



T190 Quick Start in Customization

Vidhyalakshmi Hariharasubramanian

Sr. Technical Account Manager

Timing and Agenda

March 23, 2023 -10:00-11:30 AM

Day 1

Lesson 1: Creating a Customization Project

Lesson 2: Creating Custom Fields

March 24, 2023 -10:00-11:30 AM

Day 2

Lesson 3: Implementing the Update and Validation of Field Values

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

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

Lesson 6: Debugging Customization Code



Day 1

Joe Gibbs Racing
Acumatica Partner



Customization Projects

Introduction – Customization Project

- A customization project is a *set of changes to the user interface, configuration data, and functionality* of Acumatica ERP.
- The customization project holds the changes that have been made for a particular customization, which might include changes to the mobile site map, generic inquiries, and the properties of UI elements.
- To apply the content of a customization project to an instance of Acumatica ERP, you must **publish** the project.

Designing the application involves:

1. **Designing Database Structure and DACs** – Taking care of naming conventions for Tables (DACs) and Columns (Fields), deciding Primary Key and relationships, audit fields and other fields for concurrency control (*Tstamp*), attachments (*NoteID*), multitenancy support (*CompanyID* and *CompanyMask*), multiple branch support (*BranchID* and *UsrBranchID*).
2. **Designing the User Interface** – Taking care of naming/numbering of Forms/Reports, designer setup, item grouping, configuring the aspx and several elements like containers, tabs, layout, etc.
3. **Designing Graphs and Event Handlers** – deciding on names of graphs and event handlers and graph suffixes, inserting/updating/deleting data records, saving changes to database, etc.

Introduction - Customization Projects

Database

Schema

new tables, new columns in existing tables, and other new database objects

Data

custom or modified reports and changes in the application configuration (modified site map, user access roles, locales, and other new items stored in the database)

Customization project

File System

Custom files

new DLLs, custom ASPX pages, and other files required for the customized product

ASPX file changes

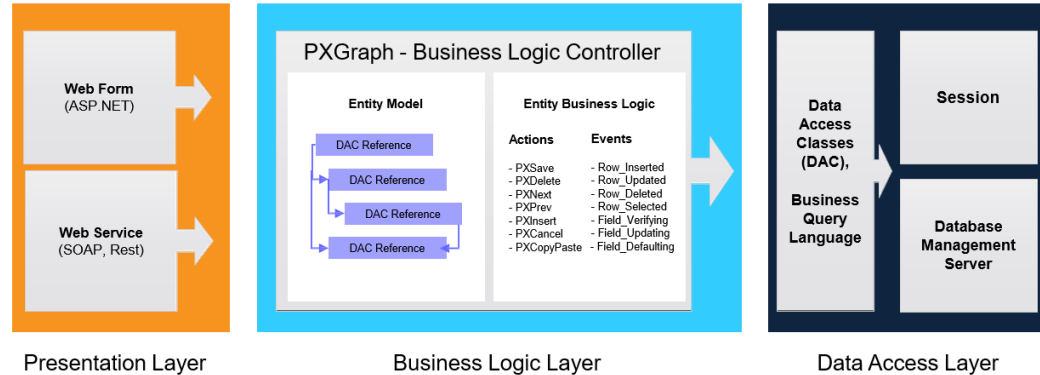
changes in the look and behavior of the user interface

C# files with customization code

changes in the business logic

Introduction – Application Architecture

1. **Data Access Layer** - Set of DACs that wrap data from tables.
2. **Business Logic Layer** – implemented through graphs – tied to one or more DACs. Graphs contain data views (references to the required data access classes, their relationships, and other meta information) and business logic (actions and events associated with the modified data).
3. **Presentation Layer** - provides access to the application business logic through the UI, web services, and Acumatica mobile application. The UI consists of ASPX webpages (which are based on the ASP.NET Web Forms technology) and reports created with Acumatica Report Designer. The ASPX webpages are bound to graphs.



Querying of the Data

- BQL (Business Query Language) – Acumatica's custom language for writing database queries.
- BQL is written in C# and based on generic class syntax and like SQL.
- Benefits of BQL:
 - does not depend on the specifics of the database provider.
 - compile-time syntax validation.
- Provides two dialects of BQL: traditional BQL and fluent BQL (short and simple).

BQL	SQL
<pre>SelectFrom<Product> .Where<Product.availQty.IsNotNull. And <Product.availQty.IsGreater <Product.bookedQty>>></pre>	<pre>SELECT * FROM Product WHERE Product.AvailQty IS NOT NULL AND Product.AvailQty > Product.BookedQty</pre>



Company Story – Smart Fix Company

Onni Group
Acumatica Customer

Company Story - Smart Fix company

The Smart Fix company specializes in repairing cell phones of several types and can both repair cell phones and sell parts for the repair. The company provides the following services:

- Battery replacement
- Repair of liquid damage
- Screen repair

Users can create repair work orders to record repair process and sales orders to record sale of the parts associated with a repair. Employees need to verify the information on both repair work orders and sales orders, including the details of the invoices created for these orders.

The Acumatica ERP instance of the Smart Fix company contains the below custom forms which we will create in other T series of courses:

1. **Repair Services maintenance form (RS201000)** - will be used to view the list of all services, add a new service, edit an existing service, and delete a service.
2. **Serviced Devices maintenance form (RS202000)** - used to view the list of devices that are serviced by the company in a grid.
3. **Services and Prices maintenance form (RS203000)** – used to define and maintain the prices for each repair service.
4. **Repair Work Orders data entry form (RS301000)** – used to create and manage individual work orders for repairs.
5. **Repair Work Order Preferences setup form (RS101000)** – used by an admin user to specify company's preferences for the repair work orders.

In this course, we will customize the **Stock Items form (IN202500)** – to mark a stock item as repair item. And perform the below changes:

- Update of a field value that depends on another field value on the Services and Prices custom maintenance form.
- Validation of a field value on the Repair Work Orders custom data entry form.
- Creation of an SO invoice for a repair work order on the Repair Work Orders form.

Lesson 1: Creating a Customization Project

Learning Objectives:

In this lesson, you will learn how to do the following:

- 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 and 1.2: Creating a Customization Project and Loading Items to the Customization Project

Creation of a Customization Project:

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**.

Loading Items:

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 the *Customization\T190\SourceFiles\PhoneRepairShop* folder, which you have downloaded from Acumatica GitHub in *Initial Configuration*, and click **OK**.

Figure: Items of the customization project

Customization Project Editor [Back](#) [Reload](#)

File Publish Extension Library Source Control

PhoneRepairShop Custom Files

SCREENS

Data Access

Code

Files (23)

Generic Inquiries (3)

Reports

Dashboards

Site Map (6)

Database Scripts (11)

System Locales

Import/Export Scenarios

Shared Filters (1)

Access Rights

Wikis

Web Service Endpoints

Analytical Reports

Push Notifications

Business Events

Mobile Application

User-Defined Fields

Webhooks

Connected Applications

DETECT MODIFIED FILES

Object Name	Third Party Assembly	Description	Last Modified By	Last Modified On
Bin\PhoneRepairShop_Code.dll	<input type="checkbox"/>		admin admin	11/15/2021
InputData\InventoryItem.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVDevice.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVLabor.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVRepairItem.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVRepairPrice.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVRepairService.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVSetup.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVStockItemDevice.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVWarranty.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVWorkOrder.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVWorkOrderItem.csv	<input type="checkbox"/>		admin admin	11/15/2021
InputData\RSSVWorkOrderLabor.csv	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS101000.aspx	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS101000.aspx.cs	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS201000.aspx	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS201000.aspx.cs	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS202000.aspx	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS202000.aspx.cs	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS203000.aspx	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS203000.aspx.cs	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS301000.aspx	<input type="checkbox"/>		admin admin	11/15/2021
Pages\RS\RS301000.aspx.cs	<input type="checkbox"/>		admin admin	11/15/2021

✓ Demo



Extension Library

Cherry Lake Tree Farm
Acumatica Customer

Introduction - Extension Libraries

An extension library is a Visual Studio project that contains customization code and can be individually developed and tested.

An extension library .dll file must be in the Bin folder of the website. At run time during the website initialization, all the .dll files of the folder are loaded into the server memory for use by the Acumatica ERP application.

	Code in DAC and Code items	Code in an Extension Library
Best for:	Quick start of a customization	Development of a customization project when more than one developer is involved
Primary storage:	Database	File system
Location within the website folder:	App_RuntimeCode	Bin
Intellectual property protection:	No—the source code is open in the deployment package	Yes—the source code is not provided in the deployment package
Run-time compilation without the application domain being restarted:	Yes	No
Editor:	Code Editor, Visual Studio	Visual Studio
IntelliSense feature in Visual Studio:	No	Yes
Acuminator extension for Visual Studio:	No	Yes
Debugging:	Yes	Yes
Integration with a version control system:	Yes	Yes
Additional:	Can be moved to an extension library when needed	

More about Extension Libraries

Two ways to maintain the source code of customization:

1. Keep the code in the customization project as *DAC* and *Code* items.
2. Move the code to an *extension library* and include the library in the project as a *File* item.

To decide about how to work with the code, consider the following questions:

- How much code will be in the customization project?
- Is there a need for replicability of the customization?
- How many developers will take part in coding?
- Do you need to open the source code in the production environment?

Factors to decide the necessity of the Extension Library are:

1. More than five class extensions for business logic controllers.
2. Will be deployed on more than one system.
3. Will be developed by a team that needs to use a version control system.
4. To protect the intellectual property of the source code of the solution.
5. Will be a plug-in for Acumatica ERP.

Step 1.3 : Binding the Extension Library

Steps to create an Extension Library:

1. Copy the `Customization\T190\SourceFiles\PhoneRepairShop_Code` folder to the `App_Data\Projects` folder of the Acumatica ERP instance that is prepared for this training course.
2. On the menu of the *Customization Project Editor*, click **Extension Library > Bind to Existing**.
3. In the dialog box that opens, specify the path to the `App_Data\Projects\PhoneRepairShop_Code` folder, and click **OK**.
4. Open the Visual Studio solution and build the `PhoneRepairShop_Code` project.

Files created in Extension Lib	Description
PhoneRepairShop_Code.sln	Microsoft Visual Studio Solution file
Solution.bat	Windows batch file to open the website solution Visual Studio
Solution.lnk	Shortcut file to the project to open the website solution
folder.lnk	Shortcut file to the website folder
PhoneRepairShop_Code\PhoneRepairShop_Code.csproj	Visual C# project file
PhoneRepairShop_Code\Examples.cs	Visual C# source file that contains examples of source code
PhoneRepairShop_Code\Properties\AssemblyInfo.cs	Visual C# Source file that contains general information about an assembly

Step 1.4: Publishing the Customization Project

To publish the project, do the following:

1. Open the *PhoneRepairShop* customization project in the *Customization Project Editor*.
2. Click **Files** on the left pane of the *Customization Project Editor*. The Custom Files page opens.
3. On the page toolbar, click **Detect Modified Files**. Because we have rebuilt the extension library in the `PhoneRepairShop_Code` Visual Studio project, the `Bin\PhoneRepairShop_Code.dll` file has been modified.
4. In the Modified Files Detected dialog box, which opens, make sure the Selected check box is selected for the `Bin\PhoneRepairShop_Code.dll` file, and click **Update Customization Project**.
5. Close the dialog box.
6. On the menu of the *Customization Project Editor*, click **Publish > Publish Current Project**. The Compilation panel opens, which shows the progress of the publication.
7. 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

1. Open the Acumatica ERP instance and notice the changes in the main menu under **Phone Repair Shop** workspace (as shown in the next slide).
2. Open the **Repair Services (RS201000)** form and review its content and functionality.
3. Open any other forms in the **Phone Repair Shop** workspace and review their content and functionality.
4. In Microsoft SQL Server Management Studio, connect to the database of the current Acumatica instance and find the database tables with names starting with *RSSV*. These are the custom tables added during the publishing of the customization project.
5. Open the Acumatica ERP instance folder in the file system. Notice the following files and folders:
 1. **Pages\RS**: Contains the ASPX code of the custom forms. The forms have the *RS* prefix in their IDs. therefore, they are placed in the custom *RS* subfolder.
 2. **InputData**: Contains CSV files with the data for the custom tables. This data is inserted in the database by the InputData customization plug-in, which is included in the customization project.
 3. **CstPublished\pages_RS**: Contains the published code of the custom ASPX pages.
 4. **Bin\PhoneRepairShop_Code.dll**: Contains the customization source code in an extension library.

Figure: The Phone Repair Shop workspace

The screenshot displays the Acumatica software interface for the 'Phone Repair Shop' workspace. The top navigation bar is blue and contains the Acumatica logo, a search bar, a refresh icon, the user 'Yogifon', the date '11/15/2021 6:26 AM', a help icon, and a user profile dropdown for 'admin, admin'. The left sidebar is light gray and lists various modules: Favorites, Data Views, Phone Repair Shop (highlighted with a red rectangle), Time and Expenses, Finance, Banking, Payables, Receivables, Sales Orders, Purchases, Inventory, and More Items. The main content area is white and titled 'Phone Repair Shop'. It features three sub-sections: 'Configuration' with links for 'Repair Services', 'Serviced Devices', and 'Services and Prices'; 'Profiles' with links for 'Stock Items' and 'Repair Work Orders'; and 'Preferences' with a link for 'Repair Work Order Preferences'. A 'TOOLS' dropdown menu is visible on the right side of the main area. The bottom of the interface shows a standard Windows-style taskbar with navigation arrows.

Repair Services

			* Service ID	* Description	Active	Walk-In Service	Requires Prepayment	Requires Preliminary Check
>			BATTERYREPLACE	Battery Replacement	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
			LIQUIDDAMAGE	Liquid Damage	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
			SCREENREPAIR	Screen Repair	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

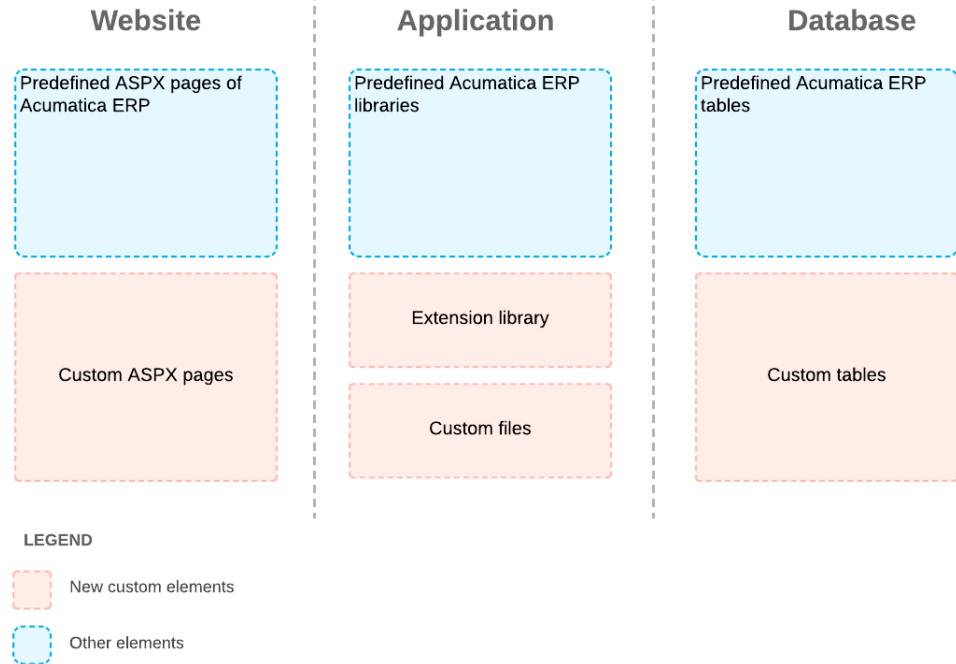
✓ Demo

Lesson Summary

In this lesson, you have learned how to create a customization project, load content to a customization project from a local folder, bind the project to an extension library, and publish the project.

The following diagram shows the changes that have been applied to the Acumatica ERP instance for the training course after the customization project has been published.

New Custom Elements



Lesson 2: Creating Custom Fields

Learning Objectives

In this lesson, you will learn how to do the following:

- 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

Purpose

The manager of the Smart Fix Company needs to specify some stock items on **Stock Items (IN202500)** form as repair item and select the corresponding type. It can be achieved by changing below:

- The Database table - create a custom database column using Customization Project Editor.
- The DAC - add a new field to accommodate the database column using Visual Studio.
- The User Interface – create a control in the screen or ASPX from the DAC field using Customization Project Editor.

Changes to be implemented to **Stock Items (IN202500)** form to the **Item Defaults** section of **General** tab:

- The **Repair Item** check box will be used to define whether the selected stock item is a repair item.
- The **Repair Item Type** box will hold the repair item type to which the repair item belongs
 - *Battery, Screen, Screen Cover, Back Cover, or Motherboard.*
- Custom Fields are added to *IN.InventoryItem* DAC and *InventoryItem* database table.

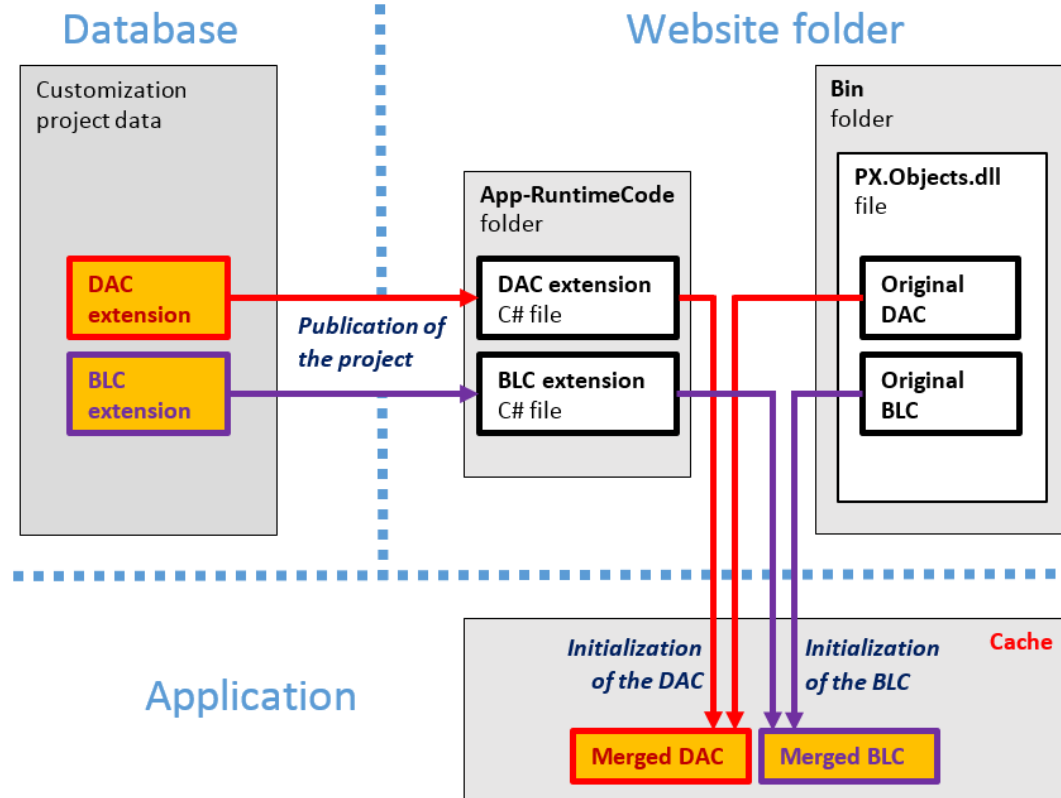


Acumatica Customization Platform

– An Overview

Acumatica Customization Platform – An Overview

- Provides the ability to customize the functionality or behavior of the form.
- Based on **extension models**.
- An extension for a graph (BLC) or a DAC is a class derived from a generic class defined in **PX.Data** assembly of Acumatica.
 - DAC Extension is derived from `PXCacheExtension<T>` generic class.
 - BLC/Graph Extension is derived from `PXGraphExtension<T>` generic class.
- The graph/cache extensions present/published in a customization project are applied to the base class at run time during **the first initialization** of the base class.
- Supports **multi-level extensions** – to develop applications distributed in multiple editions. During run-time, the system collects list of all the extensions and load in alphabetical order.



```
> PXCacheExtension<Table>
> PXCacheExtension<Extension1,Table>
> PXCacheExtension<Extension2,Extension1,Table>
> PXCacheExtension<Extension3,Extension2,Extension1,Table>
> PXCacheExtension<Extension4,Extension3,Extension2,Extension1,Table>
> PXCacheExtension<Extension5,Extension4,Extension3,Extension2,Extension1,Table>
> PXCacheExtension<Extension6,Extension5,Extension4,Extension3,Extension2,Extension1,Table>
> PXCacheExtension<Extension7,Extension6,Extension5,Extension4,Extension3,Extension2,Extension1,Table>
> PXCacheExtension<Extension8,Extension7,Extension6,Extension5,Extension4,Extension3,Extension2,Extension1,Table>
> PXCacheExtensionAttribute
```

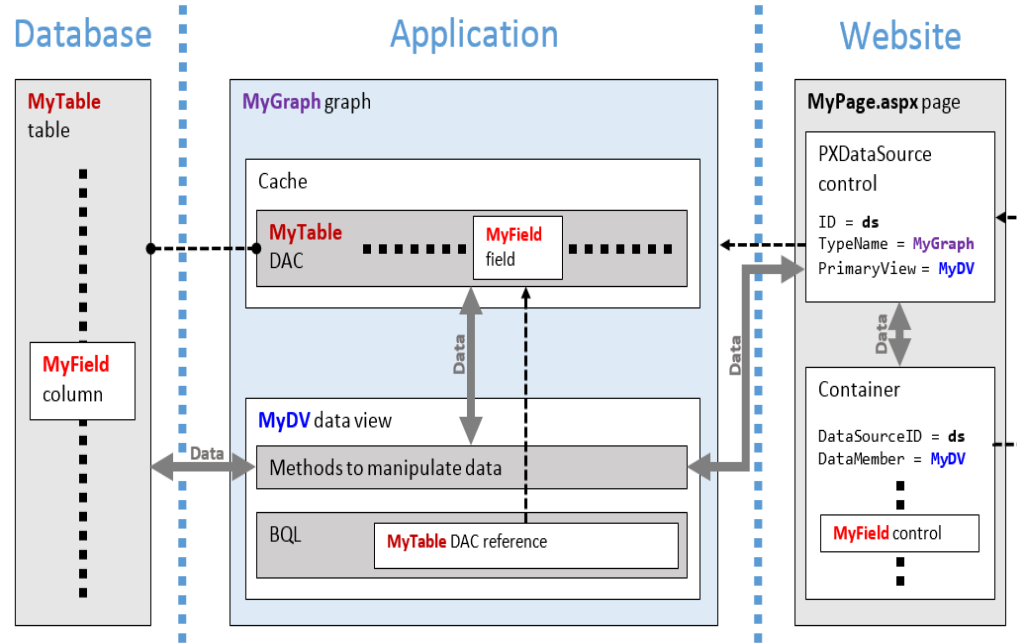
Acumatica Customization Platform – An Overview (Contd..)

```
public class DACExtension : PXCacheExtension<BaseDAC>
{
    //Put new fields definition here

    //Customize existing attributes and fields
}

public class BLCEExtension : PXGraphExtension<BaseBLC>
{
    //Put new event handlers, actions, data views or
    //methods here

    //Customize existing logic with defining new one
    //with the same name
}
```



Step 2.1: Creating a custom column and field with the Project Editor

In this step, we will create a custom column for **Repair Item** checkbox to *InventoryItem* database table and a custom field to *IN.InventoryItem* Data Access Class.

- Create a DAC extension or cache extension of *IN.InventoryItem* DAC to hold the custom fields.
- Open the Stock Items (IN202500) form, and then open the **Screen Editor** for it as follows:
 1. On the form title bar, click **Customization > Inspect Element** – to find the details about the tab and section
 2. Click the name of the **General** tab to open the **Element Properties** dialog box (provides details about the Control Type, Data access Class, View Name and the BLC or Graph of the control and the form).
 3. Click **Customize**.
 4. In the **Select Customization Project** dialog box, which opens, select the *PhoneRepairShop* customization project, and click **OK**. The Customization Project Editor opens for the *PhoneRepairShop* project; the Screen Editor is displayed for the **Tab: ItemSettings** node, which is selected in the control tree.
- To add a custom field for the **Repair Item** check box in the customization project
 1. On the Screen Editor page, click the **Add Data Fields** tab.
 2. On the table toolbar, click **New Field**.
 3. In the **Create New Field** dialog box, which opens, specify the following settings for the new field:
 1. Field Name: `RepairItem`
 2. Display Name: `Repair Item`
 3. Storage Type: `DBTableColumn`
 4. Data Type: `bool`
 4. Click **OK** to create the extensions to both DAC and the database table with Ext as suffix to the *IN.InventoryItem* DAC.

Figure: Custom elements to be added to the Stock Items form

Stock Items

New Record

NOTES ACTIVITIES FILES CUSTOMIZATION TOOLS

← ↻ 📁 ↶ + 🗑️ 📄 ↩️ < > >| ...

* Inventory ID: Product Workgroup:
Item Status: **Active** Product Manager:
Description:

GENERAL PRICE/COST WAREHOUSES VENDORS ATTRIBUTES PACKAGING CROSS-REFERENCE GLACCOUNTS >>

ITEM DEFAULTS

* Item Class:
Type: **Finished Good**

☐ Repair Item

Repair Item Type:

Valuation Method: **Standard**
* Tax Category:
* Posting Class:
Auto-Incremental Value:
Country Of Origin:

UNIT OF MEASURE

* Base Unit: ☒ Divisible Unit
* Sales Unit: ☒ Divisible Unit
* Purchase Unit: ☒ Divisible Unit
☐ Weight Item

↻ + ×

* From Unit	Multiply/Divide	Conversion Factor	To Unit

WAREHOUSE DEFAULTS

Default Warehouse:

Figure: Customization menu

Stock Items

New Record

NOTES ACTIVITIES FILES **CUSTOMIZATION** TOOLS ▾

Select Project...
Inspect Element (Ctrl+Alt+Click)
Edit Project...
Manage Customizations...

Inventory ID: Product Workgroup:
Item Status: **Active** Product Manager:
Description:

GENERAL PRICE/COST WAREHOUSES VENDORS ATTRIBUTES PACKAGING CROSS-REFERENCE GLACCOUNTS DESCRIPTION

ITEM DEFAULTS

* Item Class:
Type: **Finished Good**
Valuation Method: **Standard**
* Tax Category:
* Posting Class:
Auto-Incremental Value:
Country Of Origin:

UNIT OF MEASURE

* Base Unit: ☒ Divisible Unit
* Sales Unit: ☒ Divisible Unit
* Purchase Unit: ☒ Divisible Unit
☐ Weight Item

⌂ + ×

* From Unit	Multiply/Divid	Conversion Factor	To Unit

WAREHOUSE DEFAULTS

Default Warehouse:

Figure: Element Properties dialog box

Stock Items

New Record

NOTES ACTIVITIES FILES CUSTOMIZATION TOOLS

Inventory ID: Product Workgroup: Item Status: Active Product Manager: Description:

GENERAL PRICE/COST WAREHOUSES VENDORS ATTRIBUTES PACKAGING CROSS REFERENCE GL ACCOUNTS DESCRIPTION

ITEM DEFAULTS

Item Class: Type: Finished Good Valuation Method: Standard Tax Category: Posting Class: Auto-Incremental Value: Country Of Origin: Unit Factor

WAREHOUSE DEFAULTS

Default Warehouse:

Element Properties

Control Type: Tab

Data Class: [InventoryItem](#)

View Name: ItemSettings

Business Logic: InventoryItemMaint

CUSTOMIZE ACTIONS CANCEL

Step 2.1: Creating a custom column and field with the Project Editor – Continued..

- Move the data access class extension to the PhoneRepairShop_Code extension library:
 1. In the navigation pane, click **Data Access**.
 2. On the Customized Data Classes page, click the line with *InventoryItem*.
 3. On the page toolbar, click **Convert to Extension**.
 4. The *InventoryItemExtensions* Code item appears in the Code Editor.
 5. On the toolbar of the Code Editor, click **Move to Extension Lib**.
- In Visual Studio, adjust the DAC extension as follows:
 1. Move the `InventoryItemExtensions.cs` file to the DAC folder and open the file. Notice that Acuminator displays the PX1016 error and the PX1011 warning for the `InventoryItemExt` class. We can either suppress or fix the Acuminator errors/warnings.
 2. Remove `virtual` from the `UsrRepairItem` property field.
 3. Make sure the `UsrRepairItem` field has the attributes shown in the following code.

```
[PXDBBool]

[PXUIField(DisplayName="Repair Item")]

[PXDefault(false, PersistingCheck = PXPersistingCheck.Nothing)]
```
 4. Build the project.

Figure: Suppression of the error in a comment

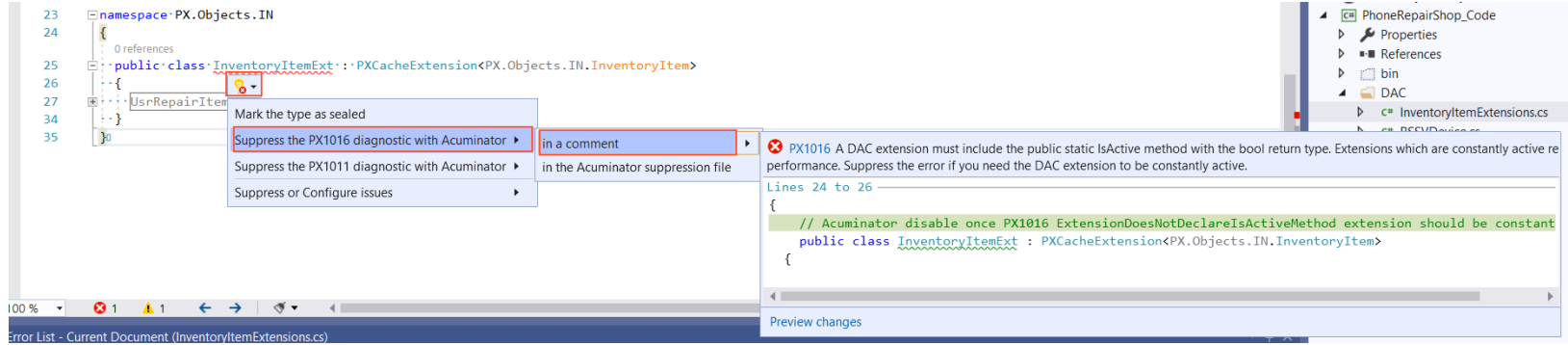
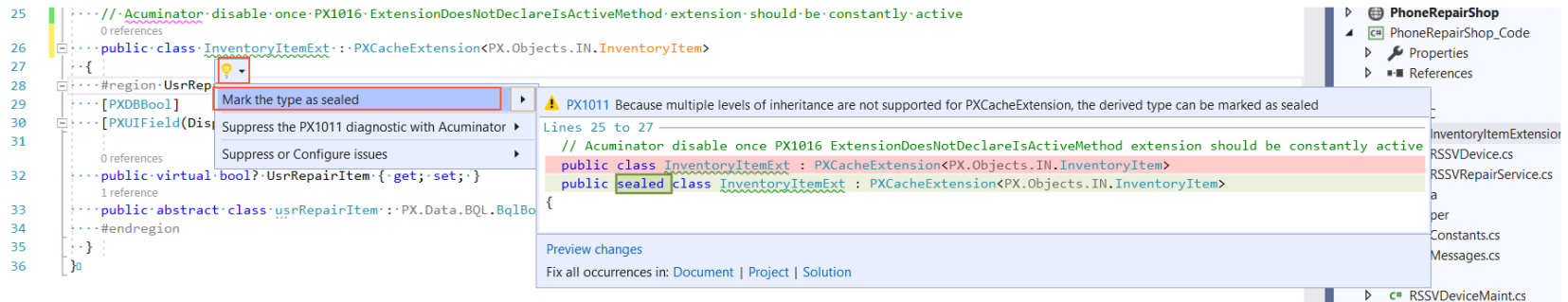


Figure: Fix of the warning



Step 2.2: Creating a Control for the Custom Field

- Open the **Screen Editor** for the **Stock Items (IN202500)** form.
- In the control tree of the **Screen Editor**, click the **Tab: ItemSettings** node.
- On the **Add Data Fields** tab, select the **Custom** filter tab to view the custom fields that are available through the data view of the container. Notice that the Control column displays the available control type for the custom field.
- Create the control for the custom field as follows:
 - In the control tree of the **Screen Editor**, select the **Type** node to position the new control beneath it.
 - On the **Add Data Fields** tab, select the unlabeled check box for the row with the custom field.
 - On the table toolbar, click **Create Controls** to create the control for the selected field.
 - On the menu of the Customization Project Editor, click **Publish > Publish Current Project** to apply the customization to the site.
 - Close the **Compilation** window.
 - Refresh the **Stock Items** form in the browser to view the added control on the **General** tab of the form.

Figure: The Type node in the control tree

Screen Editor: IN202500 (Stock Items)

EDIT ASPX PREVIEW CHANGES ...

Layout Properties ATTRIBUTES EVENTS ADD CONTROLS **ADD DATA FIELDS** VIEW ASPX

Data View: Inventory Item(ItemSettings)

CREATE CONTROLS NEW FIELD ALL VISIBLE **CUSTOM**

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

Control Tree:

- DataSource: InventoryItemMaint
 - Form: Item
 - Tab: ItemSettings
 - General
 - Column
 - Template ID
 - Group
 - Item Class
 - Type**
 - Is a Kit
 - Valuation Method
 - Tax Category
 - Posting Class
 - Lot/Serial Class
 - Auto-Incremental Value
 - Country Of Origin
 - Form: CurySettings_InventoryItem
 - Merge
 - [Layout Rule]
 - Column
 - Subitems
 - Price/Cost
 - Manufacturing
 - Warehouses

Figure: The added control

Screen Editor: IN202500 (Stock Items)

EDIT ASPX PREVIEW CHANGES ...

Layout Properties ATTRIBUTES EVENTS ADD CONTROLS **ADD DATA FIELDS** VIEW ASPX

Data View: Inventory Item(ItemSettings)

CREATE CONTROLS NEW FIELD ALL VISIBLE **CUSTOM**

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

DataSource: InventoryItemMaint

- Form: Item
 - Tab: ItemSettings
 - General
 - Column
 - Template ID
 - Group
 - Item Class
 - Type**
 - Repair Item
 - Is a Kit
 - Valuation Method
 - Tax Category
 - Posting Class
 - Lot/Serial Class
 - Auto-Incremental Value
 - Country Of Origin
 - Form: CurySettings_InventoryItem
 - Merge
 - [Layout Rule]
 - Column
 - Subitems
 - Price/Cost
 - Manufacturing

Figure: The Repair Item check box

Stock Items

New Record

NOTES ACTIVITIES FILES CUSTOMIZATION TOOLS ▾

← ↺ ↻ + 🗑️ 📄 ▾ ⏪ ⏩ ⏴ ⏵ ⋮

* Inventory ID: Product Workgroup:
Item Status: **Active** Product Manager:
Description:

GENERAL PRICE/COST WAREHOUSES VENDORS ATTRIBUTES PACKAGING CROSS-REFERENCE >>

ITEM DEFAULTS

* Item Class:
Type: **Finished Good**
☐ **Repair Item**
Valuation Method: **Standard**
* Tax Category:
* Posting Class:
Auto-Incremental Value:
Country Of Origin:

UNIT OF MEASURE

* Base Unit: ☒ Divisible Unit
* Sales Unit: ☒ Divisible Unit
* Purchase Unit: ☒ Divisible Unit
☐ Weight Item

🔄 + ×

* From Unit	Multiply/Divide	Conversion Factor	To Unit

WAREHOUSE DEFAULTS

Default Warehouse:

✓ Demo

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

Will define the `UsrRepairItemType` data field in the `InventoryItemExt` DAC extension and **Repair Item Type** combo box as input control.

To add the column to the *InventoryItem* table, in the Customization Project Editor, open the `PhoneRepairShop` project -> **Database Scripts** -> **Add Column to Table**. In the dialog box that opens, specify the following values and click **OK**:

- Table: `InventoryItem`
- Field Name: `UsrRepairItemType`
- Data Type: `string`
- Length: 2

In Visual Studio, in the `Helper\Constants.cs` file, add the `RepairItemTypeConstants` class – to define the constants for repair item types.

```
public const string Battery = "BT";
```

In the `Helper\Messages.cs` file, define the strings for the repair item types in the `Messages` class.

```
public const string Battery = "Battery";
```

Add the field `UsrRepairItemType` – (`PXDBString`) to the `InventoryItemExt` DAC – with `PXStringList` attribute utilizing the above constant strings.

Build the project.

Step 2.4: Creating a control for the Custom Field

- Create a control for **Repair Item Type** custom field from the `UsrRepairItemType` field added to the `InventoryItemExt` DAC.
- Can be done either from the **Screen Editor** or from the **ASPX** of the page.
- For custom forms, ASPX can be found inside `Pages` folder of the instance. For customized version of existing pages, ASPX can be found inside `CstPublished` folder of the instance. Files in `CstPublished` folder are available for preview in the UI and are overridden once the package is published.
- Make sure the type of the control is Combo box or Dropdown.

Figure: The Repair Item Type box

Stock Items

New Record

NOTES ACTIVITIES FILES CUSTOMIZATION TOOLS ▾

← ↺ ↻ + 🗑️ 📄 ▾ ⏪ < > ⏩ ⋮

* Inventory ID: Product Workgroup:
Item Status: **Active** Product Manager:
Description:

GENERAL PRICE/COST WAREHOUSES VENDORS ATTRIBUTES PACKAGING CROSS-REFERENCE GLACCOUNTS >>

ITEM DEFAULTS

* Item Class:
Type: **Finished Good**
☐ Repair Item
Repair Item Type:

Valuation Method: **Standard**
* Tax Category:
* Posting Class:
Auto-Incremental Value:
Country Of Origin:

UNIT OF MEASURE

* Base Unit: ☒ Divisible Unit
* Sales Unit: ☒ Divisible Unit
* Purchase Unit: ☒ Divisible Unit
☐ Weight Item

🔄 + ×

* From Unit	Multiply/Divid	Conversion Factor	To Unit

WAREHOUSE DEFAULTS

Default Warehouse:

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

The **Repair Item Type** box should be unavailable on the **Stock Items (IN202500)** form unless the **Repair Item** check box is selected.

Changes to the DAC:

Repair Item Type box is made unavailable by default by setting the `Enabled` property of the `PXUIField` attribute of `UsrRepairItemType` in `InventoryItemExt` DAC to `false`.

Changes to the graph:

- Add the `RowSelected` event handler to make the **Repair Item Type** field available based on the selection of **Repair Item** checkbox.
- Access `UsrRepairItem` by invoking `GetExtension` method to `InventoryItem` DAC and use `PXUIFieldAttribute.SetEnabled<>()` to change the `Enabled` property of `UsrRepairItemType` extension field.

An event handler can be added either from **Screen Editor** and navigating to the corresponding control -> **Events** tab -> **Add Handler -> Keep Base Method** or can be added from Visual Studio though code in `InventoryItemMaint.cs` file.

Changes to the ASPX page:

Set `CommitChanges` to `True` for **Repair Item** checkbox.

If the value in a box needs to be processed every time the user changes this value, we need to set the `CommitChanges` property of the box to `True` to enable callbacks for the box.

Figure: The generation of the event handler

Screen Editor: IN202500 (Stock Items)

EDIT ASPX PREVIEW CHANGES ...

DataSource: InventoryItemMaint
Form: Item
Tab: ItemSettings
General
Column
Template ID
Group
Item Class
Type
Repair Item
Repair Item Type
Is a Kit
Valuation Method
Tax Category
Posting Class
Lot/Serial Class
Auto-Incremental Value
Country Of Origin
Form: CurySettings_InventoryItem
Merge
[Layout Rule]
Column
Subitems
Price/Cost

LAYOUT PROPERTIES ATTRIBUTES **EVENTS** ADD CONTROLS ADD DATA FIELDS VIEW ASPX

Data Class: PX.Objects.IN.InventoryItem
Field Name: UsrRepairItem
Business Logic: PX.Objects.IN.InventoryItemMaint:ItemSettings

ADD HANDLER VIEW SOURCE

Keep Base Method

Event	Handled in Source	Customized
Com	<input type="checkbox"/>	<input type="checkbox"/>
RowSelecting	<input type="checkbox"/>	<input type="checkbox"/>
RowSelected	<input type="checkbox"/>	<input type="checkbox"/>
FieldSelecting	<input type="checkbox"/>	<input type="checkbox"/>
RowInserting	<input type="checkbox"/>	<input type="checkbox"/>
RowInserted	<input type="checkbox"/>	<input type="checkbox"/>
RowUpdating	<input type="checkbox"/>	<input type="checkbox"/>
RowUpdated	<input checked="" type="checkbox"/>	<input type="checkbox"/>
RowDeleting	<input type="checkbox"/>	<input type="checkbox"/>
RowDeleted	<input type="checkbox"/>	<input type="checkbox"/>
FieldDefaulting	<input type="checkbox"/>	<input type="checkbox"/>
FieldUpdating	<input type="checkbox"/>	<input type="checkbox"/>
FieldVerifying	<input type="checkbox"/>	<input type="checkbox"/>
ExceptionHandling	<input type="checkbox"/>	<input type="checkbox"/>

Figure: The CommitChanges property

Screen Editor: IN202500 (Stock Items)

EDIT ASPX PREVIEW CHANGES ...

Screen Editor: IN202500 (Stock Items)

EDIT ASPX PREVIEW CHANGES ...

DataSource: InventoryItemMaint

Form: Item

Tab: ItemSettings

General

Column

Template ID

Group

Item Class

Type

Repair Item

Repair Item Type

Is a Kit

Valuation Method

Tax Category

Posting Class

Lot/Serial Class

Auto-Incremental Value

Country Of Origin

Form: CurySettings_InventoryItem

Merge

[Layout Rule]

Column

Subitems

Price/Cost

LAYOUT PROPERTIES ATTRIBUTES EVENTS ADD CONTROLS ADD DATA FIELDS VIEW ASPX

Override	Property	Value
Base Properties		
<input type="checkbox"/>	CommitChanges	True
<input checked="" type="checkbox"/>	DataField	UsrRepairItem
<input checked="" type="checkbox"/>	ID	CstPXCheckBox1
<input type="checkbox"/>	Size	
<input type="checkbox"/>	Text	
Ext Properties		
<input type="checkbox"/>	AlignLeft	
<input type="checkbox"/>	AlreadyLocalized	
<input type="checkbox"/>	AutoCallBack	
<input type="checkbox"/>	CheckImages	
<input type="checkbox"/>	Enabled	
<input type="checkbox"/>	FalseValue	
<input type="checkbox"/>	LabelWidth	
<input type="checkbox"/>	RenderStyle	

Indicates whether the control performs commit callback after the value of the control has been changed.

✓ Demo

More about Event Handlers and RowSelected event

An event handler can be implemented in a graph, as well as in an attribute of a data field.

- *Graph event handlers* - defined as methods in a BLC class for a particular DAC or a DAC field.
- *Attribute event handlers* - defined as methods in attribute classes. And is attached to all DAC objects or data fields with these attributes.

Types of Event Handlers

Classic (Obsolete):

```
public virtual void DAC_Field_Updated(PXCache cache,
PXFieldUpdatedEventArgs e)
{
}

public virtual void DAC_RowInserting(PXCache cache,
PXRowInsertingEventArgs e)
{
}
```

Generic (Recommended):

```
public virtual void _(Events.FieldUpdated<DAC, DAC.field> e)
{
}

public virtual void _(Events.RowInserting<DAC> e)
{
}
```

-ing events (e.g. `FieldDefaulting`, `RowUpdating`):

- *Graph* handlers first, then *attribute* handlers
- Graph handlers can cancel execution of attribute handlers by setting `e.Cancel = true`

-ed events (e.g. `FieldUpdated`, `RowUpdated`):

- *Attribute* handlers first, then *graph* handlers
- One typically can't cancel anything in graph

RowSelected Event:

- `RowSelected` occurs each time a data record is displayed in the UI.
- When one sets the `Current` property of a `PXCache` object.
- The *best* place to configure the UI based on the values of data fields.
- However, note that `RowSelected` is fired several times for a record during each round trip and it is the *worst* place to read data from the database.

Step 2.6: Testing the Customized Form

UI Element (Location)	First Modified Record	Second Modified Record	Third Modified Record
Inventory ID (Summary area)	<i>BAT3310</i>	<i>BAT3310EX</i>	<i>BCOV3310</i>
Repair Item (Item Defaults section of the General tab)	Selected	Selected	Selected
Repair Item Type (Item Defaults section of the General tab)	<i>Battery</i>	<i>Battery</i>	<i>Back Cover</i>

Lesson Summary

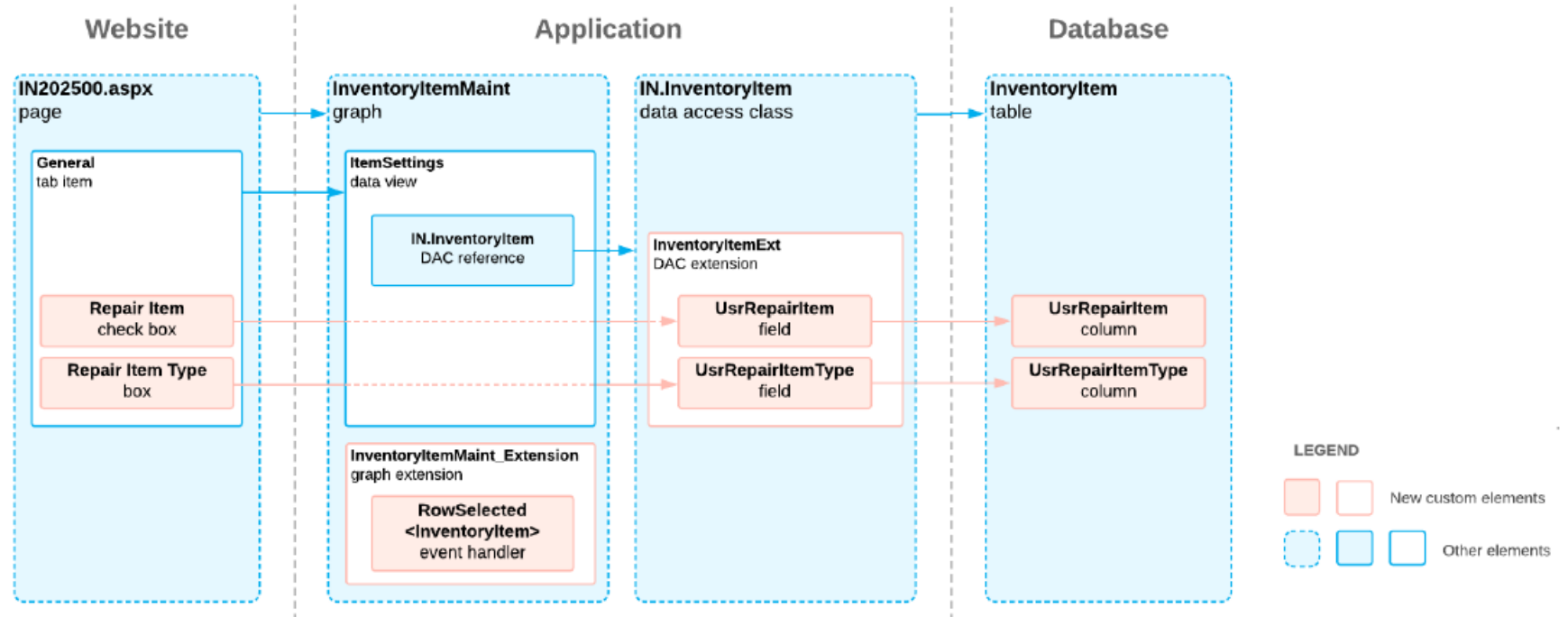
In this lesson, you have learned how to create a control so that you can display on a form a custom field bound to the database. To implement this customization, you have learned how to add the necessary modifications to a customization project and how to publish the project to apply the changes to the system.

As you have completed the lesson, you have added the following elements to the PhoneRepairShop customization project:

- Two column definitions in the InventoryItem table of the database.
- Two custom field declarations in the extension of the IN.InventoryItem data access class (in the PhoneRepairShop_Code extension library).
- Two controls to display the custom fields on the Stock Items (IN202500) form.
- One custom event handler, which you have added to the InventoryItemMaint graph. You have used the RowSelected event handler to configure the UI presentation logic.

Lesson Summary

Addition of New Custom Elements





Day 2

Joe Gibbs Racing
Acumatica Partner

Recap of Day 1

1. Learnt about Customization projects, Extension Library, Acumatica Customization Platform and Application Architecture.
2. Created a new Customization Project, Loaded items from a folder, Bound it to an existing extension library, Published the Customization project to apply the changes to the instance.
3. **Inspect Element** properties for **Stock Items** screen and customized *InventoryItem* DAC and added `UsrRepairItem` field and added the checkbox control in Stock Items form using **Screen Editor**.
4. Added another field `UsrRepairItemType` to *InventoryItemExt*, a custom column to *InventoryItem* table through Database Scripts and created a combo box control in **Stock Items** form using Screen Editor.
5. Learnt about Event Handlers, Acuminator warnings and errors.
6. Added `RowSelected` event from Screen Editor > Events. And moved the *InventoryItemMaint_Extension* to the Extension library.
7. Added `Enabled` property to false in `PXUIField` attribute of `UsrRepairItemType` field.
8. Modified the classic Handler to Generic Handler type. And added the condition to enable the `UsrRepairItemType` only if the `UsrRepairItem` value is true.

Lesson 3: Implementing the Update and Validation of Field Values

Learning Objectives

In this lesson, you will learn how to do the following:

- 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

Changes to be Implemented

- Modify the Business logic of **Services and Prices (RS203000)** form and **Repair Work Orders (RS301000)** forms.
- In **Repair** tab of the **Services and Prices (RS203000)** form – if an *Inventory ID* is selected, the values in *Repair Item Type* and *Price* must be filled from *Repair Item Type* and *Base Price* from **Stock Items (IN202500)** form.
- In **Labor** tab of the **Repair Work Orders (RS301000)** form – the values are automatically filled from **Services and Prices (RS203000)** form and the *Quantity* column must be validated to fulfill the below conditions:
 - Must be greater than or equal to zero.
 - Must be greater than or equal to the Quantity column in the Labor tab of Services and Prices form with same Inventory ID, Service ID and Device ID.
 - Show error when lesser than zero.
 - Display warning and automatically change the value when lesser than the value in Labor tab of the Services and Prices form.

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

- In **Services and Prices (RS203000)** form, when the `RSSVRepairItem.InventoryID` value is changed, will copy the `RSSVRepairItem.BasePrice` and `RSSVRepairItem.RepairItemType` values from the stock item record that has the ID equal to the new `RSSVRepairItem.InventoryID` value.
- In `RSSVRepairPriceMaint` graph, we will add
 - `FieldUpdated` event handler for `RSSVRepairItem.InventoryID` field to update:
 - `RSSVRepairItem.RepairItemType` – by calling `SetValueExt<field>` to assign value.
 - `RSSVRepairItem.BasePrice` – trigger `FieldDefaulting` event by calling `SetDefaultExt<field>` and assign value.
 - `FieldDefaulting` event handler for `RSSVRepairItem.BasePrice` to set the base price
 - `RSSVRepairItem.BasePrice` – find the `InventoryItem` and corresponding `InventoryItemCurySettings` to get the base price of an inventory item and set to `NewValue`.
- `PXSelectorAttribute.Select<>()` method – to select a stock item with `InventoryID` from the updated field using the BQL query from `PXSelector` on the specific field.
- `PK.Find()` method – used to select a record based on the values of the key fields and can be used when a primary key is defined for a DAC.
- Enable callback for the control in the `RS203000.aspx` by setting `CommitChanges` property to `True` for the `InventoryID` control of **Repair Items** tab.

Introduction to BQL

- Is a part of Acumatica Data Access Layer
- Is mapped to SQL queries
- Hides the underlying database engine
- Is checked at compile time
- Comes with a variety of clauses allowing to express most DB queries

Some Common BQL Clauses

- | | |
|--------------------|--|
| • Where<> | • SelectFrom<>.View |
| • InnerJoin<>.On<> | • SelectFrom<>.OrderBy<>.View |
| • OrderBy<> | • SelectFrom<>.[Joins].View |
| • GroupBy<> | • SelectFrom<>.[Joins].AggregateTo<>.View.ReadOnly |
| | • SelectFrom<>.Where<>.OrderBy<>.View.ReadOnly |

BQL – Querying Data

Passing Parameters from the code – Required:

```
foreach(Product record in SelectFrom<Product>
    Where<Product.isActive.IsEqual<@P.AsBool>>
    .View.Select(this, true));
```

Parameter value from context - Current:

```
Product record = SelectFrom<Product>.
    Where<Product.productid
    .IsEqual<Tran.productid.FromCurrent>>
    .View.Select(this);
```

Optional value in a query:

```
public SelectFrom<Document>.
    Where<Document.docType.IsEqual<Document.doctype
    .AsOptional>>.View Receipts;
```

Comparison of Fluent BQL, Traditional BQL, and LINQ

Characteristic	Fluent BQL	Traditional BQL	LINQ
The queries can be used to define data views in graphs.	Yes	Yes	No
The queries can be defined in code.	Yes	Yes	Yes
The queries can be defined in DAC field attributes.	Yes	Yes	No
DACs are used to define database tables in the queries.	Yes	Yes	Yes
The queries can be used for dynamic query building.	Yes	Yes	Yes
The queries can be parsed and modified by the direct use of reflection—that is, by <code>Type.GetGenericArguments()</code> .	No	Yes	No
Clauses (such as <code>Join</code> , <code>Where</code> , <code>Aggregate</code> , <code>OrderBy</code> , and <code>On</code>) can be used separately of the query.	No, but you can pass fluent BQL expressions to traditional BQL clauses	Yes	No
The query language includes numbered classes (such as <code>Select2</code> and <code>Select6</code>).	No	Yes	No
Each subsequent element of the query is passed as a generic parameter of the previous one.	No	Yes	No
To create a query, a developer needs to select a suitable command overload.	No	Yes	No
IntelliSense can offer continuations that are relevant for the current query state.	Yes	No	Yes
The queries use strongly typed expressions, which makes compile-time type checks possible.	Yes	No	Yes
The queries can contain explicit brackets in conditions.	Yes	No; the <code>Where</code> clause can be used instead	Yes
You can specify particular columns of the tables to be selected.	Yes; you have to use <code>PXFieldScope</code>	Yes; you have to use <code>PXFieldScope</code>	Yes
The query is not executed until it is iterated over.	Yes	Yes	Yes

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

In **Repair Work Orders (RS301000)** form, in the associated graph - `RSSVWorkOrderEntry`, `FieldVerifying` event is added to validate the *Quantity* as below:

- When `RSSVWorkOrderLabor.Quantity < 0`, throw an exception using `PXSetPropertyException` and cancel the assignment of the new value
- When `RSSVWorkOrderLabor.Quantity > 0` but smaller than `RSSVLabor.Quantity` value (set from **Services and Prices** form), attach the exception as warning to `RSSVWorkOrderLabor.Quantity` field using `RaiseExceptionHandling<>` which will prevent the saving of the record.
- Retrieve the default labor item related to the work order labor using Fluent BQL statement – `SelectFrom<RSSVLabor>` using `RSSVWorkOrder.ServiceID`, `RSSVWorkOrder.DeviceID` and `RSSVWorkOrderLabor.InventoryID`.

In the Screen Editor or ASPX code, set `CommitChanges` is set to `True` for the `Quantity` field in the grid of Labor tab of **Repair Work Orders (RS301000)** form.

Figure: The error for a negative value

Repair Work Orders

000001 - Battery Replacement

NOTES FILES CUSTOMIZATION TOOLS

← ↻ 📄 ↶ + 🗑️ 📄 < > >|

* Order Nbr.:	000001	* Customer ID:	C000000001 - Jersey Central Office E	Order Total:	35.00
Status:	On Hold	* Service:	BATTERYREPLACE - Battery Replace	Invoice Nbr.:	
* Date Created:	3/3/2022	* Device:	NOKIA3310 - Nokia 3310		
Date Completed:		Assignee:			
Priority:	Medium	Description:			

REPAIR ITEMS **LABOR**

↻ + × ⇄ ☒

📄	🔍	📄	Inventory ID	Description	Default Price	Quantity	Ext. Price
↶	🔍	📄	CONSULT	Consulting service	5.00	-1.00	5.00

The value in the Quantity column cannot be negative.

< < > >|

Figure: The warning message

Repair Work Orders

000001 - Battery Replacement

NOTES FILES CUSTOMIZATION TOOLS ▾

← ↺ ⏏ ↶ + ⏏ ↷ |< < > >|

* Order Nbr.: 000001 * Customer ID: C000000001 - Jersey Central Office E Order Total: 35.00

Status: On Hold * Service: BATTERYREPLACE - Battery Replace Invoice Nbr.:

* Date Created: 3/3/2022 * Device: NOKIA3310 - Nokia 3310

Date Completed: Assignee:

Priority: Medium Description:

REPAIR ITEMS **LABOR**

↺ + × |<| ⏏

Inventory ID	Description	Default Price	Quantity	Ext. Price
CONSULT	Consulting service	5.00	1.00	5.00

The value in the Quantity column has been corrected to the minimum possible value.

|< < > >|

✓ Demo

Lesson Summary

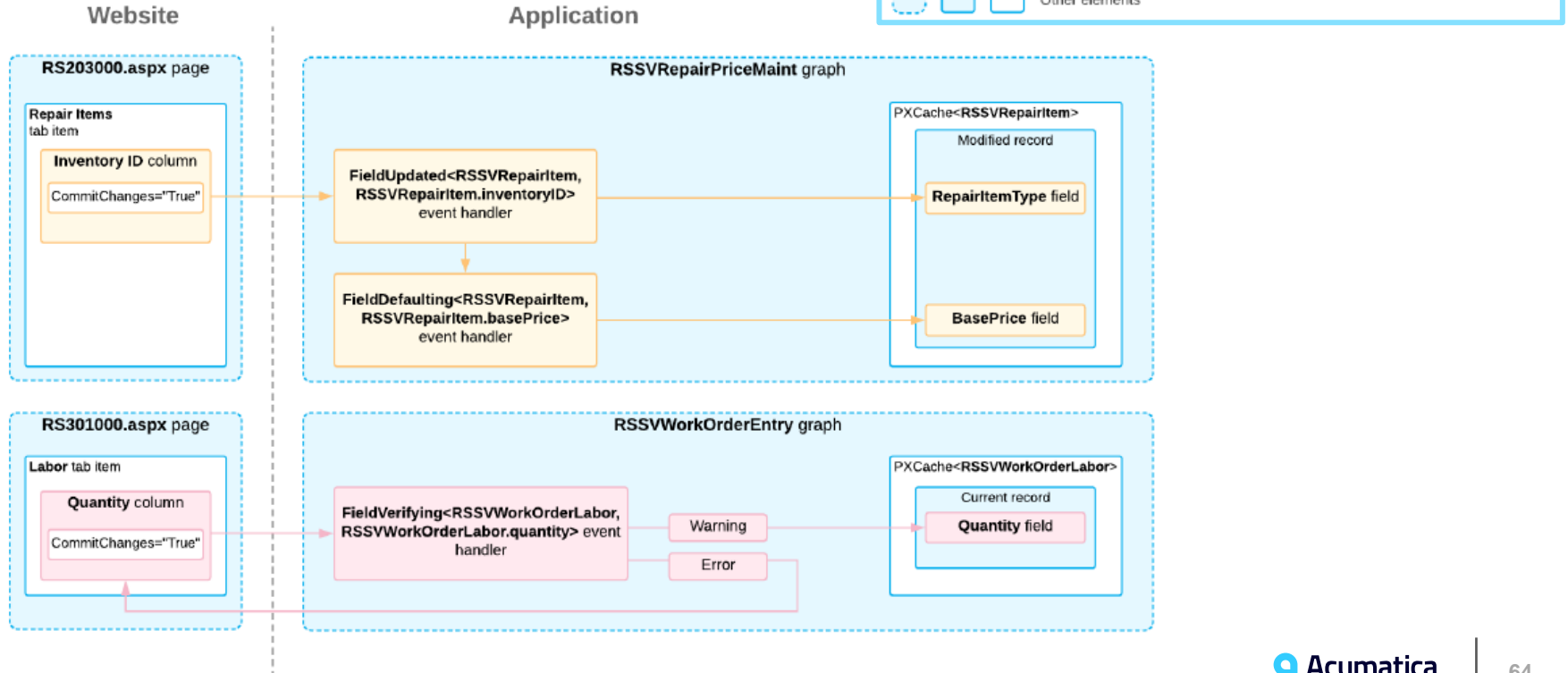
In this lesson, you have defined the business logic scenarios on the Repair Items tab of the Services and Prices (RS203000) form and on the Labor tab of the Repair Work Orders (RS301000) form.

You have used the FieldUpdated and FieldDefaulting event handlers to modify the values of a detail record on update of the Inventory ID column of this detail record. In the FieldUpdated event handler, you have used the PXSelectorAttribute.Select<>() method to obtain the stock item record with the inventory ID selected in the updated field.

To verify the value of a field that does not depend on other fields of the same record, you have used the FieldVerifying event handler. In this event handler, you have thrown an exception by using PXSetPropertyException to display an error and cancel the assignment of the new value. To display a warning, you have attached the exception to the field by using the RaiseExceptionHandling method.

Lesson Summary

Implementation of the Update and Verification of a Field Value



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

Learning Objectives

In this lesson, you will learn how to implement an asynchronous operation by using the `PXLongOperation` class.

Step 4.1: Performing Preliminary Steps

1. In **Enable/Disable Features (CS10000)** form, enable/check *the Advanced SO Invoices* feature – to enable the addition of a stock item directly to an SO Invoice without processing sales orders and shipments.
2. On the **Item Classes (IN201000)** form, in the **Item Class Tree**, select *STOCKITEM*.
3. On the **General** tab (**General Settings** section), select the **Allow Negative Quantity** check box.
4. On the form toolbar, click **Save**.

Figure: Item Classes form

Item Classes

NOTES ACTIVITIES FILES CUSTOMIZATION TOOLS

Item Class Tree

- LABOR***** Labor
- NSTOCKITEM Non-stock item
- STOCKITEM* Stock item**

* Class ID: STOCKITEM - Stock item

Description: Stock item

GENERAL RESTRICTION GROUPS ATTRIBUTES

GENERAL SETTINGS

☒ Stock Item

☒ Allow Negative Quantity

☐ Accrue Cost

Item Type: Finished Good

Valuation Method: Average

Tax Category: EXEMPT - Exempt

Posting Class: STOCKITEM - Stock item

Price Class:

Default Warehouse:

* Availability Calculation ... STOCKITEM

Country Of Origin:

SHIPPING THRESHOLDS

Undershup Threshold (%): 100.00

Overship Threshold (%): 100.00

UNIT OF MEASURE

* Base Unit: PIECE ☒ Divisible Unit

* Sales Unit: PIECE ☒ Divisible Unit

* Purchase Unit: PIECE ☒ Divisible Unit

From Unit	Multiply/Divide	Conversion Factor	To Unit

PRICE MANAGEMENT

Price Workgro...

Price Manager:

Min. Markup %: 0.00

Step 4.2: Defining the Logic of Creating an SO Invoice

In `RSSVWorkOrderEntry` graph, will add the `CreateInvoice` static method to create SO Invoice for the current work order.

1. Define a `PXTransactionScope` – to save data from multiple graphs and avoid incomplete data being saved in case of any errors.
2. Create an instance of `SOInvoiceEntry` graph using `PXGraph.CreateInstance<SOInvoiceEntry>()` and initialize the summary (Document view of `SOInvoiceEntry`) of the Invoice.
3. Create an instance of `RSSVWorkOrderEntry` graph and assign the `workOrder` to Current Workorder.
4. Add the lines (Transactions view) associated with the repair items from the **Repair Items** tab (RepairItems view).
5. Add the lines (Transactions view) associated with labor from the **Labor** tab (Labor view).
6. Save the invoice to the database by calling `Actions.PressSave()` or `Actions.Save.Press()`.
7. Assign the generated invoice number and save the changes.
8. Complete/Close the `PXTransactionScope`.

Step 4.3: Defining the Create Invoice Action

1. In the `RSSVWorkOrderEntry` graph, will define the `CreateInvoiceAction` action, which adds the **Create Invoice** command to the **More** menu (under **Other**), adds the button with the same name on the form toolbar using `PXAction<RSSVWorkOrder> CreateInvoiceAction`.
2. Inside the action, populate a local list variable to contain the list of all workorders (`RSSVWorkOrder`).
3. Trigger the **Save** action to save changes to the database by calling `Actions.PressSave()`.
4. Get the `Current WorkOrder` and pass it as a parameter to `CreateInvoice` method.
5. Surround the invocation to `CreateInvoice` by `PXLongOperation.StartOperation()` to create an invoice for the current workorder asynchronously in a separate thread to process the long running operations.

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

For a repair work order, a user should be able to create an invoice only after the work order has been completed i.e., **Create Invoice** button should be visible only for work order with *Completed* status. And only one invoice can be created for a work order i.e., after the successful creation of the invoice, the button and command should be *disabled*.

In `RSSVWorkOrderEntry` graph, add `RowSelected<RSSVWorkOrder>` event is added.



















Inside the `RowSelected` event,









1. Will make the `CreateInvoiceAction` visible only when the *Current WorkOrder Status* is *Completed* – by calling `setVisible()` method.
2. Will make the `CreateInvoiceAction` enabled only when the *Current WorkOrder Status* is *Completed* and *InvoiceNbr* is empty (i.e., no invoice has been generated yet) – by calling `setEnabled()` method.

Step 4.5: Testing the Create Invoice Action

1. Open **Repair Work Orders (RS301000)** form.
2. Open *000001* repair work order.
3. On the form toolbar, click **Remove Hold** – to change status from *Hold* to *Ready for Assignment*.
4. On the form toolbar, click **Assign** – to change the status to *Assigned*.
5. On the form toolbar, click **Complete** – to change the status to *Completed*.
6. Now, notice that **Create Invoice** button/action is visible, click **Create Invoice**.
7. Notice that the *Invoice Nbr.* box is having the value of the newly created SO Invoice.

Repair Work Orders

NOT

Executing. Press to abort
00:00:01

CANCEL

Order Nbr.: 000001

Customer ID: C000000001 - Jersey Central Office Equi

Order Total: 40.00

Status: Completed

Service: BATTERYREPLACE - Battery Replacement

Invoice Nbr.:

* Date Created: 5/11/2020

Device: NOKIA3310 - Nokia 3310

Date Completed: 11/16/2021

Assignee: Andrews, Michael

Priority: Medium

Description: Battery replacement, Nokia 3310

REPAIR ITEMS

LABOR

↺ + ↻ × |↔| ☒

		Repair Item Type	Inventory ID	Description	Price
>		Back Cover	BCOV3310	Back cover for Nokia 3310	10.00
		Battery	BAT3310	Battery for Nokia 3310	20.00

Figure: Update of the Invoice Nbr box

Acumatica

Search...

Yogifon

11/16/2021
2:49 AM

admin, admin

Favorites

Data Views

Phone Repair Shop

Time and Expenses

Finance

Banking

Payables

Receivables

Sales Orders

Purchases

Repair Work Orders

000001 - Battery Replacement

NOT

✓ The operation has completed.

Order Nbr.: 000001

Status: Completed

* Date Created: 5/11/2020

Date Completed: 11/16/2021

Priority: Medium

Customer ID: C000000001 - Jersey Central Office Equi

Service: BATTERYREPLACE - Battery Replaceme

Device: NOKIA3310 - Nokia 3310

Assignee: Andrews, Michael

Description: Battery replacement, Nokia 3310

Order Total: 40.00

Invoice Nbr.: INV000049

REPAIR ITEMS

LABOR

Repair Item Type	Inventory ID	Description	Price
Back Cover	BCOV3310	Back cover for Nokia 3310	10.00
Battery	BAT3310	Battery for Nokia 3310	20.00

Your product is in trial mode. Only two concurrent users are allowed.

ACTIVATE

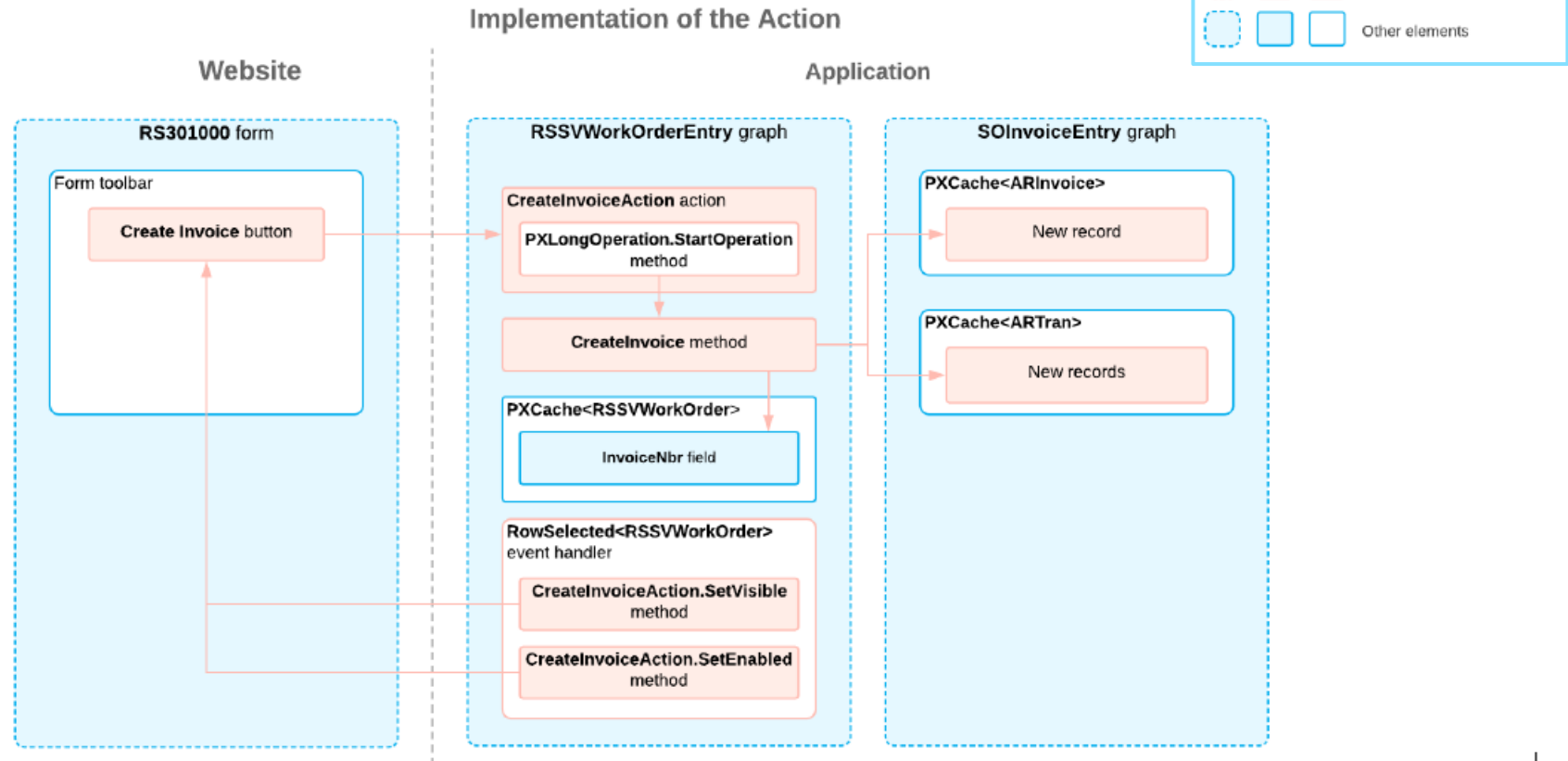
✓ Demo

Lesson Summary

In this lesson, you have learned how to initiate an asynchronous operation inside an action method by using the `PXLongOperation` class. Also, you have implemented the creation of an SO invoice based on a repair work order by doing the following in the `RSSVWorkOrderEntry` graph:

- Defining the static `CreateInvoice` method, which creates an instance of the `SOInvoiceEntry` graph
- Defining the `Create Invoice` button on the form toolbar and the command with the same name on the `More` menu; the underlying action initiates the asynchronous execution of the `CreateInvoice` method by using the `PXLongOperation` class
- Specifying the availability of the `Create Invoice` action in the `RowSelected` event handler so that only a single invoice can be created for a repair work order

Lesson Summary



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

Learning Objectives

In this lesson, you will learn how to do derive the value for a custom field from another form.

Step 5.1: Adding a Custom Field to the Payments and Applications Form

1. In **Payments and Applications (AR302000)** form, add a *Prepayment Percent* box to the form.
2. Add a column `UsrPrepaymentPercent` to `ARPayment` table with same parameters as `PrepaymentPercent` field in `RSSVSetup` table and the datatype is set to `decimal(9,6)`.
3. Create an extension for `ARPayment` or `ARRegister` DAC and add `UsrPrepaymentPercent` field.
4. In the **Screen Editor**, create checkbox control for `UsrPrepaymentPercent` in the Summary area of **Payments and Applications** form.
5. Correct the width for the *Prepayment Percent* label. To do this, in the `Column` element that is the parent to the *Prepayment Percent* element, for the `LabelsWidth` property, specify the *M* value.
6. Publish the customization project.

Figure: Prepayment Percent element

Customization Project Editor [Back](#) [Reload](#)

File Publish Extension Library Source Control

PhoneRepairShop

Screen Editor: AR302000 (Payments and Applications)

EDIT ASPX PREVIEW CHANGES ...

SCREENS

- AR302000
- IN202500
- RS101000
- RS201000
- RS202000
- RS203000
- RS301000

Data Access

Code

Files (23)

Generic Inquiries (3)

Reports

Dashboards

Site Map (6)

Database Scripts (12)

System Locales

Import/Export Scenarios

Shared Filters (1)

Access Rights

Wikis

Web Service Endpoints

Analytical Reports

Push Notifications

Business Events

Mobile Application

User-Defined Fields

Webhooks

Connected Applications

DataSource: ARPaymentEntry

style

Form: Document

- Column
- Column
- Column
- Column
 - ROT or RUT payment
 - Prepayment Percent**

style

Grid: docsTemplate

Tab

Dialogs

LAYOUT PROPERTIES ATTRIBUTES EVENTS ADD CONTROLS ADD DATA FIELDS

Override	Property	Value
<input checked="" type="checkbox"/>	Base Properties	
<input type="checkbox"/>	CommitChanges	
<input checked="" type="checkbox"/>	DataField	UsrPrepaymentPercent
<input checked="" type="checkbox"/>	ID	CstPXNumberEdit1
<input type="checkbox"/>	Size	
<input type="checkbox"/>	SkinID	
<input checked="" type="checkbox"/>	Ext Properties	
<input type="checkbox"/>	AlreadyLocalized	
<input type="checkbox"/>	AutoCallBack	
<input type="checkbox"/>	DisableSpellcheck	
<input type="checkbox"/>	DisplayFormat	
<input type="checkbox"/>	Enabled	
<input type="checkbox"/>	LabelWidth	
<input type="checkbox"/>	LinkCommand	
<input type="checkbox"/>	LocalizationInfo	
<input type="checkbox"/>	SuppressLabel	
<input type="checkbox"/>	SyncStateWithCommand	
<input type="checkbox"/>	Width	

Step 5.2: Deriving the Default Value of the PrepaymentPercent Field

To populate the `UsrPrepaymentPercent` field of the `ARPayment` extension when a payment is created, we can use either of the below:

1. `FieldDefaulting` event
2. `PXDefault` attribute

FieldDefaulting event:

1. Create an extension of `ARPaymentEntry` graph and add `FieldDefaulting` event for `UsrPrepaymentPercent` field of `ARPayment` extension.
2. Return the `PrepaymentPercent` value from `RSSVSetup` record selected using BQL as `NewValue` after checking for null to avoid `NullReferenceException`.

PXDefault attribute:

1. Add `PXDefault` attribute with the type from `RSSVSetup` field and `SourceField` is set as `RSSVSetup.prepaymentPercent` field to `UsrPrepaymentPercent` field in `ARPayment` DAC extension.

Step 5.3: Testing the Deriving of the Field Value

1. On the **Invoices (SO303000)** form, open the *INV000049* invoice (created during previous steps).
2. Release the invoice by typing *40* in the **Amount** box of the Summary area -> **Remove Hold** -> **Release**.
3. From More menu (**Processing**) -> **Pay** to open **Payments and Applications (AR302000)** form and in the Summary area, notice the **Prepayment Percent** box has a value of *10* from **Repair Work Order Preferences (RS101000)** form.
4. Save the payment.

Figure: The Prepayment Percent box

Payments and Applications

Payment - Jersey Central Office Equip

NOTES ACTIVITIES FILES CUSTOMIZATION TOOLS

← ↻ ↺ + 🗑️ 📄 < > >| REMOVE HOLD ...

Type: **Payment** Customer: C000000001 - Jersey Central Office Equip Payment Amo... 40.00 ↻ Prepayment Percent: 10.00

Reference Nbr.: <NEW> Payment Meth... CHECK - Check Payment Applied to Doc... 40.00

Status: On Hold Card/Account ... Applied to Ord... 0.00

* Application Date: 11/16/2021 * Cash Account: 102000-YOGI - Checking Account Available Bala... 0.00

* Application Pe... 11-2021 Write-Off Amo... 0.00

* Payment Ref.: 000001 Finance Chrg... 0.00

Deducted Cha... 0.00

Description:

DOCUMENTS TO APPLY SALES ORDERS APPLICATION HISTORY FINANCIAL APPROVALS CHARGES

↻ + × LOAD DOCUMENTS AUTO APPLY |<| ☒

📄	🔍	📄	☑	Branch	Doc. Type	*Reference Nbr.	Amount Paid	Cash Discount Taken	Write-Off Amount	Write-Off Reason Code	Date	Due Date	Cash Discount Date	Cross Ra
>	🔍	📄	☑	YOGIFON	Invoice	INV000049	40.00	0.00	0.00		11/16/2021	12/16/2021	11/16/2021	1.0000000

⏪ ⏩ ⏴ ⏵

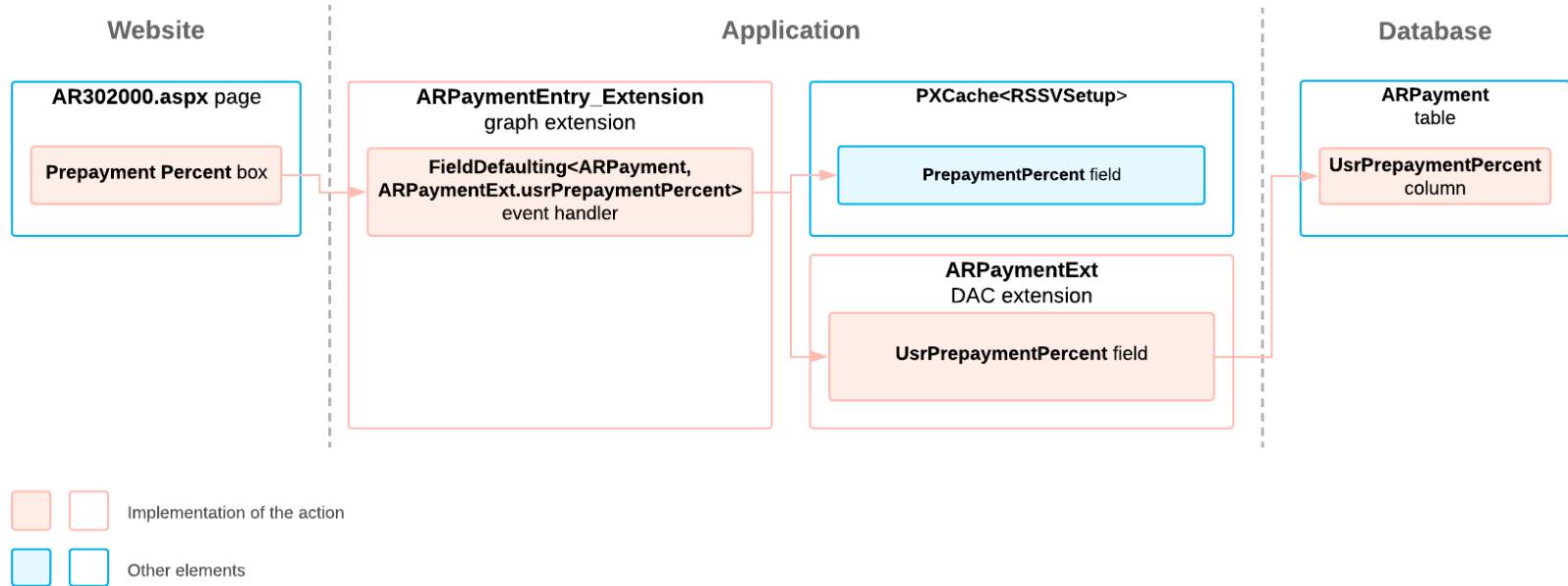
Lesson Summary

In this lesson, you created a custom field on the Payments and Applications (AR302000) form and learned how to assign its default value, which is derived from another entity. To assign a default value for a custom entity, you have done the following:

1. Defined the extension of a graph in which the field is initialized
2. In the graph extension, defined the FieldDefaulting event handler for the custom field

Lesson Summary

Deriving of a Field Value from Another Entity



Lesson 6: Debugging Customization Code

Learning Objectives

In this lesson, you will learn how to debug the source code of Acumatica ERP.

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" />`

When the environment is slow

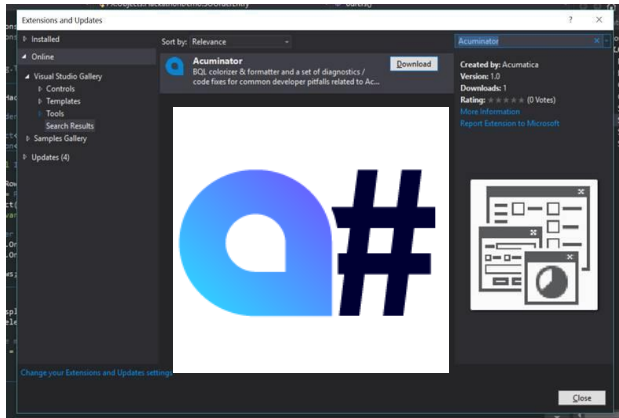
To Debug

Acumatica
The Cloud ERP

Useful Development Environment Optimization

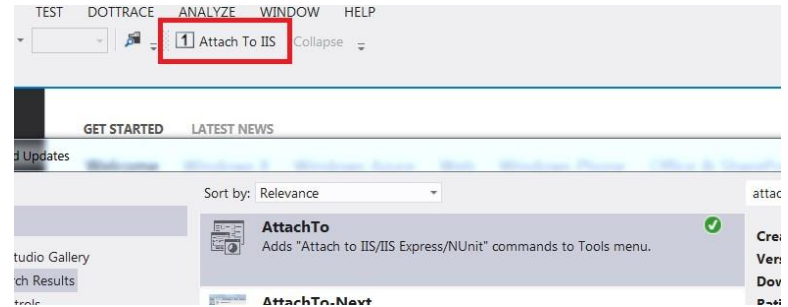
“Acuminator” Extension

Static code analysis, colorizer and suggestions tool for Acumatica Framework



“Attach To” Extension

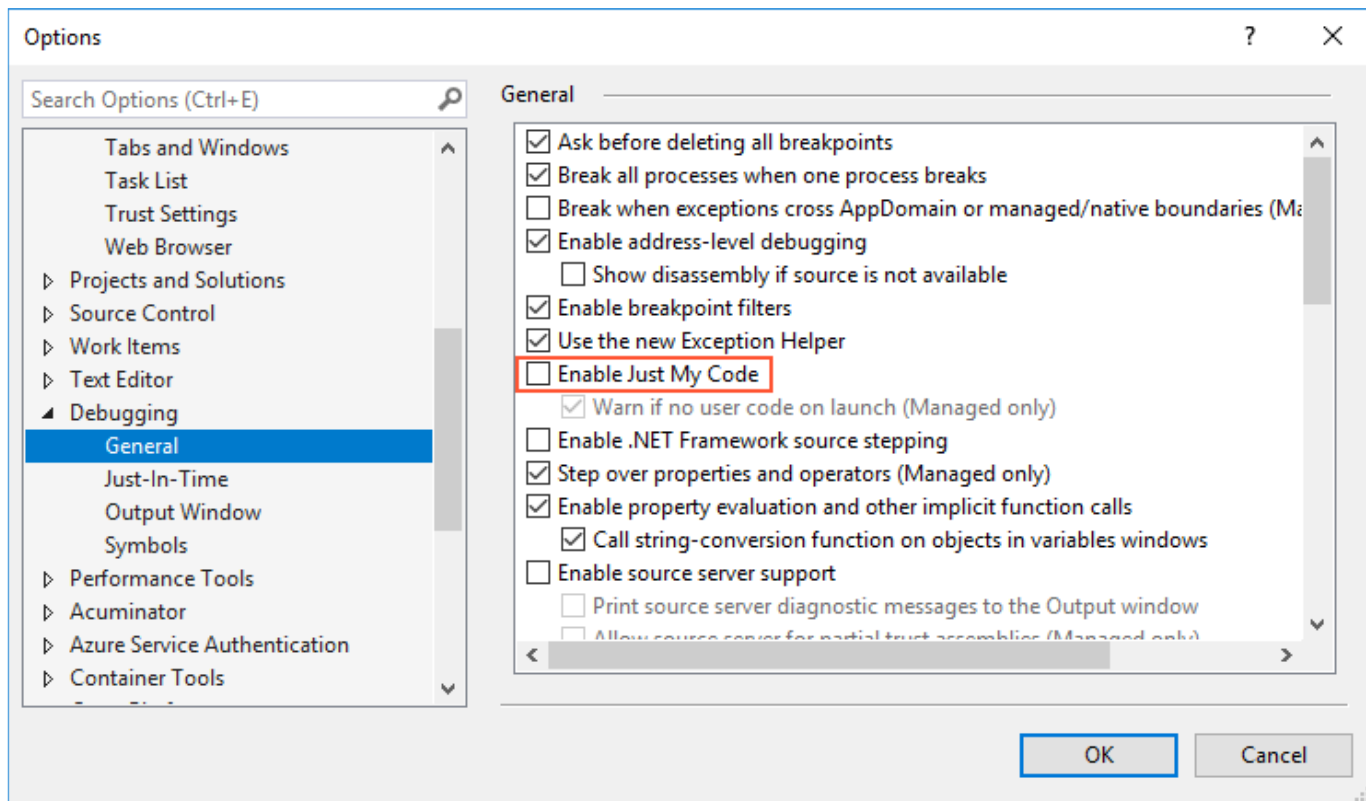
- Attach Debugger to Acumatica with 1-click



Step 6.1: Debugging the Acumatica ERP Source Code

1. Make sure the Acumatica program database (PDB) files are in the `Bin` folder of the Acumatica ERP instance folder that you use for the training course (for example, in `PhoneRepairShop\Bin`). PDB files are copied when **Install Debugger Tools** option is checked during installation from **Acumatica Configuration Wizard**. A PDB file contains the link between compiler instructions and some lines in source code.
2. Configure the `web.config` file, in `<system.web>` tag and set `<compilation debug="True" ...>` and Save.
3. In Visual Studio, open the `PhoneRepairShop_Code` solution, which includes both the `PhoneRepairShop_Code` project and the `PhoneRepairShop` website.
4. In the Visual Studio's main menu, select **Tools > Options > Debugging > General > Enable Just My Code**.
5. In the **Debugging > Symbols** section, in the **Symbols file (.pdb) locations** list, add the path to the location of the PDB files of the instance and click **OK**.

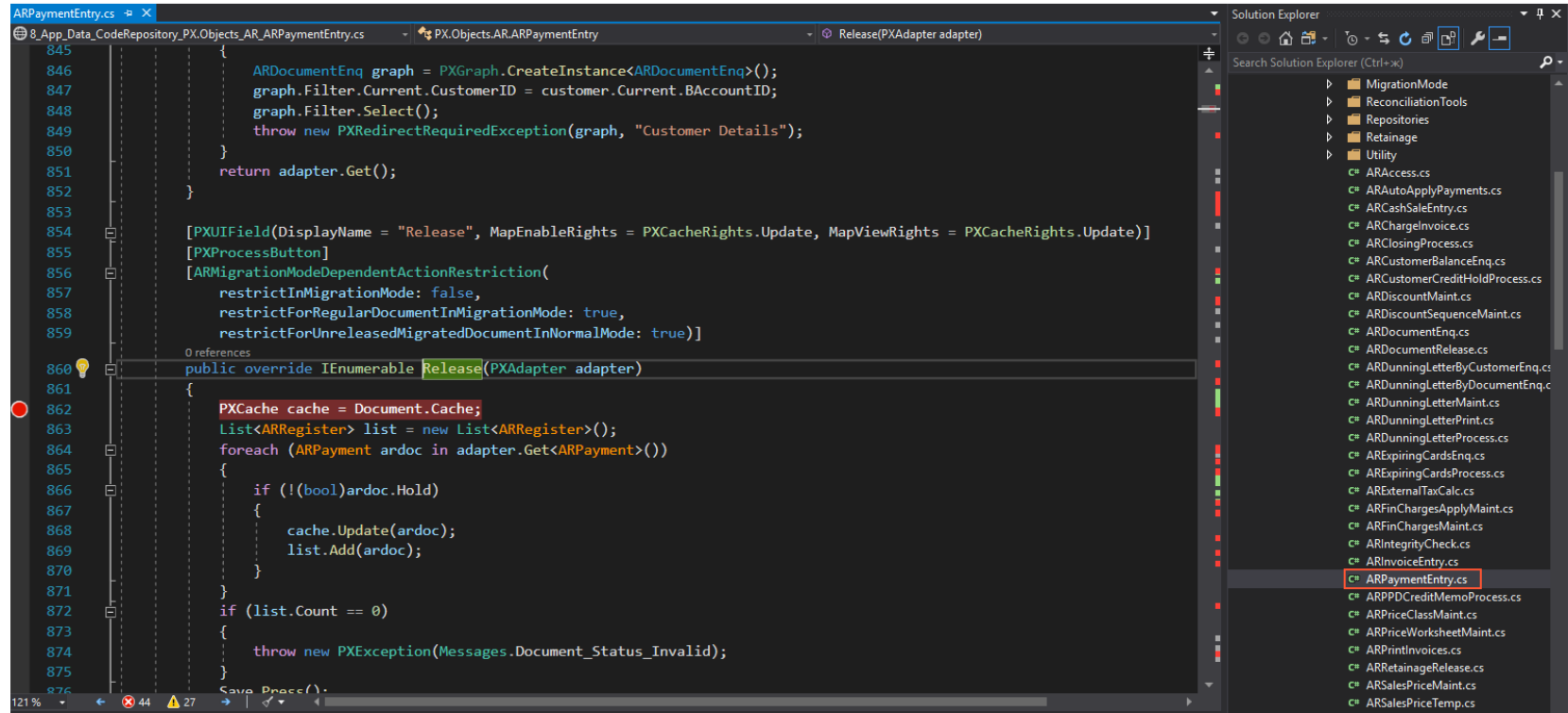
Figure: Clearing the Enable Just My Code check box



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

1. In Visual Studio, open the Acumatica ERP source code files. For the `PhoneRepairShop` instance, all files are in the `PhoneRepairShop/App_Data/CodeRepository` folder.
2. In the Solution Explorer, select **PhoneRepairShop > App_Data > CodeRepository > PX.Objects > AR > ARPaymentEntry.cs**, and go to the definition of the `IEnumerable Release(PXAdapter adapter)` method.
3. Add a breakpoint inside the `Release` method.
4. Attach the Visual Studio debugger to the `w3wp.exe` process.
5. Start debugging by navigating to **Payments and Applications** form, creating a payment and clicking **Release** button – in turn invoking the `Release` method.

Figure: Viewing the source code of the Release action



Lesson Summary

In this lesson, you have learned how to debug the code of Acumatica ERP by using program database (PDB) files.



Thank you!

Vidhyalakshmi Hariharasubramanian