

# T190 Quick Start in Customization

Nayan Mansinha

Lead – Developer Support

# Training Rules

---

- Download training materials related to the webinar. You can find the links to the training materials in the Reminder email sent by RingCentral platform to you.
- This webinar is NASBA compliant. If you want to get CPE credits, you will need to answer 3 polling questions per 1 CPE credit. Questions will be asked at random times.
- You can repeat the lessons after the instructor. Instructor will show you how to complete lessons from the guide.
- If you have any questions, you can select the Q&A option and leave your question there. Instructor will answer them at the end of the session.
- Use chat to inform the instructor whether you hear him or her.
- We encourage you to fill in the survey when the webinar ends.

# Timing and Agenda

---

**May 26, 2020 -10 AM -11 AM PST**

## **Day 1**

### **Lesson 1: Creating a Customization Project**

- Step 1.1: Creating a Customization Project
- Step 1.2: Loading Items to the Customization Project
- Step 1.3: Binding the Extension Library
- Step 1.4: Publishing the Customization Project

### **Lesson 2: Creating Custom Fields**

- Step 2.1: Creating a Custom Column and Field with the Project Editor
- Step 2.2: Creating a Control for the Custom Field

**May 27, 2020 -10 AM -11 AM PST**

## **Day 2**

### **Lesson 2: Creating Custom Fields (cont.)**

- Step 2.3: Creating a Custom Column with the Project Editor and a Custom Field with Visual Studio
- Step 2.4: Creating a Control for the Custom Field—Self-Guided Exercise
- Step 2.5: Making the Custom Field Conditionally Available (with RowSelected)
- Step 2.6: Testing the Customized Form

### **Lesson 3: Implementing the Update and Validation of Field Values**

- Step 3.1: Updating Fields of a Record on Update of a Field of This Record (with FieldUpdated)
- Step 3.2: Validating an Independent Field Value (with FieldVerifying)

# Timing and Agenda

---

**May 28, 2020 -10 AM -11 AM PST**

**Day 3**

## **Lesson 4: Creating an Acumatica ERP Entity Corresponding to a Custom Entity**

- Step 4.1: Performing Preliminary Steps
- Step 4.2: Defining the Logic of Creating an SO Invoice
- Step 4.3: Defining the Create Invoice Action
- Step 4.4: Defining the Visibility and Availability of the Create Invoice Action
- Step 4.5: Testing the Create Invoice Action

## **Lesson 5: Deriving the Value of a Custom Field from Another Entity**

- Step 5.1: Adding a Custom Field to the Payments and Applications Form—Self-Guided Exercise
- Step 5.2: Deriving the Default Value of the PrepaymentPercent Field
- Step 5.3: Testing the Deriving of the Field Value

**May 18, 2020 -10 AM -11 AM**

**Day 3 (cont.)**

## **Lesson 6: Debugging Customization Code**

- Step 6.1: Debugging the Acumatica ERP Source Code



## Quick Start in Customizations

# Useful Development Environment Optimization

---

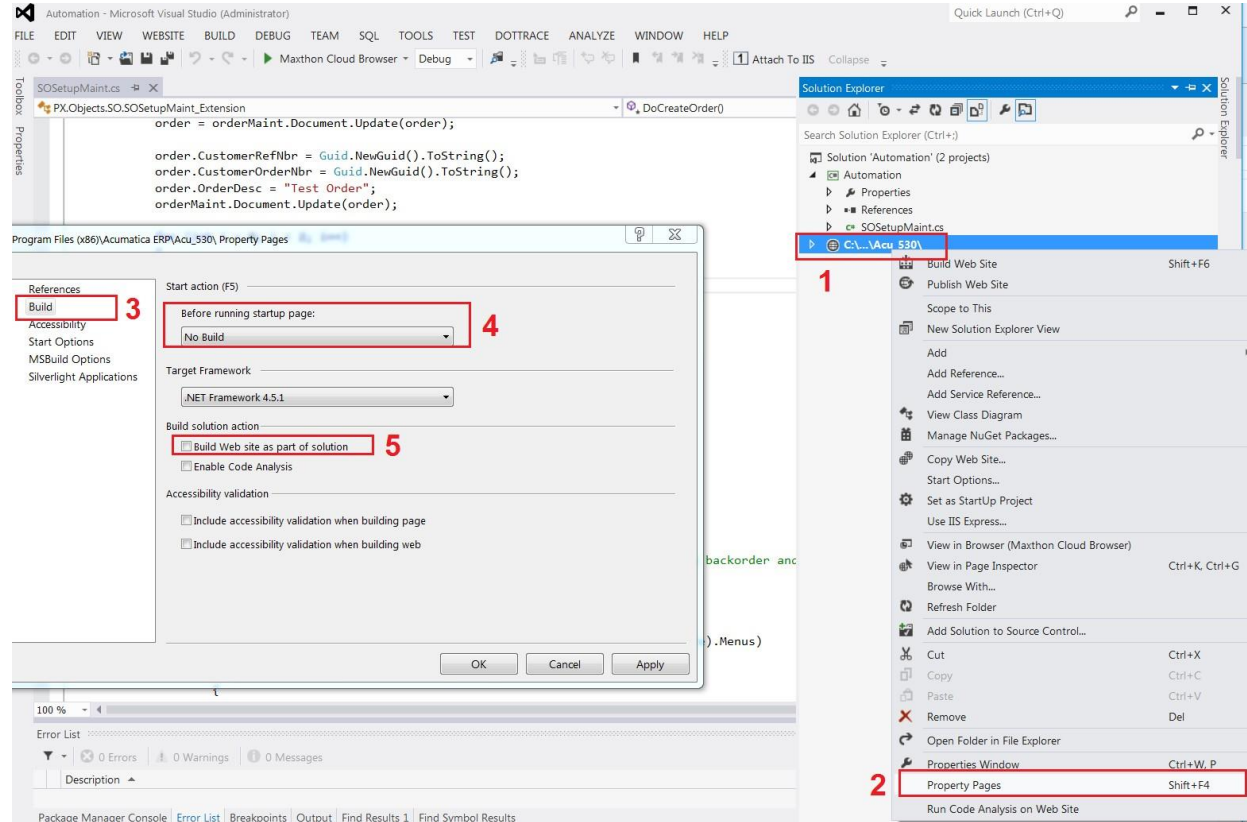
Web.config:

- Enable Debug Web Site - `<compilation debug="True" ... />`
- Optimize Compilation - `<compilation OptimizeCompilations="True" ... />`
- Show Automations - `<add key="AutomationDebug" value="True" />`
- Ignore Scheduler - `<add key="DisableScheduleProcessor" value="True" />`
- Optimize Start-up - `<add key="InstantiateAllCaches" value="False" />`
- Optimize Start-up - `<add key="CompilePages" value="False" />`
- Enable Auto Validation - `<add key="PageValidation" value="True" />`

# Useful Development Environment Optimization

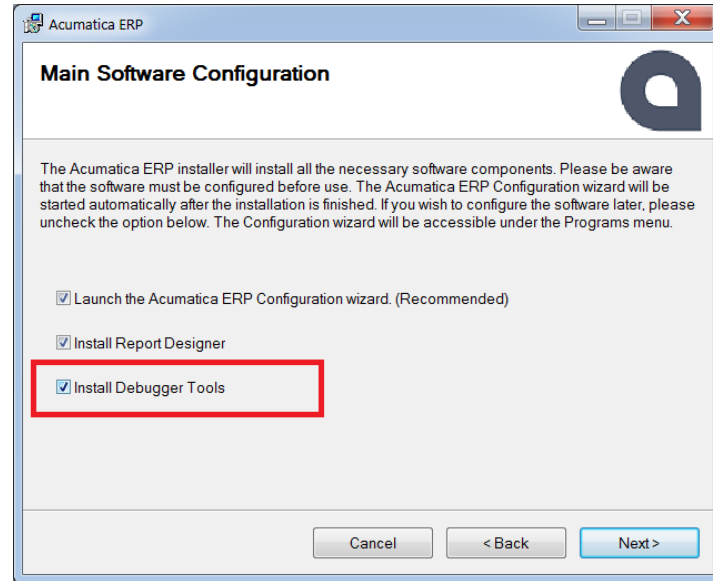
Web Site compilation is slow

...and isn't necessary



# Useful Development Environment Optimization

- Install Acumatica PDB files
  - It allows you to debug Acumatica code

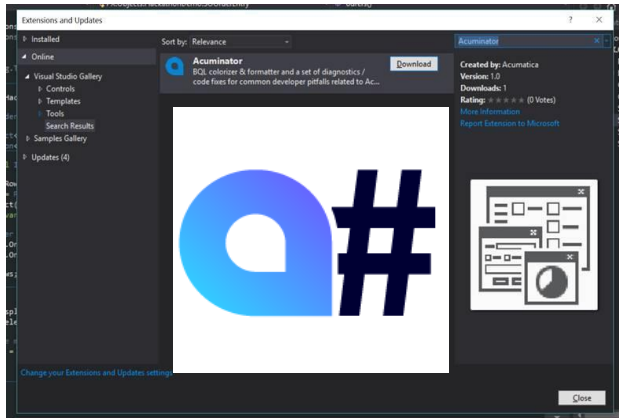




# Useful Development Environment Optimization

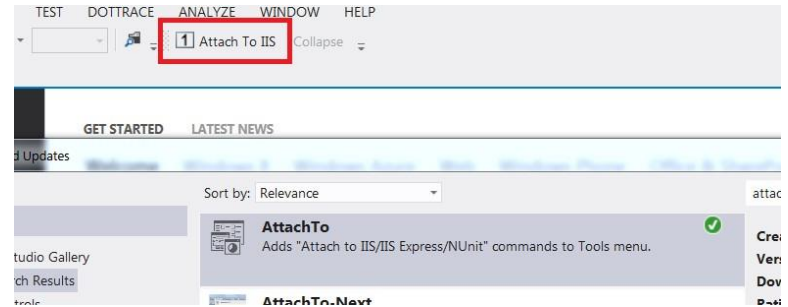
## “Acuminator” Extension

Static code analysis, colorizer and suggestions tool for Acumatica Framework



## “Attach To” Extension

- Attach Debugger to Acumatica with 1-click



# Company Story

---

The Smart Fix company specializes in repairing cell phones of several types. The company provides the following services:

- Battery replacement: This service is provided on customer request and does not require any preliminary diagnostic checks.
- Repair of liquid damage: This service requires a preliminary diagnostic check and a prepayment.
- Screen repair: This service is provided on customer request and does not require any preliminary diagnostic checks.

# Lesson 1: Creating a Customization Project

---

Learning Objectives: In this lesson, you will learn how to

- Create a customization project
- Load a customization project from a local folder
- Bind a customization project to an extension library
- Publish a customization project

## Step 1.1: Creating a Customization Project

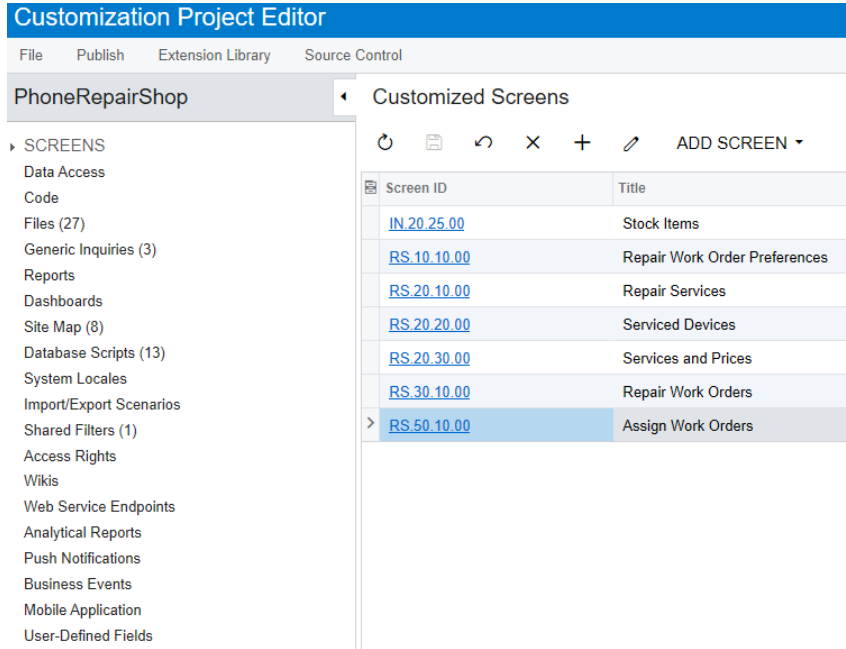
Customization Projects ★

🔄 📁 ↶ + ✕ PUBLISH ▾ UNPUBLISH ALL IMPORT

☰	<input type="checkbox"/>	<b>Add Row</b>	*Project Name	
>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<u>PhoneRepairShop</u>	

1. In Acumatica ERP, open the Customization Projects (SM204505) form.
2. On the form toolbar, click Add Row.
3. In the Project Name column, enter the customization project name: PhoneRepairShop.
4. On the form toolbar, click Save.

## Step 1.2: Loading Items to the Customization Project

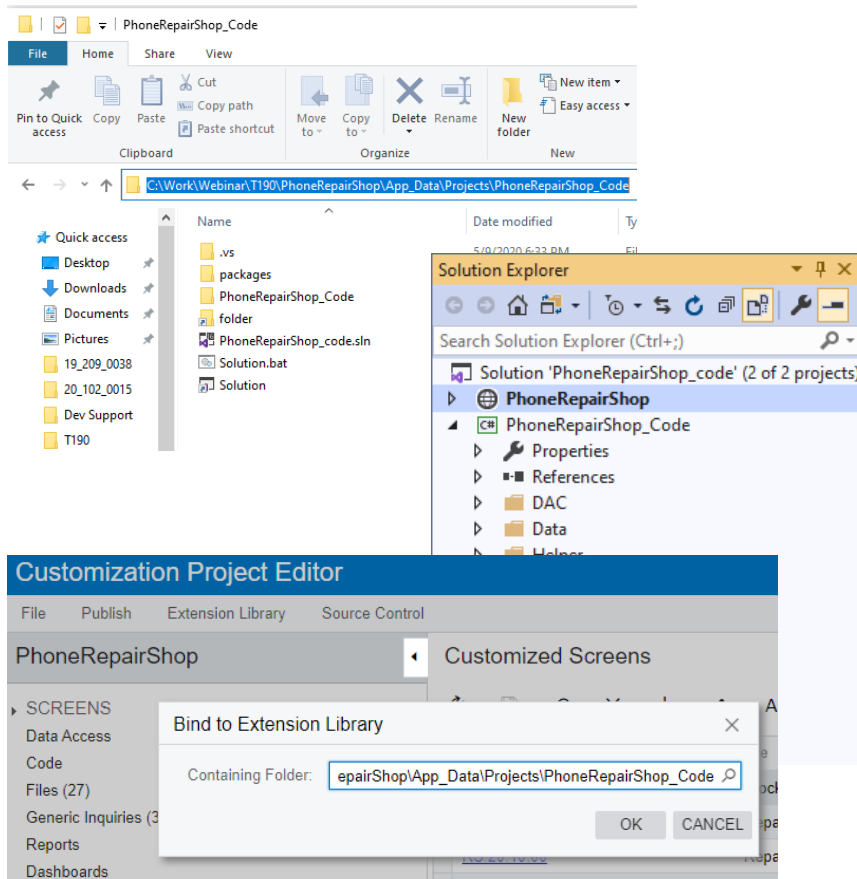


The screenshot shows the 'Customization Project Editor' interface. The top bar is blue with the title 'Customization Project Editor' and a menu with 'File', 'Publish', 'Extension Library', and 'Source Control'. Below the menu, the project name 'PhoneRepairShop' is displayed. On the left, a sidebar lists various categories: SCREENS, Data Access, Code, Files (27), Generic Inquiries (3), Reports, Dashboards, Site Map (8), Database Scripts (13), System Locales, Import/Export Scenarios, Shared Filters (1), Access Rights, Wikis, Web Service Endpoints, Analytical Reports, Push Notifications, Business Events, Mobile Application, and User-Defined Fields. The main area is titled 'Customized Screens' and contains a table with two columns: 'Screen ID' and 'Title'. The table lists several screens, with the last one, 'RS.50.10.00 Assign Work Orders', selected and expanded to show a right arrow icon.

Screen ID	Title
<a href="#">IN.20.25.00</a>	Stock Items
<a href="#">RS.10.10.00</a>	Repair Work Order Preferences
<a href="#">RS.20.10.00</a>	Repair Services
<a href="#">RS.20.20.00</a>	Serviced Devices
<a href="#">RS.20.30.00</a>	Services and Prices
<a href="#">RS.30.10.00</a>	Repair Work Orders
<a href="#">RS.50.10.00</a>	Assign Work Orders

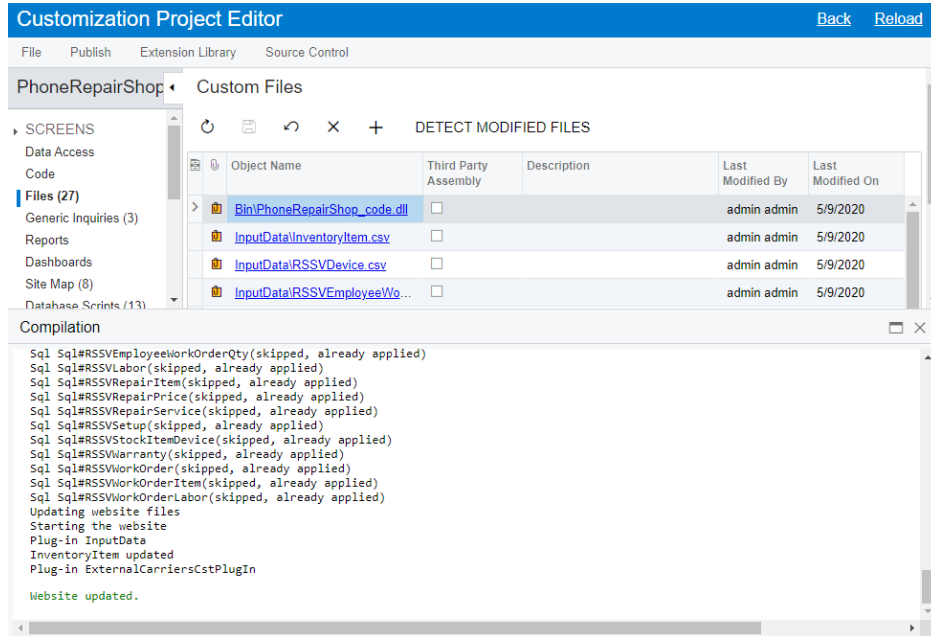
1. On the Customization Projects (SM204505) form, click PhoneRepairShop in the table to open the customization project that you have created.
2. On the menu of the Customization Project Editor, click Source Control > Open Project from Folder.
3. In the dialog box that opens, specify the path to source files of the project.

## Step 1.3: Binding the Extension Library



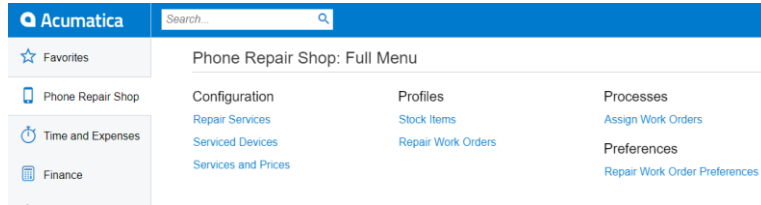
1. Copy PhoneRepairShop\_Code folder to the App\_Data \Projects folder of the Acumatica ERP instance.
2. Open the Visual Studio solution and build the PhoneRepairShop\_Code project.
3. Reopen the PhoneRepairShop project in the Customization Project Editor.
4. On the menu of the Customization Project Editor, click Extension Library > Bind to Existing.
5. In the dialog box that opens, specify the path to the App\_Data\Projects \PhoneRepairShop\_Code folder, and click OK.

# Step 1.4: Publishing the Customization Project

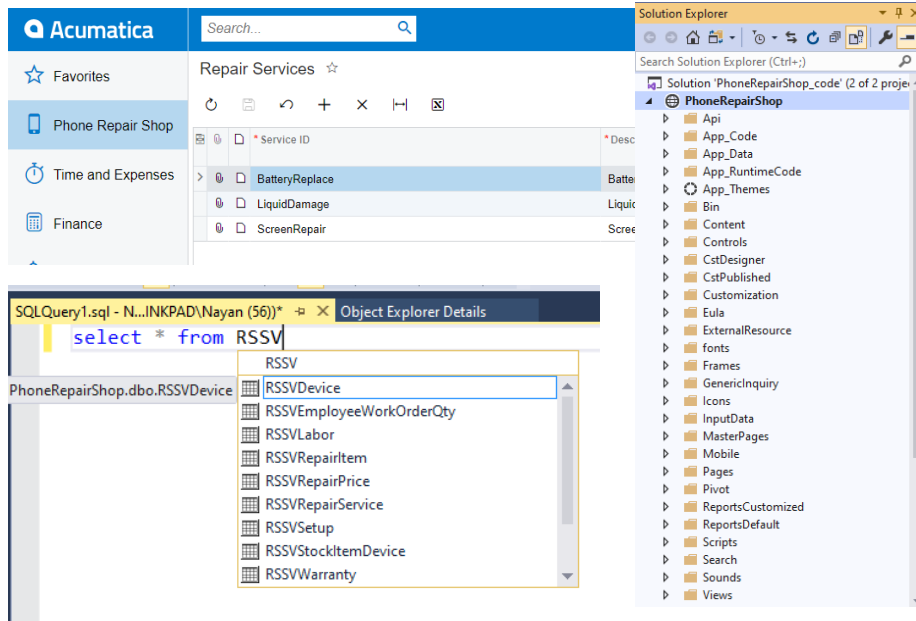


1. With PhoneRepairShop customization loaded in Customization Project Editor, click Files on the left pane.
2. On Custom Files page toolbar, click Detect Modified Files.
3. In the Modified Files Detected dialog box, ensure the Selected check box is selected for the Bin\PhoneRepairShop\_Code.dll file, and click Update Customization Project. Close the dialog box.
4. From the main menu, click Publish > Publish Current Project. The Compilation panel opens, which shows the progress of the publication.
5. Close the Compilation panel when the publication has completed and the Website updated message is displayed.

# Step 1.5: Reviewing the Changes in Acumatica ERP



Review the changes to the Acumatica ERP instance that have been applied as a result of the publication of the PhoneRepairShop customization project





# Lesson 1: Review Questions

---

1. Which of the following actions will publish the customization project opened in the Customization Project Editor to the current Acumatica ERP instance?
  - a) On the menu of the Customization Project Editor, you click Publish > Publish Current Project.
  - b) On the menu of the Customization Project Editor, you click Extension Library > Bind to Existing.
  - c) On the menu of the Customization Project Editor, you click Source Control > Open Project from Folder.

## Lesson 2: Creating Custom Fields

---

Learning Objectives: In this lesson, you will learn how to

- Add a custom column to an Acumatica ERP database table
- Add a custom field to an Acumatica ERP data access class
- Add the control for the custom field to the form

## Task at Hand

---

The Smart Fix company needs to specify that particular stock items on the Stock Items (IN202500) form of Acumatica ERP are repair items and select the type of each repair item.

You will add the following custom controls to the Stock Items (IN202500) form of Acumatica ERP:

- **Repair Item:** A check box that indicates (if selected) that the stock item can be used during the provision of the repair services of the Smart Fix company
- **Repair Item Type:** A drop-down list box for the repair item type, which is one of the following:
  - Battery
  - Screen
  - Screen Cover
  - Back Cover
  - Motherboard

You will add these controls to the Item Defaults section of the General Settings tab of the form.

# Task at Hand

Acumatica

Search...

☆ Favorites

📱 Phone Repair Shop

🕒 Time and Expenses

📊 Finance

💰 Banking

⊖ Payables

⊕ Receivables

📝 Sales Orders

🛒 Purchases

Stock Items ☆

← SAVE & CLOSE ↺ + 🗑️ ⏪ < > ⏩

\* Inventory ID: AACOMPUT01 - Acer Laptop Comput 🔍 Product V

Item Status: Active ▾ Product M

Description: Acer Laptop Computer

GENERAL SETTINGS

PRICE/COST INFO

WAREHOUSE DETAILS

VEN

ITEM DEFAULTS

\* Item Class: STOCKITEM - Stock item 🔍 ✎ Base Un

Type: Finished Good ▾ \* Sales Un

☐ Repair Item \* Purchase

Repair Item Type: ▾

Valuation Method: Battery

\* Tax Category: Screen

\* Posting Class: Screen Cover

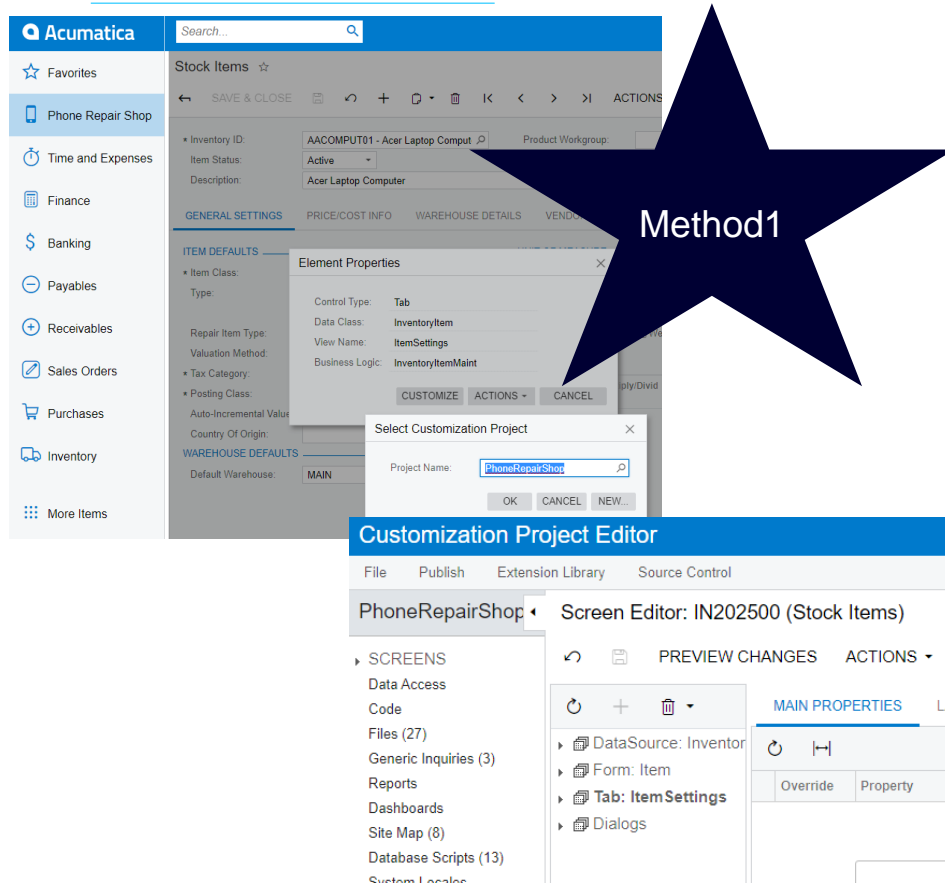
Auto-Incremental Value: Back Cover

Country Of Origin: Motherboard

UNIT OF M

\* From Unit

## Step 2.1: Creating a Custom Column and Field with the Project Editor



**Method1**

**Customization Project Editor**

File Publish Extension Library Source Control

PhoneRepairShop Screen Editor: IN202500 (Stock Items)

SCREENS

- Data Access
- Code
- Files (27)
- Generic Inquiries (3)
- Reports
- Dashboards
- Site Map (8)
- Database Scripts (13)
- System Local

PREVIEW CHANGES ACTIONS

MAIN PROPERTIES LA

DataSource: Inventor

Form: Item

Tab: ItemSettings

Dialogs

Override Property

1. From Stock Items (IN202500) form, open the Screen Editor as follows:
  - a) Click Customization > Inspect Element menu
  - b) Click the name of the General Settings tab to open the Element Properties dialog box
  - c) Click Customize
  - d) In the Select Customization Project dialog box, select the PhoneRepairShop customization project

## Step 2.1: Creating a Custom Column and Field with the Project Editor (cont.)

The screenshot shows the Customization Project Editor interface for the 'PhoneRepairShop' project, specifically the 'Screen Editor: IN202500 (Stock Items)'. The left sidebar lists various project components, including 'Data Access', 'Code', 'Files (27)', 'Generic Inquiries (3)', 'Reports', 'Dashboards', 'Site Map (8)', 'Database Scripts (13)', 'System Locales', 'Import/Export Scenarios', 'Shared Filters (1)', 'Access Rights', 'Wikis', 'Web Service Endpoints', and 'Analytical Reports'. The main area displays the 'PREVIEW CHANGES' tab, showing a tree view of the 'InventoryItemMaint' data source, 'Form: Item', and 'Tab: ItemSettings'. Under 'General Settings', a 'Column' is being added, with a 'Group' containing 'Item Class', 'Type', 'Is a Kit', and 'Valuation Method'. The 'MAIN PROPERTIES' tab shows 'Data View: Inventory Item(ItemSettings)'. The 'LAYOUT PROPERTIES' tab shows 'CREATE CONTROLS', 'NEW FIELD', 'ALL', 'VISIBLE', and 'CUSTOM' tabs. The 'CUSTOM' tab displays a table with columns 'Used', 'Field Name', and 'Control'. The table contains one entry: 'UsrRepairItem (Repair Item)' with a 'CheckBox' control.

Below the Project Editor, the Visual Studio code editor shows the 'InventoryItemExtensions.cs' file. The code defines a namespace 'PX.Objects.IN' and a sealed class 'InventoryItemExt' that inherits from 'PXCacheExtension<PX.Objects.IN.InventoryItem>'. The class contains a region 'UsrRepairItem' with a 'PXBBool' field 'UsrRepairItem' and a 'PXUIField' 'UsrRepairItem'. The 'UsrRepairItem' field is defined as a 'public bool? UsrRepairItem' with 'get' and 'set' methods. The 'UsrRepairItem' field is also defined as a 'public abstract class usrRepairItem' that inherits from 'PX.Data.BQL.BqlBool.Field<usrRepairItem>'.

1. Add custom field **Repair Item** check box in the customization project.
2. Move the data access class (DAC) extension to the PhoneRepairShop\_Code extension library.
3. From the main menu, click Publish > Publish Current Project to apply the customization to the site.
4. Switch to Visual Studio, move the InventoryItemExtensions.cs file to the DAC folder.
5. Add the PXDefault attribute, and build the library.

## Step 2.2: Creating a Control for the Custom Field

The screenshot shows the Acumatica Customization Project Editor interface. The top bar displays 'Customization Project Editor' with 'Back' and 'Reload' buttons. Below the bar, the 'PhoneRepairShop' project is selected, and the 'Screen Editor: IN202500 (Stock Items)' is open. The left sidebar shows the project structure, including 'SCREENS', 'Data Access', 'Code', 'Files (27)', 'Generic Inquiries (3)', 'Reports', 'Dashboards', 'Site Map (8)', 'Database Scripts (13)', 'System Locales', 'Import/Export Scenarios', 'Shared Filters (1)', and 'Access Rights'. The main editor area shows the 'Stock Items' screen with the 'General Settings' tab selected. The 'UsrRepairItem' field is highlighted, and a 'Repair Item' checkbox is being added to the 'UsrRepairItem' field. The bottom section shows the 'Stock Items' form with fields for 'Inventory ID', 'Item Status', 'Description', 'Item Class', 'Type', 'Valuation Method', 'Tax Category', and 'Posting Class'.

Customization Project Editor

File Publish Extension Library Source Control

PhoneRepairShop Screen Editor: IN202500 (Stock Items)

SCREENS

- Data Access
- Code
- Files (27)
- Generic Inquiries (3)
- Reports
- Dashboards
- Site Map (8)
- Database Scripts (13)
- System Locales
- Import/Export Scenarios
- Shared Filters (1)
- Access Rights

PREVIEW CHANGES ACTIONS

MAIN PROPERTIES LAYOUT PROPERTIES ADD DATA FIELDS

Data View: Inventory Item(ItemSettings)

CREATE CONTROLS ALL VISIBLE CUSTOM

Used	Field Name	Control
<input checked="" type="checkbox"/>	UsrRepairItem (Repair Item)	CheckBox

Acumatica

Search...

Stock Items

SAVE & CLOSE

Inventory ID: AACOMPUT01 - Acer Laptop Comput

Item Status: Active

Description: Acer Laptop Computer

GENERAL SETTINGS PRICE/COST INFO WAREHOUSE DETAIL

ITEM DEFAULTS

Item Class: STOCKITEM - Stock item

Type: Finished Good

Repair Item

Valuation Method: Average

Tax Category: EXEMPT - Exempt

Posting Class: STOCKITEM - Stock item

1. Add **Repair Item** check box under General Settings of the Stock Items screen.
2. Publish the PhoneRepairShop project.
3. Refresh the Stock Items form in the browser to view the added control on the General Settings tab of the form.

## Step 2.3: Creating a Custom Column with the Project Editor and a Custom Field with Visual Studio

The image shows two screenshots. The top screenshot is from the 'Customization Project Editor' for 'PhoneRepairShop'. It shows the 'Database Scripts' pane on the left and the 'Add Custom Column to Table' dialog box in the center. The dialog has 'InventoryItem - InventoryItem' selected for the table, 'UsrRepairItemType' for the field name, and 'string' for the data type. A large blue star with the text 'Method 2' is overlaid on this screenshot. The bottom screenshot is from Visual Studio, showing the 'InventoryItemExtensions.cs' file. It contains a region for 'UsrRepairItemType' with a list of strings for repair item types (Battery, Screen, Screen Cover, Back Cover, Motherboard) and a corresponding list of messages. Below this, there is a custom field definition for 'UsrRepairItemType' using the 'PXUIField' class.

Customization Project Editor

File Publish Extension Library Source Control

PhoneRepairShop Database Scripts

SCREENS  
Data Access  
Code  
Files (27)  
Generic Inquiries (3)  
Reports  
Dashboards

ADD X

Method 2

Add Custom Column to Table

\* Table: InventoryItem - InventoryItem

\* Field Name: UsrRepairItemType

\* Data Type: string

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) PhoneRepairShop, 2

InventoryItemExtensions.cs

```
27 #region UsrRepairItemType
28 [PXDBString(2, IsFixed = true)]
29 [PXStringList]
30 new string[]
31 {
32     PhoneRepairShop.RepairItemTypeConstants.Battery,
33     PhoneRepairShop.RepairItemTypeConstants.Screen,
34     PhoneRepairShop.RepairItemTypeConstants.ScreenCover,
35     PhoneRepairShop.RepairItemTypeConstants.BackCover,
36     PhoneRepairShop.RepairItemTypeConstants.Motherboard
37 },
38 new string[]
39 {
40     PhoneRepairShop.Messages.Battery,
41     PhoneRepairShop.Messages.Screen,
42     PhoneRepairShop.Messages.ScreenCover,
43     PhoneRepairShop.Messages.BackCover,
44     PhoneRepairShop.Messages.Motherboard
45 })]
46 [PXUIField(DisplayName = "Repair Item Type")]
47 public string UsrRepairItemType { get; set; }
48 public abstract class UsrRepairItemType : PX.Data.BQL.BqlString.Field<UsrRepairItemType> { }
49 #endregion
50 }
51 }
```

1. Create custom column and field for **Repair Item Type** dropdown as follows:
  - a) In the Customization Project Editor, select Database Scripts from the left pane.
  - b) Add field to InventoryItem table using **Add > Custom Column** to Table from the Database Scripts toolbar.
  - c) Publish the project.
2. Switch to Visual Studio and define the RepairItemTypeConstants class to hold various repair item types, viz. Battery, Screen, Screen Cover, Back Cover, and Motherboard.
3. In the InventoryItemExt class of the InventoryItemExtensions.cs file, add a custom field for the Repair Item Type dropdown and build the project.



## Step 2.4: Creating a Control for the Custom Field—Self-Guided Exercise

**Acumatica** Search...

★ Favorites

📱 Phone Repair Shop

🕒 Time and Expenses

📊 Finance

💰 Banking

⊖ Payables

⊕ Receivables

📝 Sales Orders

🛒 Purchases

**Stock Items** ☆

← SAVE & CLOSE 📄 ↶ + 📄 ▾ 🗑️ ⏪ < >

\* Inventory ID: AACOMPUT01 - Acer Laptop Comput 🔍 Product V

Item Status: Active ▾ Product M

Description: Acer Laptop Computer

GENERAL SETTINGS PRICE/COST INFO WAREHOUSE DETAILS VEN

**ITEM DEFAULTS**

\* Item Class: STOCKITEM - Stock item 🔍 ✎ Base Un

Type: Finished Good ▾ \* Sales Un

☐ Repair Item \* Purchase

Repair Item Type: ▾

Valuation Method: Battery

\* Tax Category: Screen

\* Posting Class: Screen Cover

Auto-Incremental Value: Back Cover

Country Of Origin: Motherboard

UNIT OF M

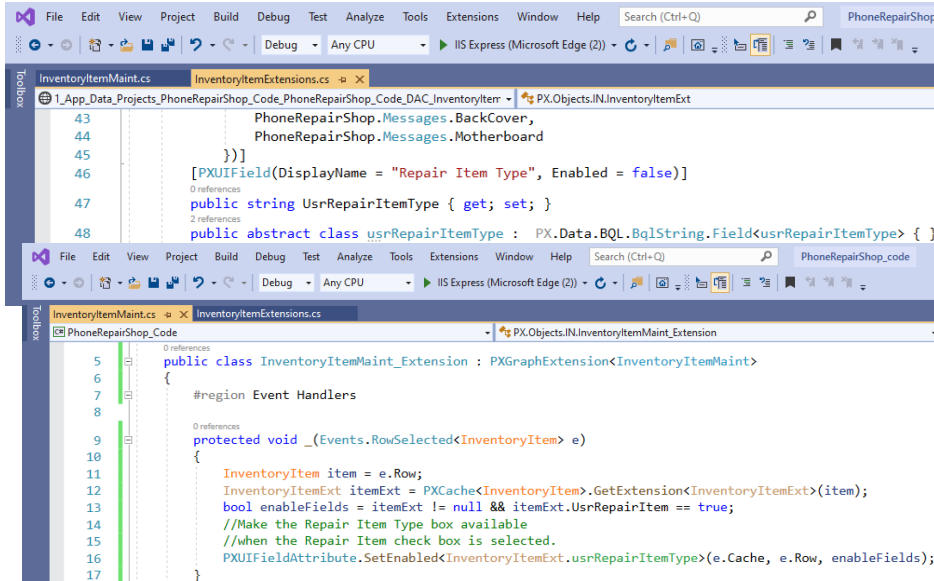
\* From Unit

## Step 2.5: Making the Custom Field Conditionally Available (with RowSelected)

---

- **RowSelected** event is intended for implementing UI presentation logic.
- Event takes 2 parameters viz.
  - PXCache sender
  - PXRowSelectedEventArgs e
- Event is triggered for
  - Displaying record in the UI
  - Execution of following PXCache class methods:
    - Locate
    - Insert
    - Update
    - Delete
- Avoid executing BQL statements in a RowSelected event handler, because this event is called multiple times for the same record and can impact performance.
- Execution order is
  - Attribute event handlers
  - Graph event handlers

## Step 2.5: Making the Custom Field Conditionally Available (with RowSelected) (cont.)



```
InventoryItemExtensions.cs
43
44
45
46
47
48
[PXUIField(DisplayName = "Repair Item Type", Enabled = false)]
public string UsrRepairItemType { get; set; }
public abstract class usrRepairItemType : PX.Data.BQL.BqlString.Field<usrRepairItemType> { }
```

```
InventoryItemMaint_Extension.cs
5
6
7
8
9
10
11
12
13
14
15
16
17
public class InventoryItemMaint_Extension : PXGraphExtension<InventoryItemMaint>
{
    #region Event Handlers
    protected void _(Events.RowSelected<InventoryItem> e)
    {
        InventoryItem item = e.Row;
        InventoryItemExt itemExt = PXCACHE<InventoryItem>.GetExtension<InventoryItemExt>(item);
        bool enableFields = itemExt != null && itemExt.UsrRepairItem == true;
        //Make the Repair Item Type box available
        //when the Repair Item check box is selected.
        PXUIFieldAttribute.SetEnabled<InventoryItemExt.usrRepairItemType>(e.Cache, e.Row, enableFields);
    }
}
```

1. In Visual Studio, set **Enabled = false** of the PXUIField attribute for *UsrRepairItemType* field.
2. Add RowSelected event handler for Stock Items form using Screen Editor.
3. Move generated template code to Visual Studio project.
4. Set **CommitChanges = True** for *UsrRepairItem* field.
5. Build Visual Studio project.
6. Update customization project with updated dll.
7. Re publish customization.

## Step 2.6: Testing the Customized Form

☆ Favorites

Phone Repair Shop

Time and Expenses

Finance

Banking

Payables

Receivables

Sales Orders

Purchases

Stock Items ☆

←

SAVE & CLOSE

+

IK

\* Inventory ID:

AACOMPUT01 - Acer Laptop Comput

Item Status:

Active

Description:

Acer Laptop Computer

GENERAL SETTINGS

PRICE/COST INFO

WAREHOUSE DETAIL

ITEM DEFAULTS

\* Item Class:

STOCKITEM - Stock item

Type:

Finished Good

☒ Repair Item

Repair Item Type:

Battery

Screen

Screen Cover

Back Cover

Motherboard

Valuation Method:

\* Tax Category:

\* Posting Class:

Auto-Incremental Value:

Country Of Origin:

## Lesson 2: Review Questions

---

1. Select all objects that together make up the minimum set of objects that are customized when you add a control for a custom field with the DBTableColumn storage type to an Acumatica ERP form.
  - a) The database table
  - b) The data access class
  - c) The graph
  - d) The .aspx page
  
2. Suppose that you have to add a control for a custom field to a form of Acumatica ERP. Select the tool of the Customization Project Editor that is designed to do this.
  1. Customization Menu
  2. Data Class Editor
  3. Screen Editor
  4. Project Items Editor
  5. Project XML Editor
  
3. Which event handler would you use to configure the UI presentation logic?
  1. FieldUpdated
  2. RowUpdated
  3. RowSelected

## Lesson 3: Implementing the Update and Validation of Field Values

---

Learning Objectives: In this lesson, you will learn how to

- Update the fields of a data record on update of a field of this record
- Validate the value of a field that does not depend on the values of other fields of the same record

# Task at Hand

---

Modify business logic of forms:

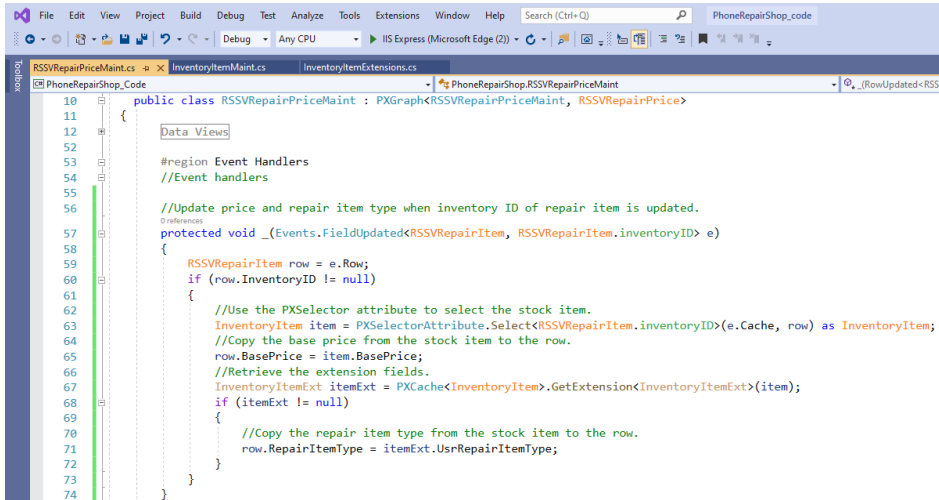
- **Services and Prices (RS203000)**
  - Copy/set Repair Item Type and Price columns on Repair Items tab from Stock Items when Inventory ID is selected.
- **Repair Work Orders (RS301000)**
  - Validate Quantity column on Labor tab such that
    - $\geq 0$
    - $\geq$  Quantity column on Labor tab of Services and Prices form

## Step 3.1: Updating Fields of a Record on Update of a Field of This Record (with FieldUpdated)

- **FieldUpdated** event is intended for implementing business logic associated with updating the value of the DAC field in following cases:
  - Assigning or updating related fields of the record
  - Updating any of the followings:
    - Detail records in 1:M relationship
    - Detail records in 1:1 relationship
    - Master records in M:1 relationship
- Event takes 2 parameters viz.
  - PXCache sender
  - PXFieldUpdatedEventArgs e
- Event is triggered for
  - Field value updated from UI or API
  - Key fields value when deleting a record
  - Execution of following methods:
    - Insert
    - SetDefaultExt
    - Update
    - SetValueExt
    - Locate (key fields)
    - Delete (key fields)
- Execution order is
  - Attribute event handlers
  - Graph event handlers



## Step 3.1: Updating Fields of a Record on Update of a Field of This Record (with FieldUpdated) (cont.)



```
10 public class RSSVRepairPriceMaint : PXGraph<RSSVRepairPriceMaint, RSSVRepairPrice>
11 {
12     [Data View]
13
14     #region Event Handlers
15     //Event handlers
16
17     //Update price and repair item type when inventory ID of repair item is updated.
18     protected void _(Events.FieldUpdated<RSSVRepairItem, RSSVRepairItem.inventoryID> e)
19     {
20         RSSVRepairItem row = e.Row;
21         if (row.InventoryID != null)
22         {
23             //Use the PXSelector attribute to select the stock item.
24             InventoryItem item = PXSelectorAttribute.Select<RSSVRepairItem.inventoryID>(e.Cache, row) as InventoryItem;
25             //Copy the base price from the stock item to the row.
26             row.BasePrice = item.BasePrice;
27             //Retrieve the extension fields.
28             InventoryItemExt itemExt = PXCache<InventoryItem>.GetExtension<InventoryItemExt>(item);
29             if (itemExt != null)
30             {
31                 //Copy the repair item type from the stock item to the row.
32                 row.RepairItemType = itemExt.UsrRepairItemType;
33             }
34         }
35     }
36 }
```

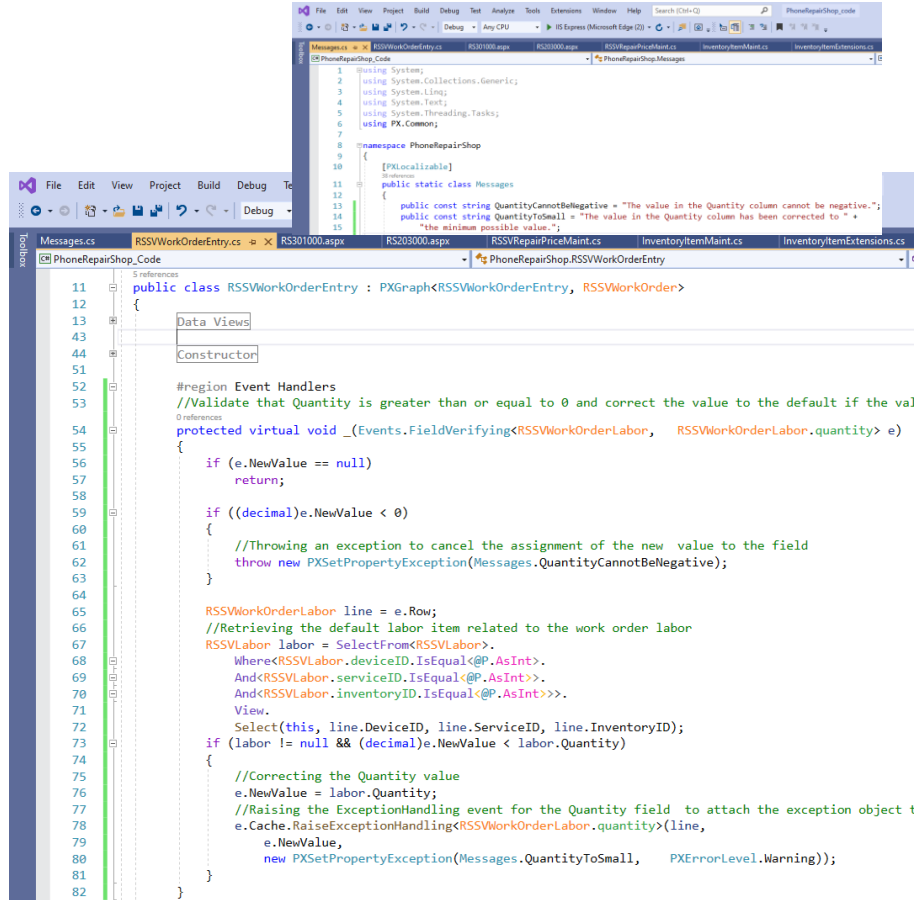
1. In Visual Studio, add FieldUpdated event for *RSSVRepairItem.InventoryID* field.
2. Set **CommitChanges = True** for InventoryID control of the Repair Items tab item
3. Build Visual Studio project.
4. Update customization project with updated dll.
5. Re publish customization.

## Step 3.2: Validating an Independent Field Value (with FieldVerifying)

---

- **FieldVerifying** event is intended for
  - Implementing business logic associated with validating the value of the DAC field.
  - Cancel the assigning of new value
  - Convert external presentation to internal presentation and implement associated business logic.
- Event takes 2 parameters viz.
  - PXCache sender
  - PXFieldVerifyingEventArgs e
- Event is triggered for
  - Field value inserted or updated from UI or API
  - Execution of following methods:
    - Insert
    - SetDefaultExt
    - Update
    - SetValueExt
    - Locate (key fields)
- Execution order is
  - Graph event handlers
  - Attribute event handlers

## Step 3.2: Validating an Independent Field Value (with FieldVerifying) (cont.)



The screenshot shows the Visual Studio IDE with the 'PhoneRepairShop\_Code' project open. The 'Messages.cs' file is visible in the background, showing message constants. The foreground shows the 'RssvWorkOrderEntry.cs' file, which is a partial class for 'RssvWorkOrderEntry' inheriting from 'PXGraph<RssvWorkOrderEntry, RssvWorkOrder>'. The code includes a 'Data Views' section, a 'Constructor' section, and a '#region Event Handlers' section. The 'FieldVerifying' event is implemented for 'RssvWorkOrderEntry.Quantity'. The event handler checks if the new value is null or less than 0. If null, it returns. If less than 0, it throws a 'PXSetPropertyException' with the message 'QuantityCannotBeNegative'. The code also includes a query to retrieve the default labor item related to the work order labor, and a check to ensure the new value is not less than the labor item's quantity. If the check fails, it corrects the value and raises the 'Cache.RaiseExceptionHandling' event.

```
11 public class RssvWorkOrderEntry : PXGraph<RssvWorkOrderEntry, RssvWorkOrder>
12 {
13     Data Views
14     Constructor
15
16     #region Event Handlers
17     //Validate that Quantity is greater than or equal to 0 and correct the value to the default if the val
18     protected virtual void _(Events.FieldVerifying<RssvWorkOrderLabor, RssvWorkOrderLabor.Quantity> e)
19     {
20         if (e.NewValue == null)
21             return;
22
23         if ((decimal)e.NewValue < 0)
24         {
25             //Throwing an exception to cancel the assignment of the new value to the field
26             throw new PXSetPropertyException(Messages.QuantityCannotBeNegative);
27         }
28
29         RssvWorkOrderLabor line = e.Row;
30         //Retrieving the default labor item related to the work order labor
31         RssvLabor labor = SelectFrom<RssvLabor>.
32             Where<RssvLabor.deviceID.IsEqual<@P.AsInt>|.
33             And<RssvLabor.serviceID.IsEqual<@P.AsInt>|.
34             And<RssvLabor.inventoryID.IsEqual<@P.AsInt>>>.
35             View.
36             Select<this, line.DeviceID, line.ServiceID, line.InventoryID>;
37         if (labor != null && (decimal)e.NewValue < labor.Quantity)
38         {
39             //Correcting the Quantity value
40             e.NewValue = labor.Quantity;
41             //Raising the ExceptionHandling event for the Quantity field to attach the exception object t
42             e.Cache.RaiseExceptionHandling<RssvWorkOrderLabor.Quantity>(line,
43                 e.NewValue,
44                 new PXSetPropertyException(Messages.QuantityToSmall, PXErrorLevel.Warning));
45         }
46     }
47 }
```

1. In Visual Studio, define required message constants.
2. Add FieldVerifying event for *RssvWorkOrderLabor.Quantity* field.
3. Set **CommitChanges = True** for Quantity control of the Labor tab item
4. Build Visual Studio project.
5. Update customization project with updated dll.
6. Re publish customization.

# Lesson 3: Review Questions

---

1. Which of the following objects would you use to throw an exception to cancel the assignment of a new value to a field?
  - a) e.Cancel of the FieldVerifying event handler
  - b) PXSetPropertyException
  - c) RaiseExceptionHandling
  
2. Which event handler should be used to validate an independent field value?
  - a) FieldDefaulting
  - b) FieldSelecting
  - c) FieldVerifying
  - d) FieldUpdated
  - e) RowSelected
  
3. Which event handler is used to update a value of a dependent field within a particular data record?
  - a) FieldDefaulting
  - b) FieldSelecting
  - c) FieldVerifying
  - d) FieldUpdated
  - e) RowSelected
  
4. 4. How would you specify a required integer parameter in a fluent BQL query?
  - a) @P.AsInt
  - b) Argument.AsInt
  - c) @P
  - d) Argument

## Lesson 4: Creating an Acumatica ERP Entity Corresponding to a Custom Entity

---

Learning Objectives: In this lesson, you will learn how to

- Learn about the usage of the PXLongOperation class
- Define the Create Invoice action and the related button on the Repair Work Orders (RS301000) form
- Configure the availability of the Create Invoice button

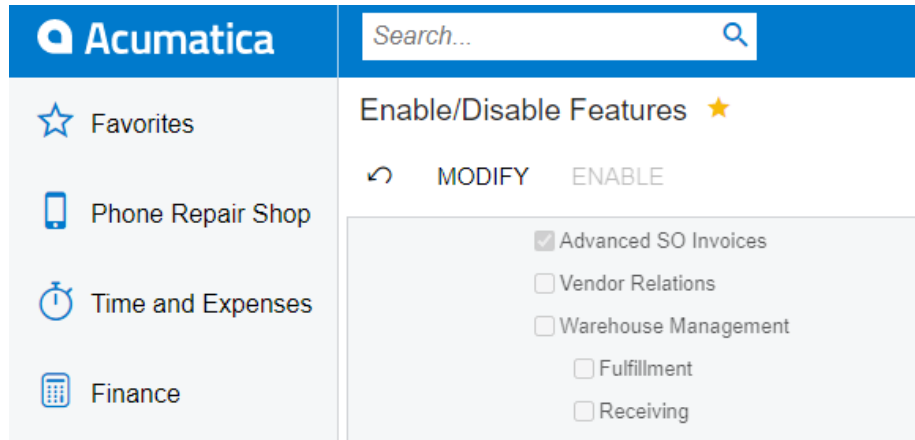
# Task at Hand

---

Add business logic to generate invoice from Repair Work Order.

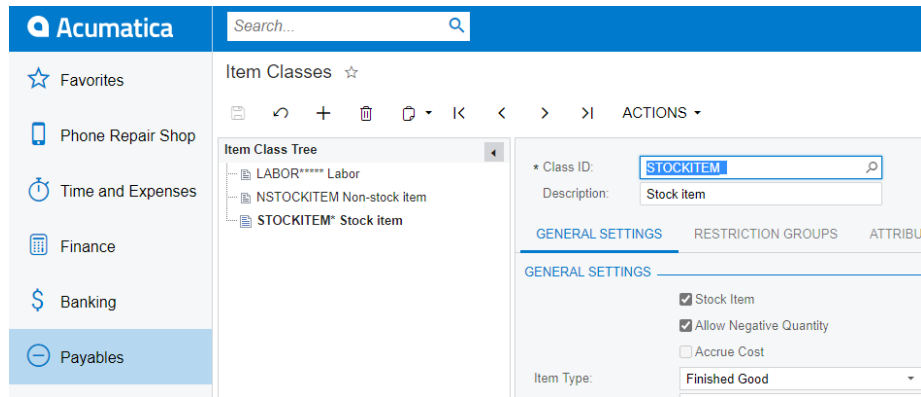
- Add “Create Invoice” button on the toolbar
- Make this button available only when repair work order status equals
  - Completed or
  - Requires Prepayment
- Invoice can be created only once per repair work order

## Step 4.1: Performing Preliminary Steps



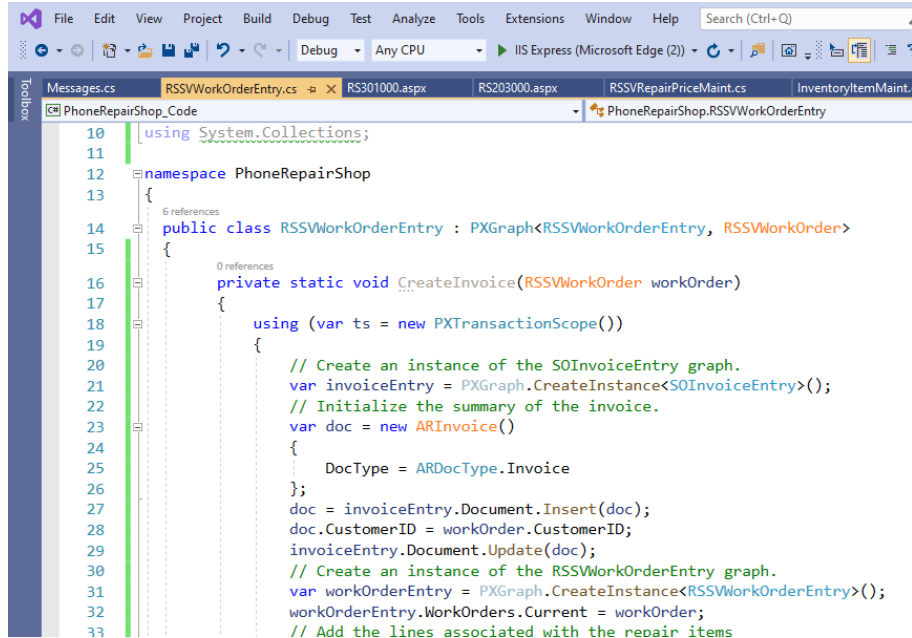
The screenshot shows the Acumatica 'Enable/Disable Features' interface. On the left is a sidebar with navigation links: Favorites, Phone Repair Shop, Time and Expenses, and Finance. The top header has the Acumatica logo and a search bar. The main content area is titled 'Enable/Disable Features' with a star icon. Below the title are buttons for 'MODIFY' and 'ENABLE'. A list of features is shown with checkboxes: 'Advanced SO Invoices' (checked), 'Vendor Relations' (unchecked), 'Warehouse Management' (unchecked), 'Fulfillment' (unchecked), and 'Receiving' (unchecked).

1. Enable *Advanced SO Invoices* feature
2. Allow negative quantities for stock items.



The screenshot shows the Acumatica 'Item Classes' configuration screen. The left sidebar includes links for Favorites, Phone Repair Shop, Time and Expenses, Finance, Banking, and Payables. The top header has the Acumatica logo and a search bar. The main content area is titled 'Item Classes' with a star icon. Below the title are navigation icons and an 'ACTIONS' dropdown. The 'Item Class Tree' on the left lists 'LABOR\*\*\*\*\* Labor', 'NSTOCKITEM Non-stock item', and 'STOCKITEM\* Stock item'. The right panel shows the configuration for the 'STOCKITEM' class, with fields for 'Class ID' (STOCKITEM) and 'Description' (Stock item). The 'GENERAL SETTINGS' tab is active, showing checkboxes for 'Stock Item' (checked), 'Allow Negative Quantity' (checked), and 'Accrue Cost' (unchecked). The 'Item Type' dropdown is set to 'Finished Good'.

## Step 4.2: Defining the Logic of Creating an SO Invoice

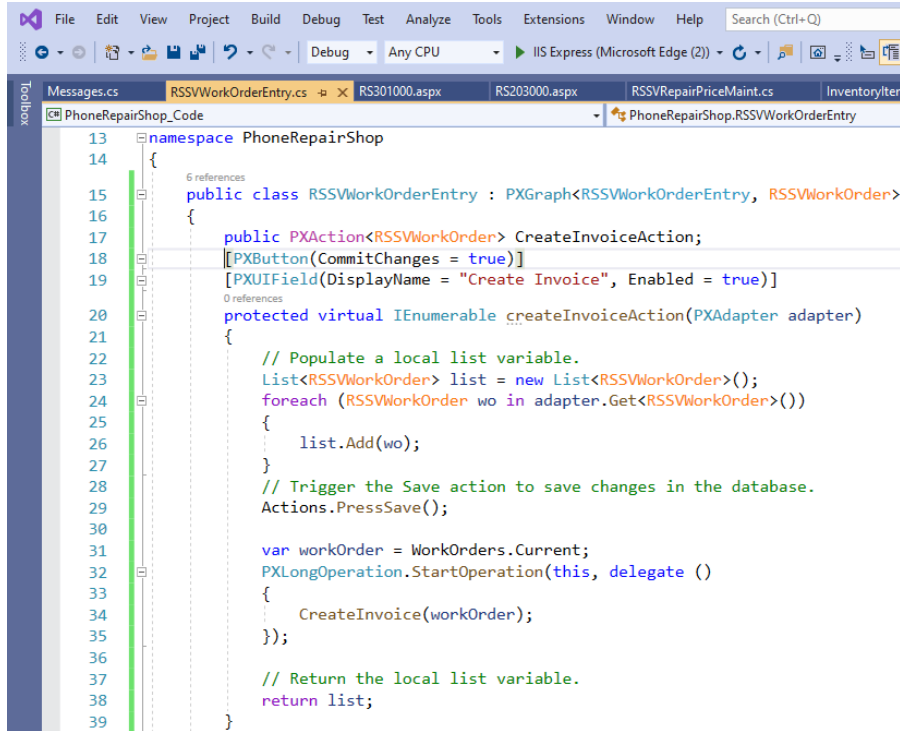


```
10 using System.Collections;
11
12 namespace PhoneRepairShop
13 {
14     public class RSSVWorkOrderEntry : PXGraph<RSSVWorkOrderEntry, RSSVWorkOrder>
15     {
16         private static void CreateInvoice(RSSVWorkOrder workOrder)
17         {
18             using (var ts = new PXTransactionScope())
19             {
20                 // Create an instance of the SOInvoiceEntry graph.
21                 var invoiceEntry = PXGraph.CreateInstance<SOInvoiceEntry>();
22                 // Initialize the summary of the invoice.
23                 var doc = new ARInvoice()
24                 {
25                     DocType = ARDocType.Invoice
26                 };
27                 doc = invoiceEntry.Document.Insert(doc);
28                 doc.CustomerID = workOrder.CustomerID;
29                 invoiceEntry.Document.Update(doc);
30                 // Create an instance of the RSSVWorkOrderEntry graph.
31                 var workOrderEntry = PXGraph.CreateInstance<RSSVWorkOrderEntry>();
32                 workOrderEntry.WorkOrders.Current = workOrder;
33                 // Add the lines associated with the repair items
```

1. Add **CreateInvoice** method to RSSVWorkOrderEntry graph. This method should be static as it will be called from PXLongOperation.



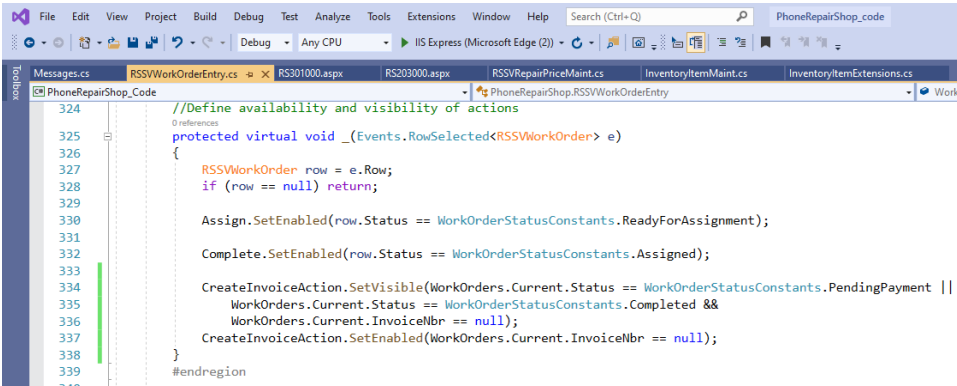
## Step 4.3: Defining the Create Invoice Action



```
13 namespace PhoneRepairShop
14 {
15     public class RSSVWorkOrderEntry : PXGraph<RSSVWorkOrderEntry, RSSVWorkOrder>
16     {
17         public PXAction<RSSVWorkOrder> CreateInvoiceAction;
18         [[PXButton(CommitChanges = true)]]
19         [PXUIField(DisplayName = "Create Invoice", Enabled = true)]
20         protected virtual IEnumerable createInvoiceAction(PXAdapter adapter)
21         {
22             // Populate a local list variable.
23             List<RSSVWorkOrder> list = new List<RSSVWorkOrder>();
24             foreach (RSSVWorkOrder wo in adapter.Get<RSSVWorkOrder>())
25             {
26                 list.Add(wo);
27             }
28             // Trigger the Save action to save changes in the database.
29             Actions.PressSave();
30
31             var workOrder = WorkOrders.Current;
32             PXLongOperation.StartOperation(this, delegate ()
33             {
34                 CreateInvoice(workOrder);
35             });
36
37             // Return the local list variable.
38             return list;
39         }
40     }
41 }
```

1. Add **CreateInvoiceAction** action to RSSVWorkOrderEntry graph. This adds “Create Invoice” button to the form toolbar.

## Step 4.4: Defining the Visibility and Availability of the Create Invoice Action



```
//Define availability and visibility of actions
protected virtual void _(Events.RowSelected<RSSVWorkOrder> e)
{
    RSSVWorkOrder row = e.Row;
    if (row == null) return;

    Assign.SetEnabled(row.Status == WorkOrderStatusConstants.ReadyForAssignment);

    Complete.SetEnabled(row.Status == WorkOrderStatusConstants.Assigned);

    CreateInvoiceAction.SetVisible(WorkOrders.Current.Status == WorkOrderStatusConstants.PendingPayment ||
        WorkOrders.Current.Status == WorkOrderStatusConstants.Completed &&
        WorkOrders.Current.InvoiceNbr == null);
    CreateInvoiceAction.SetEnabled(WorkOrders.Current.InvoiceNbr == null);
}
#endregion
```

1. Add **RowSelected** event to RSSVWorkOrderEntry graph to control visibility and availability.
2. Finally, rebuild Visual Studio project.

## Step 4.5: Testing the Create Invoice Action

---

## Lesson 4: Review Questions

---

1. What attribute do you use to set up a button on the user interface?
  - a) PXButton
  - b) PXAction
  - c) PXUIField
  
2. How do you configure the visibility of an action at run time?
  - a) By configuring the ASPX page
  - b) By calling the SetVisible method
  - c) By setting the Visible property value to True

## Lesson 5: Deriving the Value of a Custom Field from Another Entity

---

Learning Objectives: In this lesson, you will learn how to

- Derive the value for a custom field from another form

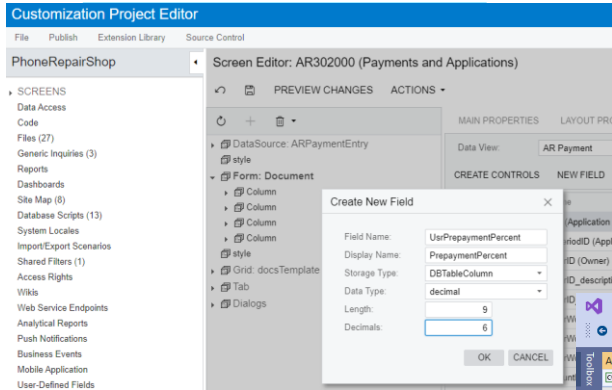
## Task at Hand

---

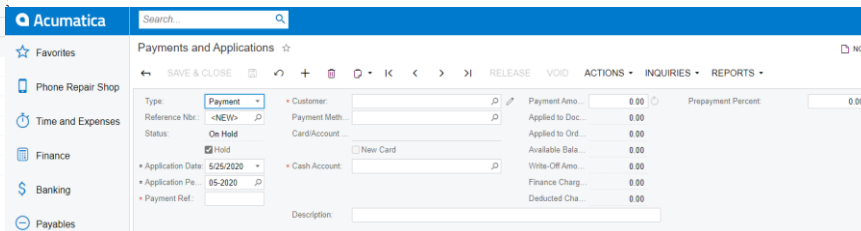
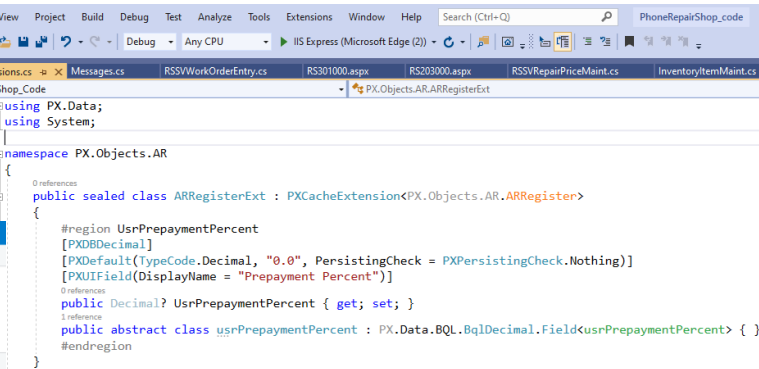
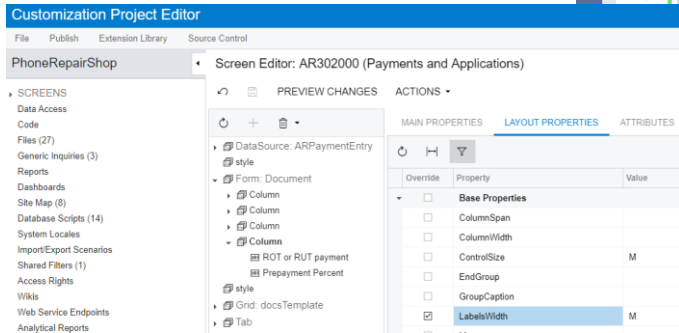
Add business logic to generate invoice from Repair Work Order.

- Default prepayment % on Payments and Application form from “Prepayment Percent” field on Repair Work Order Preferences (RS101000) form.

# Step 5.1: Adding a Custom Field to the Payments and Applications Form—Self-Guided Exercise



1. Add new field *UsrPrepaymentPercent* by extending ARPayment DAC.
2. Add this new field to Summary area of the Payments and Applications form.



## Step 5.2: Deriving the Default Value of the PrepaymentPercent Field

---

This can be done in 2 different ways:

1. Using FieldDefaulting event
2. Using PXDefault attribute



## Step 5.2: Deriving the Default Value of the PrepaymentPercent Field (cont.)

---

- **FieldDefaulting** event is used to generate and assign the default value to DAC field.
- Event takes 2 parameters viz.
  - PXCache sender
  - PXFieldDefaultingEventArgs e
- Event is triggered for
  - Row inserted from UI or API
  - Execution of following methods:
    - Insert
    - SetDefaultExt
- Execution order is
  - Graph event handlers
  - Attribute event handlers

## Step 5.2: Deriving the Default Value of the PrepaymentPercent Field (cont.)

- **PXDefault** attribute is used to generate and assign the default value to DAC field.
- This attribute also allows to make DAC field a required field.
- Attribute is triggered for
  - Row inserted from UI or API
  - Execution of following methods:
    - Insert
    - SetDefaultExt
- Examples:
  - Set default value  

```
[PXDefault(false)]
```

```
[PXDefault(TypeCode.Decimal, "0.0")]
```

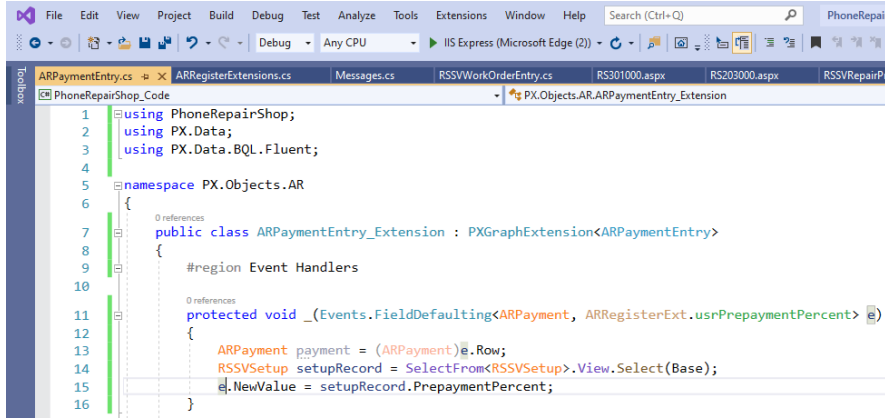
```
[PXDefault(typeof(Search<CAEntryType.referenceID, Where<CAEntryType.entryTypeID, Equal<Current<AddTrxFooter.entryTypeID>>>>)))]
```

```
[PXDefault( false, typeof(Select<VendorClass, Where<VendorClass.vendorClassID, Equal<Current<Vendor.vendorClassID>>>>), SourceField = typeof(VendorClass.allowOverrideRate)))]
```
  - Make required field  

```
[PXDefault]
```

## Step 5.2: Deriving the Default Value using FieldDefaulting



```
1 using PhoneRepairShop;
2 using PX.Data;
3 using PX.Data.BQL.Fluent;
4
5 namespace PX.Objects.AR
6 {
7     public class ARPaymentEntry_Extension : PXGraphExtension<ARPaymentEntry>
8     {
9         #region Event Handlers
10
11         protected void _(Events.FieldDefaulting<ARPayment, ARRegisterExt.usrPrepaymentPercent> e)
12         {
13             ARPayment payment = (ARPayment)e.Row;
14             RSSVSetup setupRecord = SelectFrom<RSSVSetup>.View.Select(Base);
15             e.NewValue = setupRecord.PrepaymentPercent;
16         }
17     }
18 }
```

1. Extend Payment and Application form's graph.
2. Add FieldDefaulting event for *UsrPrepaymentPercent* field of the ARPayment extension.

## Step 5.3: Testing the Deriving of the Field Value

---

## Lesson 5: Review Questions

---

1. Which ways can you use to derive the value of a custom field from a custom entity to an Acumatica ERP entity? Select all applicable ways.
  - a) Use the PXDefault attribute
  - b) Use the FieldDefaulting event handler
  - c) Use the RowSelected event handler

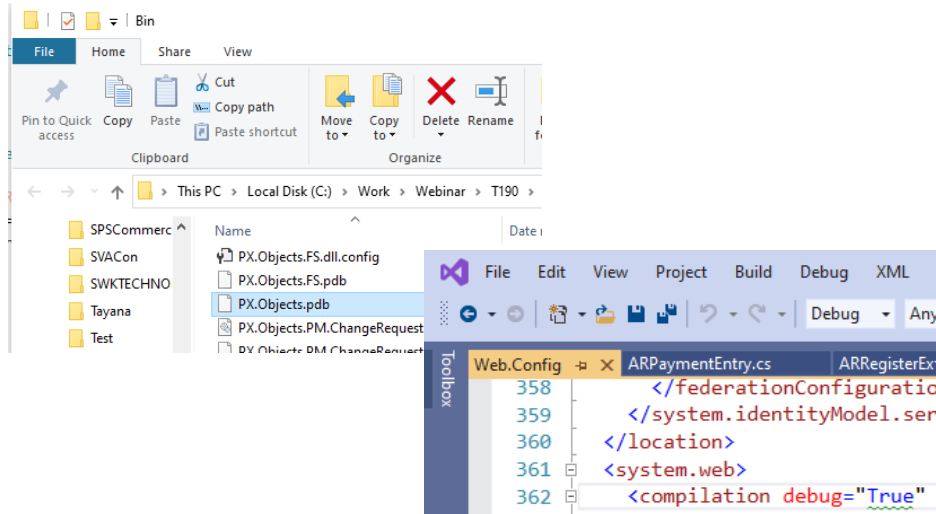
## Lesson 6: Debugging Customization Code

---

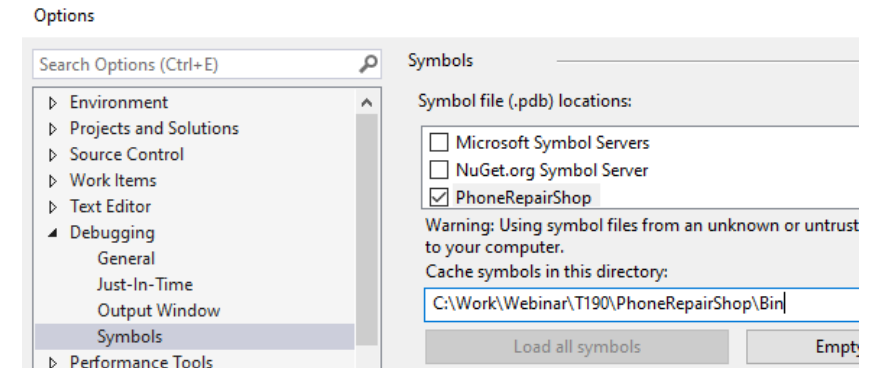
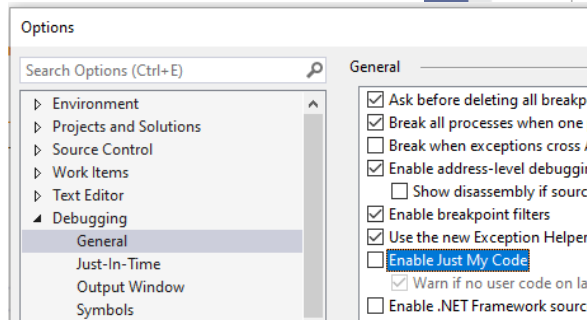
Learning Objectives: In this lesson, you will learn how to

- Debug the source code of Acumatica ERP

## Step 6.1: Debugging the Acumatica ERP Source Code



1. Ensure PDB files are present in the instance's bin folder.
2. Configure web.config file  
`<system.web> <compilation debug="True" ...>`
3. Clear **Enable Just My Code** check box under Debugging > General in Visual Studio.
4. Specify symbols file location.



## Step 6.1: Debugging the Acumatica ERP Source Code (cont.)

1. Attach Visual Studio debugger to a running process.
2. Release payment for some repair work order.

Payments and Applications

The screenshot displays the Acumatica ERP 'Payments and Applications' window on the left and the Visual Studio code editor on the right. The ERP window shows a 'Payment' entry for 'Jersey Central Office Equi' with a payment amount of 50.00. The Visual Studio editor shows the source code for 'ARPaymentEntry.cs', specifically the 'Release' method. The code includes a 'PXCache' and a 'List<ARRegister>' to process the release. A tooltip for 'List<ARRegister>' is visible, stating 'Initializes a new instance'.

Visual Studio Code Editor Content:

```
860  
861 [PXUIField(DisplayName = "Release", MapEnableRights = PXCacher  
862 [PXProcessButton]  
863 [ARMigrationModeDependentActionRestriction(  
864     restrictInMigrationMode: false,  
865     restrictForRegularDocumentInMigrationMode: true,  
866     restrictForUnreleasedMigratedDocumentInNormalMode: true)]  
867  
868 0 references  
869 public override IEnumerable Release(PXAdapter adapter)  
870 {  
871     PXCache cache = Document.Cache;  
872     List<ARRegister> list = new List<ARRegister>();  
873     foreach (ARPayment ardoc in adapter.Get<ARPayment>())  
874     {  
875         if (!bool)ardoc.Hold)  
876         {  
877             Update(ardoc);  
878         }  
879     }  
880 }
```



## Lesson 6: Review Questions

---

1. Where are the Acumatica ERP PDB files located?
  - a) On the Partner Portal
  - b) In the Files/Bin folder of the Acumatica ERP installation folder
  - c) In the Bin folder of the customization project



# Thank you!

---

We would love to hear your feedback on online training.  
Please complete the survey by the following link:  
<https://www.surveymonkey.com/r/DevOnlineTraining2020>