetCommand();
        PXView inView = new PXView(this, true, cmd);
        int startRow = PXView.StartRow;
        int totalRows = 0;
        foreach (SOOrder order in inView.Select(PXView.Currents, PXView.Parameters,
            PXView.Searches, PXView.SortColumns, PXView.Descendings, PXView.Filters,
            ref startRow, PXView.MaximumRows, ref totalRows))
        {
            yield return new SOOrderData
            {
                OrderType = order.OrderType,
                OrderNbr = order.OrderNbr,
                OrderDate = order.OrderDate,
                Status = order.Status,
                OrderDesc = order.OrderDesc,
                OrderTotal = order.OrderTotal,
                DueDate = order.DueDate,
            };
        }
        PXView.StartRow = 0;
    }
    public SOOrderInq()
    {
        Result.Cache.AllowInsert = false;
        Result.Cache.AllowUpdate = false;
        Result.Cache.AllowDelete = false;
    }
    public PXAction<SOOrderFilter> Report;
    [PXButton]
    [PXUIField(DisplayName = "View As Report", MapEnableRights = PXCacheRights.Select,
MapViewRights = PXCacheRights.Select)]
    protected virtual void report()
    {
        PXReportResultset reportData = new PXReportResultset(typeof(SOOrderData));
        foreach (SOOrderData row in Result.Select())
        {
            reportData.Add(row);
        }
        throw new PXReportRequiredException(reportData, "SO610501",
PXBaseRedirectException.WindowMode.NewWindow, "Report");
   }
}
```

- 4. Create SO401090.aspx page by selecting FormDetail template, and set the following properties for PXDataSource:
- PrimaryView: Filter
- TypeName: PX.Documentation.SOOrderInq

After that add input control on the Filter header form:

And create the following columns for the Detail grid:

```
<pr:PXGrid ID="grid" runat="server" DataSourceID="ds" Style="z-index: 100"</pre>
    Width="100%" Height="150px" SkinID="Inquire" AllowPaging="True"
AdjustPageSize="Auto">
     <Levels>
         <px:PXGridLevel DataMember="Result">
             <Columns>
                 <px:PXGridColumn DataField="OrderType" />
                 <px:PXGridColumn DataField="OrderNbr" Width="90px" />
                 <px:PXGridColumn DataField="OrderDate" Width="90px" />
                 <px:PXGridColumn DataField="Status" />
                 <px:PXGridColumn DataField="OrderDesc" Width="200px" />
                 <px:PXGridColumn DataField="DueDate" Width="90px" />
             </Columns>
         </px:PXGridLevel>
     </Levels>
     <AutoSize Container="Window" Enabled="True" MinHeight="150" />
 </px:PXGrid>
```

5. Add created screen to the Site Map

To populate report with data returned by a data view delegate:

1. Paste SO610501.rpx report file in ReportsCustomized folder of your Acumatica website, then add Sales Orders report in the Site Map **Hidden** folder

```
C Revision Two HQ - Site Map 🖈
```

- r

C		c + 🖊	× ‰ ⊅ ঢ় ♠ + ⊮
	6	Screen ID	Title Icon
⊕ Organization		AR.30.60.00	Dunning Letter
⊕⊖ Finance		EP.61.20.00	Expense Claim Details
Distribution		EP.61.20.00	Expense Claim Details
		GI.00.00.10	BI-Creation Date
u ⊂ Svstem		WZ.20.15.20	Welcome Page
		WZ.20.15.01	Scenario History
		CA.20.45.50	Bank Transaction Rules
+ Criticen		SM.20.20.25	Wiki Articles
		GL.30.70.00	Base Form for Voucher Batches
		CA.20.80.00	Update Expiration Dates
		GL.40.50.00	Reclassification History
	>	SO.61.05.01	Sales Orders
	<		

 Declare View as Report action in the SOOrderInq BLC to generate and show Sales Orders report. The PXReportRequiredException accepts PXReportResultset prepared inside the action to populate report with data returned by Result data view delegate:

```
public class SOOrderIng : PXGraph<SOOrderIng>
{
    . . .
   public PXAction<SOOrderFilter> Report;
   [PXButton]
    [PXUIField(DisplayName = "View as Report", MapEnableRights = PXCacheRights.Select,
MapViewRights = PXCacheRights.Select)]
   protected virtual void report()
    {
       PXReportResultset reportData = new PXReportResultset(typeof(SOOrderData));
       foreach (SOOrderData row in Result.Select())
        {
            reportData.Add(row);
        }
        throw new PXReportRequiredException(reportData, "SO610501",
PXBaseRedirectException.WindowMode.NewWindow, "Report");
   }
}
```

ę	3 Revi	sion Two H	Q - 5	ales	Orde	ers 🔰	k					
r	VIEV	V AS REPOR	т									
	Date Fron	n: 1/1/2010	🗸 Da	te To:	1/1/	2020	Ŧ					
		_										
C	3  ↔	x	Sale	s Orders			×	+				
	Туре	Order Nbr.	Jaie	solueis	•		^	<u> </u>				
>	SO	SO003631	(*)	D   loca	alhost/	04961	3-2/(W(1	0005))/	frame	s/reportlaun	cher.aspx?i	d=so C
	SO	SO003630	D		<b>T</b>		<b>.</b>	<u> </u>				
	SO	SO003629	Rev	ISION	IWO	HQ	Sales	Order	S			
	SO	SO003628			-							
	50 50	SO003627	G	PDF	*8=	<	<	>	Я	PRINT	SEND	EXPORT
	50 50	80003626					_					
_	30 80	SO003626					Sales (	Order S	umma	агу		
_	30	50003625					Company	/: R	evision	Two HQ		
	50	SO003624					USEI.	a	umm			
	SO	SO003623					Туре	Ref. Nt	or.	Order Date	Order Des	scription
	SO	SO003622					SO SO	SO003	<u>531</u> 530	10/19/2016	Food purch	goods purchase lase for event
	SO	SO003621					SO	S0003	<u>529</u>	10/21/2016	Widget pur	chase
	SO	SO003620					SO SO	SO003	<u>528</u> 527	10/29/2016	Tech sale v	<ul> <li>new account with installation</li> </ul>
	SO	SO003619					SO	<u>SO003</u>	<u>626</u>	11/7/2016	Widget adv	ance order
	SO	SO003618					SO SO	SO003	525 524	10/30/2016 10/30/2016	Custom Or	t order der with 50% pr
	SO	SO003617	1				SO	SO003	523	10/21/2016	Industrial E	quipment - Spec
	SO	SO003616					SO SO	SO003	<u>522</u> 521	10/12/2016	Food Order	- Drop Shipped r for future deliv
	s0	SO003615					SO	<u>SO003</u>	620	10/30/2016	Widget Ord	er - Pay with C
_	80	50003013					SO SO	SO003	519 518	10/25/2016	Consumer	ier Good Order
	30	30003614					SO	S0003	617	10/9/2016	Consumer	Good Order
	50	SO003613					SO SO	SO003	515 515	10/14/2016	Consumer	Good Toy Order Good Toy Order
	SO	SO003612					SO	<u>SO003</u>	614	10/31/2016	Industrial E	quipment Order
	SO	SO003611					SO SO	SO003	513 512	10/9/2016 10/5/2016	Industrial E	quipment Order quipment Order
	SO	SO003610					SO	<u>SO003</u>	<u>611</u>	10/13/2016	Laptop con	nputer order
	SO	SO003609					SO SO	SO003	510 509	10/6/2016	Electronics	Headset Order
	SO	SO003608					SO	<u>SO003</u>	508	10/26/2016	Beverage (	Order
	SO	SO003607	1				SO SO	SO003	507 506	10/25/2016	Food Order Food Order	r r
	SO	SO003606					SO	<u>SO003</u>	605	10/10/2016	Food Order	Minnahin
	SO	SO003605					50 S0	SO003 SO003	504 503	10/6/2016	Food Order	- MICROCNIP
	50	SO003604					SO	S0003	<u>602</u>	9/6/2016	Sell spare	widget inventor
	30	00003004					SO SO	<u>SO003</u>	501 500	9/16/2016 9/19/2016	Food purch	goods purchase lase for event
	50	50003603	-				SO	S0003	599	9/20/2016	Widget pur	chase
			-				SO SO	<u>S0003</u> S0003	598 597	9/28/2016 9/23/2016	Tech sale v	<ul> <li>new account with installation</li> </ul>
							SO	<u>SO003</u>	596	10/7/2016	Widget adv	ance order
							SO	<u>SO003</u>	595	9/29/2016	Industrial lif	t order

Read Populating report with data through code online: https://riptutorial.com/acumatica/topic/7700/populating-report-with-data-through-code

# Chapter 21: Publishing skipped already applied customization content

#### Introduction

When publishing a customization project, you might see some item being skipped for the reason of being already applied. Ex:

EntityEndpoint EntityEndpoint#6.00.001§DefaultPlus(skipped, already applied)

This can happen for any items contained saved in the database. Ex: Generic inquiries, reports, site map nodes, DB scripts, system locales, import/export scenarios, shared filters, access rights, wikis, web service endpoints and analytical reports.

#### **Examples**

Publish with cleanup from the customization screen

- 1. You must obviously select the project that you want to publish.
- 2. You must click on the small arrow right next to the "Publish" button.
- 3. You must click on the "Publish to Multiple Companies" option.

<b>Q</b> Acuma	atica 🦷	ORGANIZATION	FINANCE	DIS	TRIBUT	ION CONF
Management	Integration	Automation	Customization			
Customizati	on			0	Revi	sion Two HC
Type your que	ry here		Search	с		r +
- MANAGE						Published
Customizatio	n Projects			>		
Generic Inquir	rv					
Lists as Entry	Points					

- 4. On the smart panel that will appear you must select the companies that you want to publish the project(s). Only one company is also a possibility.
- 5. Check the check box indicating "Publish with Cleanup", this will make sure to reapply all item present in the customization project replacing the already present one with their newer version.

Acumation	<b>tica</b> o	RGANIZATION	FINANCE	DIST	TRIBUT	TION	CONF
Management	Integration	Automation	Customization				
Customization	n		•	Ø	Revi	sion <sup>-</sup>	Two HC
Type your query	here		Search	с		5	+
- MANAGE				8			Publishee
Customization	Projects			>			
Generic Inquiry				-			
Lists as Entry P	oints						
Pivot Tables	onno					Publis	h to Mult
Dashboards						E Sele	cted
Site Map						• 🗸	
Portal Map							
Filters							
External Applica	ations						
Application Res	ources						
- PROCESS	our ooo						
Refresh Applica	tion Access	Tokens					
	10117100033	Tokons					
Source Code							)atabase (
Source Code							ublich wit
							ublish wit

Publish with clean up from inside a customization project

- 1. Open the customization project that you want to publish with this method.
- 2. Open the publish menu at the top and select the "Publish with Cleanup" option.

<b>Q</b> A	cumat	tica		
File	Publish	Extension Library	Source Control	
Custo Scree Data Code Files Gene Repor Site M DB So Syste Impor Share Acces Wikis Web	Publish Validate Multiple Publish ric Inquiries ( rts (1) Map (1) cripts (1) m Locales (1) t/Export Sce ed Filters (1) ss Rights (1) (1) <b>Service End</b> tical Reports	a Current Project (Ctrl+ e Current Project e Projects a with Cleanup (1) (1) (1) (1) (1) (1) (1)	+Space)	<ul> <li>Web Service Endpoints</li> <li>C □</li></ul>

\*Please take note that all customization project that are selected on the customization screen will be republish even if you are inside only a single project.

Read Publishing skipped already applied customization content online: https://riptutorial.com/acumatica/topic/10155/publishing-skipped-already-applied-customizationcontent

# Chapter 22: Replacing Images on the Login Page

#### Introduction

In this topic you will learn how to replacing standard Acumatica images on the login page. The demonstrated approach will make sure to keep your custom images on login page after the upgrade to a newer version and restore original images, provided by Acumatica, if at some point your customization appears unpublished.

#### Examples

#### Using customization to replace images on the login page

To create a customization package replacing images on the login page, follow the steps below on your **local Acumatica instance**:

1. Create a new folder in the Acumatica instance folder. For this example I added a folder called *IconsCustomized* in my local **LoginImages** instance:

		CONTRACTOR OF	
	Local Deal (C.) + Californiqueben	· · Lopomage: ·	
and the second se	Outs modified	Type	248
Api	4/12011245764	Princhester	
App; Code	4/1/2011 2-45-PM	Fire hybrides	
App_Deta	4/12017-245-996	File holder	
App, RuntimaCude	ACT/2011 246 PM	Frie fulde-	
App, Thereas	A/12017248-PM	File follow-	
Bin .	4/1007346/Mil	File Indan	
Content	4/3/2017 2-46-PM	The hotse	
Carerala	4/1/2017/246/PM	Kine bytime	
Cithesigner	47520112-46744	The Iplan	
Catholined	ACTION CONTRACTOR	Fire hybrid	
Customiation	4/120112-46-964	Prig-builder	
Eule .	4/1/2011/246798	the future	
Estatualization	4/12017-246494	File hoties	
Frances	ACCOUNT 2 NO PER	File fulle-	
Generalization	A/10217246-PM	The follow	
loss	4/1/20/1/2.44 PM	The Indian	
Icons/Cattorneed	4/120124EPM	Title folder	
MatterPages	A/3/2017.2x46/948	Kim further	
Pages	ACTORYCOMPANY	The lipides	
Firet	4/12017.046494	Fire hutches	
Reports Customized	ACCEPT 2-46-PM	Fig-hadder	
Report/Delaut	4/32011246798	the balance	
Scripto	4/12012-044946	File fulling	
Serch	ACCENTED AN PEAK	File holder	
Sounds	A/0201246/MI	The follow-	
Views	4/12017.046764	File follow	
was	4/1/2017 2:46/967	File holder	
field.	APA/2017 IDek.AM	1.07 File	100.00
Chubalanas	LINEST BUSINE	ANPART Internation	18.68
Norman lice	STATUTEALAN	UCLEM .	3.0
Main.augo	DIA/DITERLIN	APTIN	3.48
Main argn co	L/14/007/040-484	That Of Install,	20.0
OuterPage.html	ATA/DITTERLAM	Influt. Fee	3.10
3 packages coving	1/14/2017 848-464	108, Crobpettin-	1.0
collects. Add	MARCHITERED AND	Test Discovery	1.00
Web Config	4/52017346966	Mill Cooligandia.	-10-12
Toronto acception	STRACTORY BURN AND	COMPLEX FOR	46.13
and the second s			

2. Add your custom images in this folder. For the sake of this example, I used images from Acumatica 4.2 login page:

- 1 (c) - 1 (c)	a l	Share		Public faith	Instructuationsigned
	- +	+ This PC	+ Leos	Disk (C) > 0	Customizations + Engineerages + IconsCustomized
-	5				
1	1	A		6	
inger.	Jug Lipo	light by	ilipa	logis,bgl.	arg hepszeleter hepszeletere hepszeletere

Keep in mind, *to replace all images on the login page*, you have to add at least as many custom images in your **IconsCustomized** folder as the number of the <code>login\_bg\*.\*</code> files originally present in the **Icons** folder of your Acumatica website. It's perfectly fine to use same image or images multiple times (by naming the files differently), if the number of your custom images is less then what was originally provided by Acumatica.

- 3. Now login to your Acumatica application, create new customization project called *LoginPageImages* and open it in Customization Manager.
- 4. In Customization Manager, navigate to the **Files** section and click the **Add New Record** button to open the **Add Files** dialog:

N. Popula Coleman	n Libery Seat	a Central		
oginPagelmages 4	Custom File	6		
		C 2		
Screets	C H .	A X + DETECT MODIFIED FILES		
Data Access	III G Otgest Rame			
Code 1	-			-
des	Addition			
ienetic Inquines	c			
beporte	E belected	Puth	Modified	Size.
ite Map	2.1	App_DataR/pdateStatus.xml	4/7/2017	108
o ocripta	D D	App. Data/WebstelD1982058358.txt	4/7/2017	
ystem Locares	0	Birr/AWSSDK.pdb	3/14/2017	36867584
sport/Export Scienarios		BinittelAgiltyPack.pdb	3/14/2017	298496
hared Filters	i a	EinridentityModel pdb	3/14/2017	153088
ccess Rights		BinlidentlyGever3 pdb	3/14/2017	5248512
Vikia		Bin/Microsoft Practices ServiceLocation pdb	3/14/2017	24064
Veb Service Endpoints		Bin/Microsoft WindowsAzure Storage pdb	3/14/2017	2229760
nalytical Reports	0	Bin/PX Api ContractBased Common pdb	3/14/2017	105984
	0	Bin/PX.Api ContractBased pdb	3/14/2017	1080832
		Bin/PX Api OData pdb	3/14/2017	222720
		ALCONTRACTOR AND A	This support	and star

5. In the Add Files dialog, select all files from your **IconsCustomized** folder and click **Save**:

Selected	Path	Modified	Size	
	Bin/PX.Web.UI.Design.pdb	3/14/2017	478720	1
0	Bin/PX Web ULpdb	3/14/2017	4781568	
D	Bin/PX.WebConfig.pdb	3/14/2017	116224	
0	Bin/StackExchange Redis StrongName.pdb	3/14/2017	728576	
C	Bin/System.IdentityModel.Tokens.Jwt.pdb	3/14/2017	151040	
• 🗹	IconsCustomizedVogin_bg1.jpg	1/24/2017	128468	
•	IconsCustomizedVlogin_bg2.jpg	1/24/2017	226892	
• 🗹	IconsCustomizedVogin_bg3.jpg	1/24/2017	311225	
• 🗹	IconsCustomized/login_bg4.jpg	1/24/2017	950692	
• 2	IconsCustomizedVogin_bg5.jpg	1/24/2017	144836	
• 🗹	IconsCustomizedVogin_bg6.jpg	1/24/2017	406944	
• •	IconsCustomizedVogin_bg7.jpg	1/24/2017	101258	8

Now you have the custom login page images in the customization project, but you still need to edit the path so they correctly replace the standard images.

6. In Customization Manager, select Edit Project XML from the File menu:

ile Publish Exter	on Library Source Control					
Manage Customization	ojecta im Files					
Edit Project XML						
Edit Project Items	B • X + DETECT MODIFIED	DETECT MODIFIED FILES				
Export Project Package	bject Name	bject Name				
Replace from Package	onsCustomized/login_bg1.jpg	onsCustomized/login_bg1 jpg				
	bnsCustomized/login_bg2 jpg					
Generic Inquiries	konsCustomized/login_bg3.jpg					
Reports	konsCustomizedVogin_bg4.jpg	konsCustomizedVogin_bg4.jpg				
Site Map	konsCustomized/login_bg5_ipg	konsCustomized/login_bg5.jpg				
OB Scripts	a konsCustomizedVogin bali jog					
System Locales						

7. For all the File tags, generated for your custom images, charge the AppRelativePath attribute to AppRelativePath="lcons..." and set the SystemFile attribute to True for those images, that currently present in the lcons folder, then click the Save to Database button when done:

Acumatica	
File Publish Extension	n Library Source Control
LoginPageImages <	LoginPageImages
Screens Data Access	Save to database Download Package Upload Package Browse No fil
Code	<customization description="" level="0"></customization>
Files (7)	<File AppRelativePath="Icons\login_bg1.jpg" FileID="4f30b2</td>
Generic Inquiries	<File AppRelativePath="Icons\login_bg2.jpg" FileID="fad8c5</td>
Reports	<pre>&lt;File AppRelativePath="Icons\login_bg3.jpg" FileID="f4337;&lt;br&gt;&lt;File AppRelativePath="Icons\login_bg4.ipg" FileID="129774&lt;/pre&gt;</pre>
Site Man	<pre>&lt;File AppRelativePath="Icons\login_bg5.jpg" FileID="60e29a&lt;/pre&gt;</pre>
	<File AppRelativePath="Icons\login_bg6.jpg" FileID="1133b0</td>
DB Scripts	<File AppRelativePath="Icons\login_bg7.jpg" FileID="8e2420</td>
System Locales	

While publishing customization, Acumatica will automatically backup files currently present in the website folder, which are replaced by files from the customization with **SystemFile** attribute set *True*.

8. If you now proceed with publishing the customization, it's very likely for Some files have been modified in the file system. error message to show up. To prevent this quite frightening message from appearing, open you project in Customization Manager, navigate to the Files section and click Detect Modified Files to open the Modified Files Detected dialog, then click the Discard All Changes button:

ginPagelmages	Custom Files	+ 🗖	TECT NO	OFIED FLES
es Access 1 1 1 1 1 1 1 1 1 1 1 1 1	II & Operform • II LONDAR MEAN II LONDAR MEAN	Martine 12	a Descar	Theorytes
same mysion Bandis San Map Ol Duran Dynami Caran Synam Caran Samet Falma Access English Web Banca English Web Banca English Maharda Reports	<ul> <li>tonsiver_belas</li> <li>tonsiver_belas</li> <li>tonsiver_belas</li> </ul>	ON PROJECT DECARD ALL CHANCES 3		
	<ul> <li>Constructure of an</li> <li>Constructure of an</li> <li>Constructure of an</li> </ul>	* 2 2 2 2 2	Cuentian 2 2 2 3 4 3 4 5 5 5 5 5 5 5 5 5 5 5 5 5	Film C Contractations's apprintinger (knowlinger, typ), pp C Contractations's apprintinger (knowlinger, typ), pp C Contractations's apprintinger (knowlinger, typ), pp C Contractations's (apprintinger (knowlinger, typ), pp

9. Now you can go ahead and publish the customization to enjoy your custom images on the login page:



Read Replacing Images on the Login Page online: https://riptutorial.com/acumatica/topic/9657/replacing-images-on-the-login-page

# Chapter 23: Significant API Changes Between Versions

#### **Examples**

PXSelectGroupBy and Bit Values in Acumatica 5.1 and 5.2+

The method of SQL generation from BQL PXSelectGroupBy<> data views has been changed in Acumatica Framework 5.2.

The sections below illustrate the differences using the example of PXSelectGroupBy<FinYear, Aggregate<GroupBy<FinYear.finPeriods>>>.Select(graph):

### Acumatica Framework 5.2 and Later

```
SELECT Max([finyear].[year]),
      Max([finyear].[startdate]),
      Max([finyear].[enddate]),
      [finyear].[finperiods],
       -- Attention!
       CONVERT (BIT, Max([finyear].[customperiods] + 0)),
      Max([finyear].[begfinyearhist]),
      Max([finyear].[periodsstartdatehist]),
      Max([finyear].[noteid]),
      ( NULL ),
      ( NULL ),
      ( NULL ),
      Max([finyear].[tstamp]),
      Max([finyear].[createdbyid]),
      Max([finyear].[createdbyscreenid]),
      Max([finyear].[createddatetime]),
      Max([finyear].[lastmodifiedbyid]),
      Max([finyear].[lastmodifiedbyscreenid]),
      Max([finyear].[lastmodifieddatetime])
FROM finyear FinYear
WHERE (finyear.companyid = 2)
GROUP BY [finyear].[finperiods]
ORDER BY Max([finyear].[year])
```

### **Acumatica Framework 5.1 and Earlier**

```
SELECT Max([finyear].[year]),
    Max([finyear].[startdate]),
    Max([finyear].[enddate]),
    [finyear].[finperiods],
    -- Attention!
    ( NULL ),
```

```
Max([finyear].[begfinyearhist]),
      Max([finyear].[periodsstartdatehist]),
       ( NULL ),
       ( NULL ),
       ( NULL ),
      Max([finyear].[tstamp]),
      ( NULL ),
      Max([finyear].[createdbyscreenid]),
      Max([finyear].[createddatetime]),
      ( NULL ),
      Max([finyear].[lastmodifiedbyscreenid]),
      Max([finyear].[lastmodifieddatetime])
FROM
      finyear FinYear
WHERE (finyear.companyid = 2)
GROUP BY [finyear].[finperiods]
ORDER BY Max([finyear].[year])
```

# Explanation

By default, the Max() aggregate is applied to all fields not explicitly mentioned in a BQL statement.

However, in Acumatica 5.1 and earlier, it excludes the CreatedByID, LastModifiedByID, and bool fields. When translated into SQL, these fields will always be null unless you explicitly grouped by.

Starting from version 5.2, Max() will be applied by default for them, too.

Read Significant API Changes Between Versions online: https://riptutorial.com/acumatica/topic/9697/significant-api-changes-between-versions

## **Chapter 24: User Interface Techniques**

#### **Examples**

Creating a Dropdown Menu for a Screen

Suppose that you need to define a dropdown menu for a particular Acumatica screen, such as the Reports menu on the following screenshot.



This can be achieved in three different ways:

- By adding a toolbar with a menu item to the screen's ASPX
- By declaring a special "folder" action to the graph and adding menu items in the code
- By using the Automation subsystem of the Acumatica Framework (not covered by this example)

# Option 1: Creating a Dropdown Menu in ASPX

First of all, make sure that the ASPX page's PXDataSource element contains all the necessary commands corresponding to the graph actions that you would like to perform when clicking on a menu item.

```
CallbackCommands>
```

Next, add a custom toolbar element right after the PXDataSource element. Within it, define a PXToolbarButton with the desired dropdown menu items linking to the respective datasource commands, as shown in the following code.

```
<px:PXToolBar ID="toolbar1" runat="server" SkinID="Navigation" BackColor="Transparent" CommandSourceID="ds">
```
This option might look tempting due to its simplicity; however, there is one **important drawback**. If you implement such a dropdown on a screen with a processing indicator (such as a document release screen or a mass-processing screen), the indicator will appear to the left of your dropdown menu, as shown below.

ç	C New York - Translation Worksheets ★												
ľ	r	+ 🔋	к	<	>	≻	RELEASE	VI	EW TRANSLATION BATC	н	Ø	00	
	Translation N	Number:	00000	7	Q	Trar	slation ID:		EUR	[	Debit T	otal:	
	Status: Translation Date: Currency Effective Date: Fin. Period:			Released 1/29/2016 10/31/2012 09-2012 Translation to Euro			nch:		MAIN - New York	(	Credit Total:		
							tination Ledger I	D:	TRANSEURO		Control Tota		
							tination Currency	y:	EUR				
							nslation Batch N	um	00004550				
Description:			Trans										

If this is not desirable, consider defining a dropdown menu in the code as described in the Option 2 section below.

# **Option 2: Creating a Menu in the Graph**

First, in the page's graph, declare a "folder" action that will correspond to the dropdown menu button.

```
public PXAction<TranslationHistory> reportsFolder;
[PXUIField(DisplayName = "Reports", MapEnableRights = PXCacheRights.Select)]
[PXButton(SpecialType = PXSpecialButtonType.Report)]
protected virtual IEnumerable Reportsfolder(PXAdapter adapter)
{
    return adapter.Get();
}
```

Next, in the graph's constructor, indicate that the action is indeed a dropdown menu and add all actions that need to be displayed as menu items, as shown below.

```
public TranslationHistoryMaint()
{
    this.reportsFolder.MenuAutoOpen = true;
    this.reportsFolder.AddMenuAction(this.translationDetailsReport);
}
```

If you select this approach, the processing indicator will always appear to the right of your menu, which is arguably better UX.

Read User Interface Techniques online: https://riptutorial.com/acumatica/topic/10150/userinterface-techniques

## Chapter 25: Using Customization Plug-In to Make Changes in Multiple Companies

#### Introduction

With classes derived from **CustomizationPlug** you can utilize capabilities of the Acumatica Customization Platform and execute custom code after the customization project has been published. In this topic you will learn how customization plug-ins can be used to make changes in multiple companies.

More information on customization plug-ins is available in Acumatica Customization Guide

### Examples

Implementation of a customization plug-in to update multiple companies

To create a customization plugin, you simply create a class derived from **CustomizationPlug** and package it into customization. While the system is publishing customization project, it will execute the **OnPublished** and **UpdateDatabase** methods implemented in your customization plugin *only within the current company scope*.

With that said, customization plug-in will never make changes to any other than current company, unless it uses **PXLoginScope** to log into all companies, one after the other, available to the current user publishing customization.

Below is an example of customization plugin creating *MyVerticalSolution* user role in all companies available to the current user:

```
public class MyVerticalSolutionInit : CustomizationPlugin
{
   public override void UpdateDatabase()
    {
        var companies = PXAccess.GetCompanies();
        foreach (var company in companies)
        {
            using (var loginScope = new PXLoginScope(string.Format("{0}@{1}",
               PXAccess.GetUserLogin(), company)))
            {
                string roleName = "MyVerticalSolution";
                RoleAccess graph = PXGraph.CreateInstance<RoleAccess>();
                Roles existingRole = graph.Roles.Search<Roles.rolename>(roleName);
                if (existingRole != null)
                {
                    WriteLog(string.Format("{0} already exists in company '{1}' - skipped",
roleName, company));
                    continue;
                }
```

```
var wmsRole = new Roles();
wmsRole.Rolename = roleName;
wmsRole.Descr = "User Role for MyVerticalSolution";
graph.Roles.Insert(wmsRole);
graph.Save.Press();
WriteLog(string.Format("{0} was succesfully created in company '{1}'",
roleName, company));
}
}
}
```

To obtain a list of companies available to the current user, you simply invoke static PXAccess.GetCompanies() method. Then **PXLoginScope** is used to log into each of the available companies to create **MyVerticalSolution** user role. Notice instance of the **RoleAccess** BLC reinitialized for each company - this is an absolutely mandatory step to making changes to multiple companies at a time.

Let's assume there are 2 companies on your Acumatica instance: CompanyA and CompanyB. The **admin** user, that you are going to use to publish customization, has access to both companies and *MyVerticalSolution* role, created by customization plug-in, already exist in CompanyA:

<b>Acumatica</b> ORGAI	NIZATION	CONFIGU	RATION	×	3/	/23/2017 3:38 PM	A	DMIN	
Common Settings User Security	Row-Level \$	Security	Document	t Manage	ement	Email	Ċ	) Sign (	
User Security	C Revis	ion Two	HQ - L	Jser R	oles	*		Organ User	
Type your query nere Search	<b>``</b>	+ •	K	<	>	X	~	Comp	
→ MANAGE	★ Role Name	MyVert	MyVerticalSolution - User Role for My P						
User Roles	Role Desci	ription:	User R	User Role for MyVerticalSolution					
Access Rights by Screen	Membership								
Access Rights by User	c +	× ⊢							
Access History	* Username		0	)isplay Na	ame				

After you published customization (while logged into CompanyA or CompanyB) with earlier developed customization plug-in to create *MyVerticalSolution* role in all companies available to the current user, notice *MyVerticalSolution* role skipped for CompanyA and successfully created for CompanyB.



Next time you publish this customization, *MyVerticalSolution* role will be skipped for both companies in your Acumatica application:

C	;	-	+	×	PUBLISH -	UNPUBLISH	ALL	IMPORT	•	EXPORT	VIEW	PUBLIS
<b>a</b>		Publishe	d	* Project Na My Vertica	ame alSolution	Level	Screen Na	ames	Desci	ription		Created admin
С	ompil	lation										
L	Pu	blish (	Cust	tomiza	tion							
	Comp: Valid Copy: Patcl Patcl Done Valid Build Build Build Build Build Build Build Build Build Build Build Build Build Build Build Build Build	iled proj dation ha ing the w hing the hing the dating bi dating bi dating bi dating dire ding dire ding dire ding dire ding dire ding dire ding tire ding dire ting webs	ects: s bee ebsit file nary e web efaul ctory ctory ctory ctory ctory ctory ctory ctory ctory ctory ctory ctory ctory	MyVertic e C:\Custon C:\Custon C:\Custon files site C:\C tAppPool '\WebSit '\WebSit '\WebSit '\WebSit '\WebSit '\WebSit seconds: en finishe	calSolution d. tomizations\Cust mizations\Cust mizations\Cust Customizations teValidationDo teValidationDo teValidationDo teValidationDo teValidationDo teValidationDo 2.938234 ed successfull	stomization\05 omization\0552 omization\0552 \Customization main\App_Runti main\Controls\ main\Customiza main\ExternalR main\MasterPag main\Pivot\'. main\'.	5265Valida 55Valida 55Valida \055265V neCode\' '. tion\'. esource\ es\'.	dation\05 tion\05 alidatio	05526 5265W 5265W	5Website ebsite\App ebsite\App 5265Website	_RuntimeC _RuntimeC	ode \MyV ode \MyV
	Star Plug MyVer MyVer Webs:	ting the -in MyVer rticalSol rticalSol ite has b	websi tical ution ution	te Solution] already already pdated.	Init exists in com exists in com	pany 'CompanyA pany 'CompanyB	' - skip ' - skip	ped ped				

Read Using Customization Plug-In to Make Changes in Multiple Companies online: https://riptutorial.com/acumatica/topic/9522/using-customization-plug-in-to-make-changes-inmultiple-companies

## Credits

S. No	Chapters	Contributors					
1	Getting started with acumatica	Community					
2	Acumatica BQL Reference	wh1t3cat1k					
3	Acumatica Platform Attributes Reference	wh1t3cat1k					
4	Adding Attribute Support to out-of-box Sales Order Entity	DChhapgar					
5	Changing caption dynamically using readonly DAC fields.	cbetabeta, Simon ML					
6	Changing Size of Selector Drop-Down Window	Gabriel, RuslanDev					
7	Conditionally Hiding Tabs	RuslanDev					
8	Creating Date and Time Fields in Acumatica	RuslanDev					
9	Customization Mechanisms	wh1t3cat1k					
10	Displaying an Error Requiring to Enter Entity Data	wh1t3cat1k					
11	Downloading Files Attached to a Detail Entity Using Contract-Based API	RuslanDev					
12	Exporting Records via REST Contract-	RuslanDev					

	Based API	
13	Exporting Records via Screen-Based API	RuslanDev
14	Extending List of Entities Supported by Tasks, Events and Activities	RuslanDev
15	Filtering with multiple value with only one selector	samol518
16	Freight Calculation	RuslanDev
17	Modifications to Base Data Views	RuslanDev
18	Modifications to Contact and Address Info through Code	RuslanDev
19	Modifying Items in a Dropdown List	RuslanDev
20	Populating report with data through code	Gabriel, RuslanDev
21	Publishing skipped already applied customization content	samol518
22	Replacing Images on the Login Page	RuslanDev
23	Significant API Changes Between Versions	wh1t3cat1k
24	User Interface Techniques	wh1t3cat1k
25	Using Customization Plug-In to Make Changes in Multiple Companies	RuslanDev