

Developer Course



T220 Data Entry and Setup Forms 2022 R1

Revision: 4/7/2022

Contents

Copyright.....	4
Introduction.....	5
How to Use This Course.....	6
Course Prerequisites.....	7
Initial Configuration.....	8
Step 1: Preparing the Environment.....	8
Step 2: Preparing the Needed Acumatica ERP Instance for the Training Course.....	8
Step 3: Creating the Database Tables.....	9
Company Story and Customization Description.....	10
Part 1: Data Entry Form (Repair Work Orders).....	12
Lesson 1.1: Configuring a Complex Form Layout.....	12
Step 1.1.1: Creating the Form—Self-Guided Exercise.....	15
Step 1.1.2: Configuring the Controls of the Summary Area.....	19
Step 1.1.3: Configuring the Layout of the Summary Area of the Form.....	25
Step 1.1.4: Configuring Form View Mode for the Grid.....	28
Step 1.1.5: Adding the Substitute Form with a Shared Filter to the Project.....	31
Lesson Summary.....	32
Review Questions.....	35
Additional Information: Configuration of Controls.....	35
Additional Information: Layout Configuration.....	36
Lesson 1.2: Copying Field Values from One Record to Another.....	36
Step 1.2.1: Creating a Work Order from a Template (with RowUpdated).....	36
Step 1.2.2: Updating Fields of the Same Record on Update of a Field (with FieldUpdated and FieldDefaulting)—Self-Guided Exercise.....	39
Lesson Summary.....	40
Review Question.....	41
Lesson 1.3: Validating the Field Values.....	41
Step 1.3.1: Validating an Independent Field Value (with FieldVerifying).....	41
Step 1.3.2: Validating Dependent Fields of Records (with RowUpdating).....	45
Lesson Summary.....	47
Review Questions.....	48
Part 2: Setup Form (Repair Work Order Preferences).....	50
Lesson 2.1: Configuring the Auto-Numbering of a Field Value.....	50
Step 2.1.1: Creating the Form—Self-Guided Exercise.....	51

Step 2.1.2: Configuring the DAC for the Setup Form (with PXPrimaryGraph and PXCacheName).....	53
Step 2.1.3: Configuring the Auto-Numbering of Records (with CS.AutoNumberAttribute).....	54
Lesson Summary.....	58
Review Questions.....	59
Additional Information: Custom Feature Switches.....	60
Appendix: Use of Event Handlers.....	61
Appendix: Reference Implementation.....	62
Appendix: Deploying the Needed Acumatica ERP Instance for the Training Course.....	63
Appendix: Publishing the Required Customization Project.....	64

Copyright

© 2022 Acumatica, Inc.

ALL RIGHTS RESERVED.

No part of this document may be reproduced, copied, or transmitted without the express prior consent of Acumatica, Inc.

3933 Lake Washington Blvd NE, # 350, Kirkland, WA 98033

Restricted Rights

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

Disclaimer

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

Trademarks

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

Software Version: 2022 R1

Last Updated: 04/07/2022

Introduction

The *T220 Data Entry and Setup Forms* training course shows how to create data entry and setup forms by using Acumatica Framework and customization tools of Acumatica ERP. The course describes in detail how to define the complex layout of a data entry form and implement the business logic of the form (such as insertion of data from a template and validation of a field value). The course also shows how to provide configuration parameters for a data entry form by using a setup form.

This course is intended for application developers who are starting to learn how to customize Acumatica ERP.

The course is based on a set of examples that demonstrate the general approach to customizing Acumatica ERP. It is designed to give you ideas about how to develop your own embedded applications through the customization tools. As you go through the course, you will continue the development of the customization for the cell phone repair shop, which was performed in the *T200 Maintenance Forms* and *T210 Customized Forms and Master-Detail Relationship* training courses (which we recommend that you take before completing the current course).

After you complete all the lessons of the course, you will be familiar with the programming techniques for the definition of the complex layout of Acumatica ERP forms, the implementation of particular business logic scenarios, and the configuration of setup parameters of Acumatica ERP forms.



We recommend that you complete the examples in the order in which they are provided in the course, because some examples use the results of previous ones.

How to Use This Course

To complete this course, you will complete the lessons from each part of the course in the order in which they are presented and then pass the assessment test. More specifically, you will do the following:

1. Complete [Course Prerequisites](#), perform [Initial Configuration](#), and carefully read [Company Story and Customization Description](#).
2. Complete the lessons in both parts of the training guide.
3. In Partner University, take *T220 Certification Test: Data Entry and Setup Forms*.

After you pass the certification test, you will receive the Partner University certificate of course completion.

What Is in a Part?

The first part of the course explains how to create a data entry form, configure its layout, and implement basic business logic scenarios.

The second part of the course shows how to create a setup form and configure the automatic numbering of data records on the data entry form.

Each part of the course consists of lessons you should complete.

What Is in a Lesson?

Each lesson is dedicated to a particular development scenario that you can implement by using Acumatica ERP customization tools and Acumatica Framework. Each lesson consists of a brief description of the scenario and an example of the implementation of this scenario.

The lesson may also include *Additional Information* topics, which are outside of the scope of this course but may be useful to some readers.

Each lesson ends with a *Lesson Summary* topic, which summarizes the development techniques used during the implementation of the scenario.

What Are the Documentation Resources?

The complete Acumatica ERP and Acumatica Framework documentation is available on <https://help.acumatica.com/> and is included in the Acumatica ERP instance. While viewing any form used in the course, you can click the **Open Help** button in the top pane of the Acumatica ERP screen to bring up a form-specific Help menu; you can use the links on this menu to quickly access form-related information and activities and to open a reference topic with detailed descriptions of the form elements.

Licensing Information

For the educational purposes of this course, you use Acumatica ERP under the trial license, which does not require activation and provides all available features. For the production use of the Acumatica ERP functionality, an administrator has to activate the license the organization has purchased. Each particular feature may be subject to additional licensing; please consult the Acumatica ERP sales policy for details.

Course Prerequisites

To complete this course, you should be familiar with the basic concepts of Acumatica Framework and Acumatica Customization Platform. We recommend that you complete the *T200 Maintenance Forms* and *T210 Customized Forms and Master-Detail Relationship* training courses before you begin this course.

Required Knowledge and Background

To complete the course successfully, you should have the following required knowledge:

- Proficiency with C#, including but not limited to the following features of the language:
 - Class structure
 - OOP (inheritance, interfaces, and polymorphism)
 - Usage and creation of attributes
 - Generics
 - Delegates, anonymous methods, and lambda expressions
- Knowledge of the following main concepts of ASP.NET and web development:
 - Application states
 - The debugging of ASP.NET applications by using Visual Studio
 - The process of attaching to IIS by using Visual Studio debugging tools
 - Client- and server-side development
 - The structure of web forms
- Experience with SQL Server, including doing the following:
 - Writing and debugging complex SQL queries (WHERE clauses, aggregates, and subqueries)
 - Understanding the database structure (primary keys, data types, and denormalization)
- The following experience with IIS:
 - The configuration and deployment of ASP.NET websites
 - The configuration and securing of IIS

Initial Configuration

You need to perform the prerequisite actions described in this part before you start to complete the course.

If you have deployed an instance for the *T210 Customized Forms and Master-Detail Relationship* course and have the customization project and the source code for this course, you can perform only Step 3.

Step 1: Preparing the Environment



If you have completed the *T210 Customized Forms and Master-Detail Relationship* training course and are using the same environment for the current course, you can skip this step.

You should prepare the environment for the training course as follows:

1. Make sure the environment that you are going to use for the training course conforms to the [System Requirements for Acumatica ERP 2022 R1](#).
2. Make sure that the Web Server (IIS) features that are listed in [Configuring Web Server \(IIS\) Features](#) are turned on.
3. Install the Acuminator extension for Visual Studio.
4. Clone or download the customization project and the source code of the extension library from the [Help-and-Training-Examples](#) repository in Acumatica GitHub to a folder on your computer.
5. Install Acumatica ERP. On the Main Software Configuration page of the installation program, select the **Install Acumatica ERP** and **Install Debugger Tools** check boxes.



If you have already installed Acumatica ERP without debugger tools, you should remove Acumatica ERP and install it again with the **Install Debugger Tools** check box selected. The reinstallation of Acumatica ERP does not affect existing Acumatica ERP instances. For details, see [To Install the Acumatica ERP Tools](#).

Step 2: Preparing the Needed Acumatica ERP Instance for the Training Course



If you have completed the *T210 Customized Forms and Master-Detail Relationship* training course, instead of deploying a new instance, you can use the Acumatica ERP instance that you deployed and used for the training course.

You deploy an Acumatica ERP instance and configure it as follows:

1. Open the Acumatica ERP Configuration Wizard, and do the following:
 - a. Click **Deploy New Application Instance for T-series Developer Courses**.
 - b. On the **Database Configuration** page, make sure the name of the database is `PhoneRepairShop`.
 - c. On the **Instance Configuration** page, do the following:
 - a. In the **Local Path of the Instance** box, select a folder that is outside of the `C:\Program Files (x86)` and `C:\Program Files` folders. (We recommend that you store the website folder outside of these folders to avoid an issue with permission to work in these folders when you perform customization of the website.)

- b. In the **Training Course** box, select the training course you are taking.

The system creates a new Acumatica ERP instance, adds a new tenant, loads the data to it, and publishes the customization project that is needed for this training course.

2. Make sure a Visual Studio solution is available in the `App_Data\Projects\PhoneRepairShop` folder of the Acumatica ERP instance folder. This is the solution of the extension library that you will modify in this course.
3. Sign in to the new tenant by using the following credentials:
 - **Username:** admin
 - **Password:** setup

Change the password when the system prompts you to do so.

4. In the top right corner of the Acumatica ERP screen, click the username, and then click **My Profile**. On the **General Info** tab of the *User Profile* (SM203010) form, which the system has opened, select **YOGIFON** in the **Default Branch** box; then click **Save** on the form toolbar.

In subsequent sign-ins to this account, you will be signed in to this branch.

5. Optional: Add the *Customization Projects* (SM204505) and *Generic Inquiry* (SM208000) forms to your favorites. For details about how to add a form to your favorites, see *Managing Favorites: General Information*.



If for some reason you cannot complete instructions in this step, you can create an Acumatica ERP instance as described in *Appendix: Deploying the Needed Acumatica ERP Instance for the Training Course* and manually publish the needed customization project as described in *Appendix: Publishing the Required Customization Project*.

Step 3: Creating the Database Tables

Create the database tables that are necessary for the *T220 Data Entry and Setup Forms* training course and include the scripts in the customization project as follows:

1. In SQL Server Management Studio, execute the `T220_DatabaseTables.sql` script to create the database tables that are necessary for the *T220 Data Entry and Setup Forms* training course.
The script creates the following tables, which are new for this course: `RSSVWorkOrder`, `RSSVWorkOrderItem`, `RSSVWorkOrderLabor`, and `RSSVSetup`.
2. On the Database Scripts page of the Customization Project Editor, for each added table, do the following:
 - a. On the page toolbar, click **Add Custom Table Schema**.
 - b. In the dialog box that opens, select the table and click **OK**.
3. Publish the project.



The design of database tables is outside of the scope of this course. For details on designing database tables for Acumatica ERP, see *Designing the Database Structure and DACs*.

Company Story and Customization Description

In this course, you will continue the development to support the cell phone repair shop of the Smart Fix company; you began this development while completing the *T200 Maintenance Forms* and *T210 Customized Forms and Master-Detail Relationship* training courses.



If you have not completed these training courses, you have loaded and published the customization project with the results of these courses.

In the *T200 Maintenance Forms* training course, you have created two simple maintenance forms, Repair Services (RS201000) and Serviced Devices (RS202000), which the Smart Fix company uses to manage the lists of, respectively, repair services that the company provides and devices that can be serviced.

In the *T210 Customized Forms and Master-Detail Relationship* course, you have created another maintenance form, Services and Prices (RS203000), and customized the [Stock Items](#) (IN202500) form of Acumatica ERP. The Services and Prices form provides users with the ability to define and maintain the price for each provided repair service. The Stock Items form has been customized to mark particular stock items as repair items—that is, items that are used for the repair services.

In this course, you will create the Repair Work Orders (RS301000) data entry form, which is used to create and manage work orders for repairs. You will also create the Repair Work Order Preferences (RS101000) setup form, which an administrative user will use to specify the company's preferences for the repair work orders.

Repair Work Orders Form

The following screenshot shows how the Repair Work Orders (RS301000) form will look.

Repair Work Orders

000001 - Battery Replacement

NOTES FILES CUSTOMIZATION TOOLS

← ↻ ↺ + ✖ 📄 ⏪ ⏩ ⏴ ⏵

* Order Nbr.: 000001 * Customer ID: C000000001 - Jersey Central Office E Order Total: 40.00

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

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

Date Completed: Assignee:

Priority: Low Description:

REPAIR ITEMS LABOR

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

⏪ ⏩ ⏴ ⏵

Figure: Repair Work Orders form

The form will contain the following tabs:

- **Repair Items:** Will show the list of repair items (stock items) necessary to complete the repair work order.
- **Labor:** Will contain the list of labor items (non-stock items) that are performed for the selected repair work order.

You will also import a substitute form of the inquiry type with a preconfigured filter of records; this substitute form will serve as an entry point to the Repair Work Orders form.

This form will use the following custom tables, which you have added to the application database in the course prerequisites:

- **RSSVWorkOrder**: The data of this table will be displayed in the Summary area of the form.
- **RSSVWorkOrderItem**: The data of this table will be displayed on the **Repair Items** tab.
- **RSSVWorkOrderLabor**: The data of this table will be displayed on the **Labor** tab.

Repair Work Order Preferences Form

The following screenshot shows how the Repair Work Order Preferences (RS101000) form will look when you have developed it.

Repair Work Order Preferences

NOTES FILES CUSTOMIZATION TOOLS

* Numbering Sequence: WORKORDER - Rep

* Walk-In Customer: C000000001 - Jersey

Default Employee: Becher, Joseph

* Prepayment Percent: 10.00

Figure: Repair Work Order Preferences form

The form will contain the following elements:

- The **Numbering Sequence** box will hold the numbering sequence that is used to auto-number repair work orders.
- The **Walk-In Customer** box will contain the identifier of the customer record that should be used as the customer for walk-in orders.
- The **Default Employee** box will specify the default assignee for repair work orders.
- The **Prepayment Percent** box will contain the percent of prepayment that a customer should pay for a service that requires prepayment—that is, a service that has the **Requires Prepayment** check box selected on the Repair Services (RS201000) form.

This form will use the **RSSVSetup** custom table, which you have added to the application database in the course prerequisites.

Part 1: Data Entry Form (Repair Work Orders)

The Smart Fix company needs to have a custom Acumatica ERP form, on which users will create repair work orders. For this purpose, in this part of the course, you will create the Repair Work Orders (RS301000) custom data entry form, which is described in [Company Story and Customization Description](#).

Data entry forms are the most frequently used forms of Acumatica ERP. Typically, these forms are used for the input of business documents, such as sales orders and cases.

These forms have IDs that start with a two-letter abbreviation (indicating the functional area of the form) followed by 30, such as *RS301000*. The names of the graphs that work with data entry forms have the *Entry* suffix. For instance, *RSSVWorkOrderEntry* will be the name of the graph for the Repair Work Orders form. For details about the naming conventions for the ASPX pages and graphs, see [Form and Report Numbering](#) and [Graph Naming](#).

After you complete the lessons of this part, you will be able to test the functionality of the form.

Lesson 1.1: Configuring a Complex Form Layout

In this lesson, you will learn how to adjust the layout of controls on a form and how to configure a grid. As an example, you will use the Repair Work Orders (RS301000) custom data entry form, which you will create in this lesson.

You will also import a generic inquiry that presents the data entered on the Repair Work Orders form (the *entry form* in this context) in a tabular format; a shared filter has been developed for the substitute form to filter the data. The generic inquiry will function as a *substitute form* because it will be brought up instead of the entry form when a user clicks the form name in a workspace. You will include both the generic inquiry and the filter in the customization project along with the entry form.

Description of the Form Elements

The Summary area of the Repair Work Orders (RS301000) data entry form will contain the following elements:

- **Order Nbr.:** A box in which a user can add a new repair work order number or select an existing repair work order number to view the repair work order identified by the number. The repair work order number is a six-character string that can contain only digits.
- **Status:** A read-only box that displays the status of the repair work order, which is one of the following:
 - *On Hold*
 - *Pending Payment*
 - *Ready for Assignment*
 - *Assigned*
 - *Completed*
 - *Paid*
- **Date Created:** A box that is used to specify the date of creation of the repair work order. The system will insert the current business date by default.
- **Date Completed:** A read-only box that is filled in by the system when the repair work order is assigned the *Completed* status. The business logic that fills in the value of this box will be implemented in a later training course.
- **Priority:** A box in which the user can select the priority of the repair.
- **Customer ID:** A box in which the user can select the ID of the customer who requested the repair.
- **Service:** A box in which the user can select one of the services configured on the custom Repair Services (RS201000) form.

- **Device:** A box in which the user can select one of the devices configured on the custom Serviced Devices (RS203000) form.
- **Assignee:** A box in which the user can select an employee who performs the repair.
- **Description:** A box in which the user can type the description of the repair work order.
- **Order Total:** A read-only box that displays the price of the repair.
- **Invoice Nbr.:** A read-only box that displays the number of the invoice created for the repair work order. The business logic that fills in the value of this box will be implemented in the *T230 Actions* training course.

The **Repair Items** tab will contain the same columns as the **Repair Items** tab of the Services and Prices (RS203000) form except for the **Required** and **Default** columns. This tab supports form view mode, in which users can edit a particular record selected in the grid in the form layout.

The **Labor** tab will contain the same columns as the **Labor** tab of the Services and Prices form.

Layout of the Form

In the Summary area of the form, you will adjust the layout as follows:

- Arrange the input controls in three columns on the form
- Adjust the widths of controls and labels
- Expand the **Description** box to span two columns

Also, you will make the following adjustments to the **Repair Items** tab, which contains the grid:

- Enable form view mode, and configure the input controls
- Arrange the input controls into two labeled groups, each in a separate column

The resulting layout of the Repair Work Orders form is shown in the screenshot below.

The screenshot displays the 'Repair Work Orders' form. The title bar shows 'Repair Work Orders' and '000001 - Battery Replacement'. The top navigation bar includes 'NOTES', 'FILES', 'CUSTOMIZATION', and 'TOOLS'. Below the title bar is a toolbar with various icons. The form is divided into two main sections: 'Summary' and 'Repair Items'. The 'Summary' section contains fields for 'Order Nbr.' (000001), 'Status' (On Hold), 'Date Created' (3/3/2022), 'Priority' (Low), 'Customer ID' (C000000001 - Jersey Central Office E), 'Service' (BATTERYREPLACE - Battery Replace), 'Device' (NOKIA3310 - Nokia 3310), 'Assignee', 'Description', 'Order Total' (40.00), and 'Invoice Nbr.'. The 'Repair Items' tab is active, showing a table with columns: 'Repair Item Type', 'Inventory ID', 'Description', and 'Price'. The table contains two rows: 'Battery' (BAT3310, Battery for Nokia 3310, 20.00) and 'Back Cover' (BCOV3310, Back cover for Nokia 3310, 10.00).

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

Figure: The Repair Work Orders form

Database Tables Used for the Form

Below is the class diagram, which shows the relationships between the `RSSVWorkOrder`, `RSSVWorkOrderItem`, and `RSSVWorkOrderLabor` data access classes that are used for this form. The `RSSVWorkOrderItem.OrderNbr` and `RSSVWorkOrderLabor.OrderNbr` fields are the references to the data record of the master `RSSVWorkOrder` class, while the `RSSVWorkOrderItem.InventoryID` and

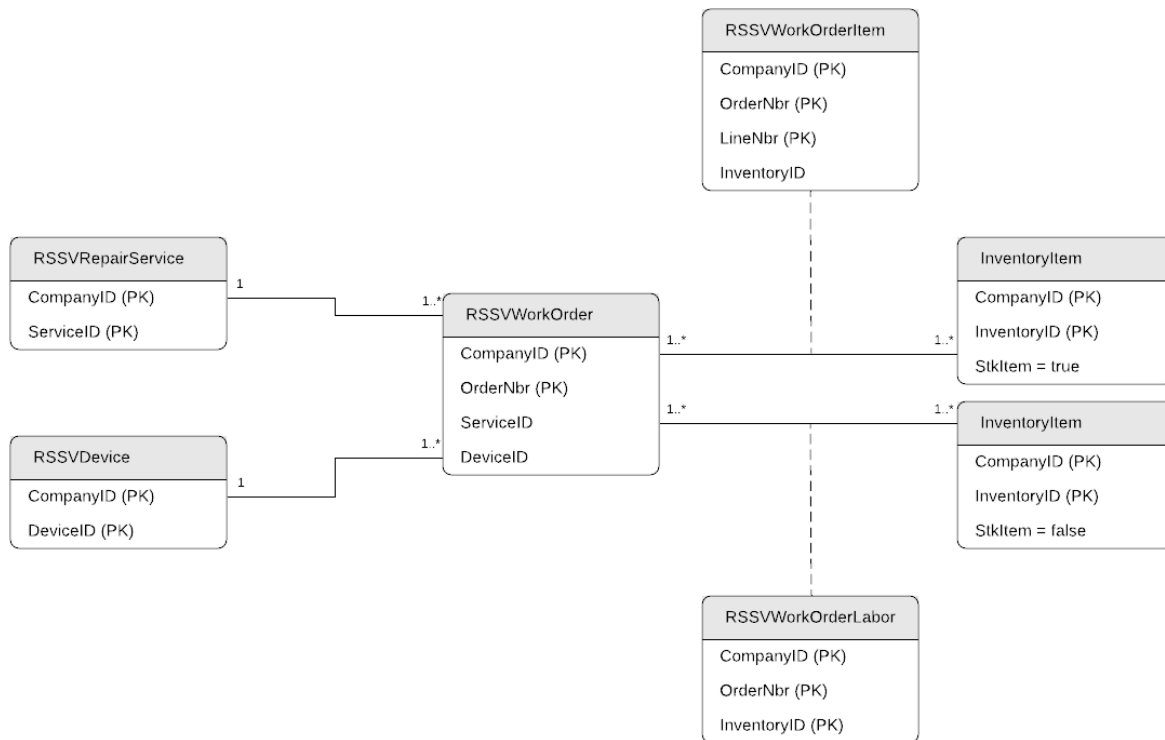
`RSSVWorkOrderLabor.InventoryID` fields are the references to the non-stock item added to the repair work order.

The structure of the key fields of the `RSSVWorkOrderItem` and `RSSVWorkOrderLabor` DACs is copied from the structure of the key fields of the `RSSVRepairItem` and `RSSVLabor` DACs, respectively. The data of `RSSVRepairItem` and `RSSVLabor` will be used to fill values of `RSSVWorkOrderItem` and `RSSVWorkOrderLabor` in [Step 1.2.1: Creating a Work Order from a Template \(with RowUpdated\)](#). The `RSSVWorkOrderItem` DAC has the additional `LineNbr` key field because users can add multiple repair items with the same service ID, device ID, and inventory ID to the **Repair Items** tab of the Repair Work Orders (RS301000) form.

For a repair work order, the user specifies the repair service and device, which are represented as the references to the corresponding classes as follows:

- `RSSVWorkOrder.ServiceID` is a reference to the `RSSVRepairService` class.
- `RSSVWorkOrder.DeviceID` is a reference to the `RSSVDevice` class.

Tables for the Repair Work Orders Form



Lesson Objectives

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

- Align controls on a form
- Adjust the size of controls and labels
- Adjust a control to span several columns
- Configure the form view of a grid
- Add a substitute form with a shared filter to the customization project

Step 1.1.1: Creating the Form—Self-Guided Exercise

In this step, you will create the Repair Work Orders (RS301000) form on your own. Although this is a self-guided exercise, you can use the details and suggestions in this topic as you create the form. The creation of a form is described in detail in the *T200 Maintenance Forms* training course.

If you are using the Customization Project Editor to complete the self-guided exercise, you can follow this instruction:

1. Create the form and graph as follows:
 - a. On the toolbar of the Customized Screens page of the Customization Project Editor, click **Create New Screen**.
 - b. In the **Create New Screen** dialog box, which opens, specify the following values:
 - **Screen ID:** RS.30.10.00
 - **Graph Name:** RSSVWorkOrderEntry
 - **Graph Namespace:** PhoneRepairShop
 - **Page Title:** Repair Work Orders
 - **Template:** FormTab
 - c. Move the generated RSSVWorkOrderEntry graph to the extension library.
2. Create and configure the DACs as specified below:
 - a. In Code Editor, generate the RSSVWorkOrder, RSSVWorkOrderItem, and RSSVWorkOrderLabor DACs and move them to the extension library.
 - b. Configure the DACs in Visual Studio as follows:
 - RSSVWorkOrder: Specify the system attributes and other attributes as shown in the code fragments below:
 - For the DAC, define the following attribute.

```
[PXCacheName("Repair Work Order")]
```

- Define attributes for the system fields. (For details about the definition of the attributes of the system fields, see [Step 1.4.2: Configure the Attributes of the New DAC](#) in the *T200 Maintenance Forms* training course or see [Audit Fields](#), [Concurrent Update Control \(TStamp\)](#), and [Attachment of Additional Objects to Data Records \(NoteID\)](#) in the documentation.)
- Define the attributes of the RepairItemLineCntr field, which is not displayed on the UI, as follows. (You will configure the attributes of the fields that are displayed in the UI in [Step 1.1.2: Configuring the Controls of the Summary Area](#).)

```
[PXDBInt()]
[PXDefault(0)]
public virtual int? RepairItemLineCntr { get; set; }
public abstract class repairItemLineCntr :
    PX.Data.BQL.BqlInt.Field<repairItemLineCntr> { }
```

You need this field to define the numbering of repair items on the **Repair Items** tab of the Repair Work Orders (RS301000) form by using the predefined PXLineNbr attribute. For details about this approach, see [Step 2.1.3: Numbering Detail Records \(with PXLineNbr\)](#) in the *T210 Customized Forms and Master-Detail Relationship* training course.

- RSSVWorkOrderItem: Specify the system attributes and other attributes as shown in the code fragments below:
 - For the DAC, define the following attribute.

```
[PXCacheName("Repair Item Included in Repair Work Order")]
```

- Configure the same attributes that have been configured for the corresponding fields of the RSSVRepairItem DAC except for the PXParent attribute, which you need to assign to the OrderNbr field.
- Make OrderNbr and LineNbr to be the key fields.



You need to make the LineNbr a key field so that a user can add multiple items with the same InventoryID. In this case, the LineNbr is an alternative field to the InventoryID field.

The RSSVWorkOrderItem DAC fields excluding the system fields should look as shown in the following code.

```
[PXCacheName("Repair Item Included in Repair Work Order")]
public class RSSVWorkOrderItem : IBqlTable
{
    #region OrderNbr
    [PXDBString(15, IsKey = true, IsUnicode = true, InputMask = "")]
    [PXDBDefault(typeof(RSSVWorkOrder.orderNbr))]
    [PXParent(typeof(SelectFrom<RSSVWorkOrder>.
        Where<RSSVWorkOrder.orderNbr.
        IsEqual<RSSVWorkOrderItem.orderNbr.FromCurrent>>))]
    public virtual string OrderNbr { get; set; }
    public abstract class orderNbr : PX.Data.BQL.BqlString.Field<orderNbr>
    { }

    #endregion

    #region LineNbr
    [PXDBInt(IsKey = true)]
    [PXLineNbr(typeof(RSSVWorkOrder.repairItemLineCtr))]
    [PXUIField(DisplayName = "Line Nbr.", Visible = false)]
    public virtual int? LineNbr { get; set; }
    public abstract class lineNbr : PX.Data.BQL.BqlInt.Field<lineNbr> { }
    #endregion

    #region RepairItemType
    [PXDBString(2, IsFixed = true)]
    [PXStringList(
        new string[]
        {
            RepairItemTypeConstants.Battery,
            RepairItemTypeConstants.Screen,
            RepairItemTypeConstants.ScreenCover,
            RepairItemTypeConstants.BackCover,
            RepairItemTypeConstants.Motherboard
        },
        new string[]
        {
            Messages.Battery,
            Messages.Screen,
            Messages.ScreenCover,
            Messages.BackCover,
            Messages.Motherboard
        }
    )]
    [PXUIField(DisplayName = "Repair Item Type")]
}
```



```

        public virtual string RepairItemType { get; set; }
        public abstract class repairItemType :
PX.Data.BQL.BqlString.Field<repairItemType> { }
        #endregion

        #region InventoryID
        [Inventory]
        [PXRestrictor(typeof(
            Where<InventoryItemExt.usrRepairItem.IsEqual<True>.
            And<Brackets<
                RSSVWorkOrderItem.repairItemType.FromCurrent.IsNull.
                Or<InventoryItemExt.usrRepairItemType.
IsEqual<RSSVWorkOrderItem.repairItemType.FromCurrent>>>>>),
            Messages.StockItemIncorrectRepairItemType,
            typeof(RSSVWorkOrderItem.repairItemType)))]
        public virtual int? InventoryID { get; set; }
        public abstract class inventoryID :
PX.Data.BQL.BqlInt.Field<inventoryID> { }
        #endregion

        #region BasePrice
        [PXDBDecimal()]
        [PXDefault(TypeCode.Decimal, "0.0")]
        [PXUIField(DisplayName = "Price")]
        [PXFormula(null, typeof(SumCalc<RSSVWorkOrder.orderTotal>))]
        public virtual Decimal? BasePrice { get; set; }
        public abstract class basePrice :
PX.Data.BQL.BqlDecimal.Field<basePrice> { }
        #endregion

        // system fields
    }

```

- RSSVWorkOrderLabor: Specify the system attributes and other attributes as shown in the code fragments below:
 - For the DAC, define the following attribute.

```
[PXCacheName("Work Order Labor")]
```

- Configure the same attributes that have been configured for the corresponding fields of the RSSVLabor DAC except for the PXParent attribute, which you need to assign to the OrderNbr field.
- Make OrderNbr and InventoryID to be the key fields.



You need to make InventoryID a key field so that a user cannot specify multiple items with the same InventoryID.

The RSSVWorkOrderLabor DAC fields excluding the system fields should look as shown in the following code.

```

[PXCacheName("Work Order Labor")]
public class RSSVWorkOrderLabor : IBqlTable
{
    #region OrderNbr
    [PXDBString(15, IsKey = true, IsUnicode = true, InputMask = "")]
    [PXDBDefault(typeof(RSSVWorkOrder.orderNbr))]

```

```

[PXParent(typeof(SelectFrom<RSSVWorkOrder>.
    Where<RSSVWorkOrder.orderNbr.
IsEqual<RSSVWorkOrderLabor.orderNbr.FromCurrent>>))]
    public virtual string OrderNbr { get; set; }
    public abstract class orderNbr : PX.Data.BQL.BqlString.Field<orderNbr>
{ }

#endregion

#region InventoryID
[Inventory(IsKey = true)]
[PXRestrictor(typeof(Where<InventoryItem.stkItem, Equal<False>>),
    Messages.ItemIsStock)]
    public virtual int? InventoryID { get; set; }
    public abstract class inventoryID :
PX.Data.BQL.BqlInt.Field<inventoryID> { }
#endregion

#region DefaultPrice
[PXDBDecimal()]
[PXDefault(TypeCode.Decimal, "0.0")]
[PXUIField(DisplayName = "Default Price")]
    public virtual Decimal? DefaultPrice { get; set; }
    public abstract class defaultPrice :
PX.Data.BQL.BqlDecimal.Field<defaultPrice> { }
#endregion

#region Quantity
[PXDBDecimal()]
[PXDefault(TypeCode.Decimal, "0.0")]
[PXUIField(DisplayName = "Quantity")]
    public virtual Decimal? Quantity { get; set; }
    public abstract class quantity : PX.Data.BQL.BqlDecimal.Field<quantity>
{ }

#endregion

#region ExtPrice
[PXDBDecimal()]
[PXDefault(TypeCode.Decimal, "0.0")]
[PXUIField(DisplayName = "Ext. Price", Enabled = false)]
[PXFormula(
    typeof(Mult<RSSVWorkOrderLabor.quantity,
RSSVWorkOrderLabor.defaultPrice>),
    typeof(SumCalc<RSSVWorkOrder.orderTotal>))]
    public virtual Decimal? ExtPrice { get; set; }
    public abstract class extPrice : PX.Data.BQL.BqlDecimal.Field<extPrice>
{ }

#endregion

// system fields
}

```

3. Configure the `RSSVWorkOrderEntry` graph: Define data views in the generated graph and make the full list of standard system actions available by specifying the second generic type parameter in the base `PXGraph` class as the following code shows.

```

public class RSSVWorkOrderEntry : PXGraph<RSSVWorkOrderEntry, RSSVWorkOrder>
{


```

```
#region Views

//The primary view
public SelectFrom<RSSVWorkOrder>.View WorkOrders;

//The view for the Repair Items tab
public SelectFrom<RSSVWorkOrderItem>.
    Where<RSSVWorkOrderItem.orderNbr.
IsEqual<RSSVWorkOrder.orderNbr.FromCurrent>>.View
    RepairItems;

//The view for the Labor tab
public SelectFrom<RSSVWorkOrderLabor>.
    Where<RSSVWorkOrderLabor.orderNbr.
IsEqual<RSSVWorkOrder.orderNbr.FromCurrent>>.View
    Labor;
#endregion
}
```

4. Build the project in Visual Studio and publish the customization project.
5. Configure the RS301000.aspx page as follows: 
 - a. Use the following settings:
 - PrimaryView of the datasource control: WorkOrders
 - DataMember of the form control: WorkOrders
 - DataMember of the first grid control: RepairItems
 - DataMember of the second grid control: Labor
 - SkinID (in the PXGrid controls in ASPX): Details
 - Enabled in AutoSize (in the AutoSize control inside PXGrid in ASPX): True
 - Width (in the PXGrid controls in ASPX): 100%
 - b. Configure the controls on the form as follows:
 - Configure the controls for the **Repair Items** and **Labor** tabs in the same way as you did for the **Repair Items** and **Labor** tabs of the Services and Prices (RS203000) form. For details on the controls, see [Step 2.1.4: Creating Controls on the Form](#) and [Step 4.1.1: Adding the Labor Tab—Self-Guided Exercise](#) in the *T210 Customized Forms and Master-Detail Relationship* training course.
 - Do not configure controls for the Summary area of the form. You will configure them in [Step 1.1.2: Configuring the Controls of the Summary Area](#) of this lesson.
6. Publish the customization project.
7. Include a link to the form in the **Profiles** category of the **Phone Repair Shop** workspace.
8. Update the *SiteMapNode* item for the Repair Work Orders form in the customization project.

Step 1.1.2: Configuring the Controls of the Summary Area

In this step, you will configure the UI elements of the Summary area of the Repair Work Orders (RS301000) form, which are detailed in [Description of the Form Elements](#). To configure the UI elements, you will specify the attributes of the fields of the RSSVWorkOrder DAC and use the corresponding control types in the RS301000.aspx page. For details about control types, see [Input Controls](#) in the documentation.

You will configure the following controls for the elements of the form (The detail instruction is in the [Configuring Controls](#) section later in this topic):

- **Order Nbr.** (a selector with an input mask):
 - By using the `InputMask` property of the `PXDBString` attribute, you will require users to enter only digits in the box. For more information about input masks, see [To Configure an Input Mask and a Display Mask for a Field](#) in the documentation.
 - You will specify that the input value cannot be null and cannot contain only spaces by setting the `PersistingCheck` property of the `PXDefault` attribute to `PXPersistingCheck.NullOrBlank`.
 - In the `PXSelector` attribute, you will not specify any fields to be displayed as columns of the selector. Instead, you will use the `Visibility` property of the `PXUIField` attribute of the `OrderNbr`, `Description`, `ServiceID`, and `DeviceID` fields to include these fields in the selector of the `OrderNbr` field.
 - You will use the `PXSelector` control in ASPX.
- **Customer ID** (a selector for a segmented key value):
 - By using the `CustomerActive` attribute, defined in the `PX.Objects.AR` namespace, you will configure a selector control for the box. This attribute selects only the customer records that have the *Active* or *One-Time* status.



You can find the attribute that can be used for a selector control that retrieves the data from an Acumatica ERP database table by investigating the source code of a similar selector. For example, if you want to find an attribute that can be used with the customer selector in the Summary area of a document, you can investigate the attributes assigned to the `ARInvoice.CustomerID` field, which is displayed as the **Customer** box on the [Invoices and Memos](#) (AR301000) form.

- You will use the `PXSegmentMask` control in ASPX. You use the `PXSegmentMask` control instead of the `PXSelector` control if you need to configure a selector control for a segmented key value.
- **Assignee** (a selector):
 - By using the `Owner` attribute, defined in the `PX.TM` namespace, you will configure a selector control for the box. This attribute shows the list of employees.



To find an attribute that can be used with the employee selector in the Summary area of a document, you can investigate the attributes assigned to the `CR.Contact.OwnerID` field, which is displayed as the **Owner** box on the [Leads](#) (CR301000) form.

- You will use the `PXSelector` control in ASPX.
- **Date Created** and **Date Completed** (date-and-time controls):
 - By using the `PXDBDateTime` and its `DisplayMask` and `InputMask` properties, you will configure the controls to display the date in the month/day pattern and require users to enter only the date in the control.
 - You will set the default value for **Date Created** as the current business date, which is stored in the `AccessInfo.BusinessDate` system field.
 - You will use the `PXDateTimeEdit` control in ASPX.
- **Status** and **Priority** (drop-down lists):
 - By using the `PXStringList` attribute, you will configure drop-down lists for these boxes. You will use localizable names for the list items.
 - You will use the `PXDropDown` control in ASPX.
- **Order Total** (a box with a decimal number):
 - You will set the default value of the field by using the `PXDefault` attribute.
 - You will make this field read-only by setting the `Enabled` property of the `PXUIField` attribute to `false`.
 - You will use the `PXNumberEdit` control in ASPX.

- **Invoice Nbr.** (a box with the invoice number):
 - You will make this field read-only by setting the `Enabled` property of the `PXUIField` attribute to `false`.
 - You will use the `PXTextEdit` control in ASPX. You will change the control for this box in the *T230 Actions* training course.

The `RSSVWorkOrder` database table also contains the `Hold` field. The `Hold` field is a legacy field which was used in the previous version of the course before the workflow has been implemented for the form. However, we recommended that you have this field in the database table in case you plan to use it alongside the workflow, for example to trigger a transition from the *On Hold* status. Now you only need to set the default value of the field by using the `PXDefault` attribute.

Configuring Controls

Do the following:

1. In the `RSSVWorkOrder` DAC, configure the attributes of the `OrderNbr` field and the fields included in the selector, as shown in the following code:

- For the `OrderNbr` field

```
#region OrderNbr
[PXDBString(15, IsKey = true, IsUnicode = true, InputMask =
">CCCCCCCCCCCCCCC")]
[PXDefault(PersistingCheck = PXPersistingCheck.NullOrBlank)]
[PXUIField(DisplayName = "Order Nbr.", Visibility =
PXUIVisibility.SelectorVisible)]
[PXSelector(typeof(Search<RSSVWorkOrder.orderNbr>))]
public virtual string OrderNbr { get; set; }
public abstract class orderNbr : PX.Data.BQL.BqlString.Field<orderNbr> { }
#endregion
```

- For the `Description` field

```
#region Description
[PXDBString(60, IsUnicode = true)]
[PXUIField(DisplayName = "Description", Visibility =
PXUIVisibility.SelectorVisible)]
public virtual string Description { get; set; }
public abstract class description :
PX.Data.BQL.BqlString.Field<description> { }
#endregion
```

- For the `DeviceID` field

```
#region DeviceID
[PXDBInt()]
[PXDefault]
[PXUIField(DisplayName = "Device", Visibility =
PXUIVisibility.SelectorVisible)]
[PXSelector(typeof(Search<RSSVDevice.deviceID>),
typeof(RSSVDevice.deviceCD),
typeof(RSSVDevice.description),
SubstituteKey = typeof(RSSVDevice.deviceCD),
DescriptionField = typeof(RSSVDevice.description))]
public virtual int? DeviceID { get; set; }
public abstract class deviceID : PX.Data.BQL.BqlInt.Field<deviceID> { }
#endregion
```

- For the ServiceID field

```
#region ServiceID
[PXDBInt()]
[PXDefault]
[PXUIField(DisplayName = "Service", Visibility =
PXUIVisibility.SelectorVisible)]
[PXSelector(typeof(Search<RSSVRepairService.serviceID>),
typeof(RSSVRepairService.serviceCD),
typeof(RSSVRepairService.description),
SubstituteKey = typeof(RSSVRepairService.serviceCD),
DescriptionField = typeof(RSSVRepairService.description))]
public virtual int? ServiceID { get; set; }
public abstract class serviceID : PX.Data.BQL.BqlInt.Field<serviceID> { }
#endregion
```

2. Configure the attributes of the CustomerID and Assignee fields as follows:

- a. In the RSSVWorkOrder.cs file, add the using directives as follows.

```
using PX.Objects.AR;
using PX.TM;
```

- b. For the CustomerID field, specify the attributes as follows.

```
#region CustomerID
[PXDefault]
[CustomerActive(DisplayName = "Customer ID", DescriptionField =
typeof(Customer.acctName))]
public virtual int? CustomerID { get; set; }
public abstract class customerID : PX.Data.BQL.BqlInt.Field<customerID> { }
#endregion
```

- c. For the Assignee field, specify the attributes as follows.

```
#region Assignee
[Owner(DisplayName = "Assignee")]
public virtual int? Assignee { get; set; }
public abstract class assignee : PX.Data.BQL.BqlInt.Field<assignee> { }
#endregion
```

3. Configure the attributes of the DateCreated and DateCompleted fields, as shown in the following code:

- For the DateCreated field

```
#region DateCreated
[PXDBDate()]
[PXDefault(typeof(AccessInfo.businessDate))]
[PXUIField(DisplayName = "Date Created")]
public virtual DateTime? DateCreated { get; set; }
public abstract class dateCreated :
PX.Data.BQL.BqlDateTime.Field<dateCreated> { }
#endregion
```

- For the DateCompleted field

```
#region DateCompleted
[PXDBDate()]
[PXUIField(DisplayName = "Date Completed", Enabled = false)]
```

```

        public virtual DateTime? DateCompleted { get; set; }
        public abstract class dateCompleted :
PX.Data.BQL.BqlDateTime.Field<dateCompleted> { }
        #endregion

```

4. Configure the attributes of the **Status** and **Priority** fields as follows:

- a. In the **Messages** class, define the constants to be used in the **Status** box, as shown in the following code.

```

//Work order statuses
public const string OnHold = "On Hold";
public const string PendingPayment = "Pending Payment";
public const string ReadyForAssignment = "Ready for Assignment";
public const string Assigned = "Assigned";
public const string Completed = "Completed";
public const string Paid = "Paid";

```

- b. Make sure the constants are already defined for the **Priority** box, as shown in the following code.

```

//Complexity of repair and work order priorities
public const string High = "High";
public const string Medium = "Medium";
public const string Low = "Low";

```

- c. In the **Constants.cs** file, define the constants for the **Priority** box, as shown in the following code.

```

//Constants for the priority of repair work orders
public static class WorkOrderPriorityConstants
{
    public const string High = "H";
    public const string Medium = "M";
    public const string Low = "L";
}

```

- d. Define the constants for the **Status** box, as shown in the following code.

```

//Constants for the statuses of repair work orders
public static class WorkOrderStatusConstants
{
    public const string OnHold = "OH";
    public const string PendingPayment = "PP";
    public const string ReadyForAssignment = "RA";
    public const string Assigned = "AS";
    public const string Completed = "CM";
    public const string Paid = "PD";
}

```

- e. For the **Status** field, specify the attributes as follows.

```

#region Status
[PXDBString(2, IsFixed = true)]
[PXDefault(WorkOrderStatusConstants.OnHold)]
[PXUIField(DisplayName = "Status", Enabled = false)]
[PXStringList(
    new string[]
    {
        WorkOrderStatusConstants.OnHold,
        WorkOrderStatusConstants.PendingPayment,

```

```

        WorkOrderStatusConstants.ReadyForAssignment,
        WorkOrderStatusConstants.Assigned,
        WorkOrderStatusConstants.Completed,
        WorkOrderStatusConstants.Paid
    },
    new string[]
    {
        Messages.OnHold,
        Messages.PendingPayment,
        Messages.ReadyForAssignment,
        Messages.Assigned,
        Messages.Completed,
        Messages.Paid
    }
    )}]
public virtual string Status { get; set; }
public abstract class status : PX.Data.BQL.BqlString.Field<status> { }
#endregion

```

f. For the **Priority** field, specify the attributes as follows.

```

#region Priority
[PXDBString(1, IsFixed = true)]
[PXDefault(WorkOrderPriorityConstants.Medium)]
[PXUIField(DisplayName = "Priority")]
[PXStringList(
    new string[]
    {
        WorkOrderPriorityConstants.High,
        WorkOrderPriorityConstants.Medium,
        WorkOrderPriorityConstants.Low
    },
    new string[]
    {
        Messages.High,
        Messages.Medium,
        Messages.Low
    }
    )}]
public virtual string Priority { get; set; }
public abstract class priority : PX.Data.BQL.BqlString.Field<priority> { }
#endregion

```

5. For the **Hold** field, add the following attribute.

```

#region Hold
[PXDBBool()]
[PXDefault(true)]
public virtual bool? Hold { get; set; }
public abstract class hold : PX.Data.BQL.BqlBool.Field<hold> { }
#endregion

```

6. Define the attributes of the **OrderTotal** field, as the following code shows.

```

#region OrderTotal
[PXDBDecimal()]
[PXDefault(TypeCode.Decimal, "0.0")]
[PXUIField(DisplayName = "Order Total", Enabled = false)]
public virtual Decimal? OrderTotal { get; set; }

```



```

    public abstract class orderTotal : PX.Data.BQL.BqlDecimal.Field<orderTotal>
    { }

    #endregion

```

7. Define the attributes of the InvoiceNbr field as follows.

```

#region InvoiceNbr
[PXDBString(15, IsUnicode = true)]
[PXUIField(DisplayName = "Invoice Nbr.", Enabled = false)]
public virtual string InvoiceNbr { get; set; }
public abstract class invoiceNbr : PX.Data.BQL.BqlString.Field<invoiceNbr> { }
#endregion

```

8. Build the project.

9. Create controls for all fields of the RSSVWorkOrder DAC except the system fields.



You can create controls by using the Screen Editor of the Customization Project Editor or by editing the ASPX code of the form directly in Visual Studio.

Related Links

- [Input Controls](#)
- [Complex Input Controls](#)
- [To Configure an Input Mask and a Display Mask for a Field](#)
- [PXSelectorAttribute Class](#)
- [PXDBDateAndTimeAttribute Class](#)
- [PXStringListAttribute Class](#)
- [Box \(Control for a Data Field\)](#)

Step 1.1.3: Configuring the Layout of the Summary Area of the Form

In this step, you will organize the layout of the Summary area of the Repair Work Orders (RS301000) form to the specifications in [Layout of the Form](#). You will do the following:

- By adding the layout rules (PXLayoutRule) with the StartRow = "True" and StartColumn = "True" properties to the form, you will define three columns of controls on the page.

When you create any form from a template, the form already contains the PXLayoutRule element with StartRow = "True". You should keep this element for the proper layout.

- By using the ControlSize and LabelsWidth properties of PXLayoutRule, you will configure the size of controls and labels in the Summary area of the form.

You need to adjust the size of the controls and labels in columns. Typically, you specify the sizes in the layout rules with StartRow = "True" and StartColumn = "True". You can also specify specific sizes for a particular control.



The PXDateTimeEdit and PXNumberEdit control types have a predefined Width property value, which you cannot change by setting the ColumnWidth or ControlSize property values for the appropriate PXLayoutRule component. You can specify the size of the controls of these types by using the Size property.

- By using the ColumnSpan property of PXLayoutRule, you will expand the Description field to span two columns of controls.

The ColumnSpan property specifies the number of columns that the next control spans. The ColumnSpan property affects only the control that follows the layout rule.

The basic approach to designing a form is to start any form layout with a layout rule with `StartRow = "True"`; this layout rule starts the first column of controls and adds proper margins between controls on the form. Then you can add as many layout rules as you need. You can specify the size of controls and labels in layout rules or define these sizes directly in the control properties. Unless the size is overridden in a control, the size that is specified in a layout rule is applied to all controls that follow this rule until the next layout rule.



To configure the layout of the form, you can use the Screen Editor of the Customization Project Editor or edit the ASPX code of the form directly in Visual Studio. The addition of controls with the Screen Editor and with Visual Studio is described in [Step 1.5.1: Add Columns to the Grid](#) and [Step 2.3.1: Add Input Controls](#) (respectively) of the *T200 Maintenance Forms* training course. The instructions below are presented in general terms to accommodate both methods.

Configuring the Layout of the Summary Area

Complete the following steps:

1. In the Screen Editor for the Repair Work Orders form or in the ASPX code of the `Pages\RS\RS301000.aspx` file of the site, reorder the fields in the Summary area of the form (PXFormView control) so that they are located in the following order:

- Order Nbr.
- Status
- Date Created
- Date Completed
- Priority
- Customer ID
- Service
- Device
- Assignee
- Description
- Order Total
- Invoice Nbr.

2. Set the `CommitChanges` of the `Customer ID` control to `True`.

It will allow the system to retrieve and display the customer name along with the customer ID when the customer is selected.

3. Under the `Priority` and `Description` fields, add the layout rules with `StartColumn = "True"` (which are the **Column** controls in the Screen Editor). The following ASPX code shows an example of the layout rule with `StartColumn = "True"`.

```
<px:PXLayoutRule runat="server" ID="CstPXLayoutRule16"
  StartColumn="True" ControlSize="XM" LabelsWidth="S" ></px:PXLayoutRule>
```

4. Set the `ControlSize` property value for layout rules of the form control as follows:

- For the first layout rule, `SM`
- For the second layout rule, `XM`
- For the third layout rule, `M`

5. Set the `LabelsWidth` property value to `S` for all layout rules of the form control.

6. Add an empty layout rule (which is the **Empty Rule** control in the Screen Editor) before the control for the `Description` field. Set the `ColumnSpan` property for the layout rule to 2. The following ASPX code shows the layout rule with `ColumnSpan = "2"`.

```

    <px:PXLayoutRule runat="server" ID="CstLayoutRule18" ColumnSpan="2" >
</px:PXLayoutRule>
    <px:PXTextEdit runat="server" ID="CstPXTextEdit7" DataField="Description" >
</px:PXTextEdit>

```

7. Clear the Height property of the PXFormView control.



PXFormView control automatically calculate their Height based on the size of visible controls that they contain.

8. Save your changes. The resulting definition of the Summary area should look as the following code shows.

```

<px:PXFormView ID="form" runat="server" DataSourceID="ds" DataMember="WorkOrders"
Width="100%" Height="" AllowAutoHide="false">
  <Template>
    <px:PXLayoutRule ControlSize="SM" LabelsWidth="S" ID="PXLayoutRule1"
      runat="server" StartRow="True"></px:PXLayoutRule>
    <px:PXSelector runat="server" ID="CstPXSelector11" DataField="OrderNbr" >
</px:PXSelector>
    <px:PXDropDown runat="server" ID="CstPXDropDown20" DataField="Status" >
</px:PXDropDown>
    <px:PXDateTimeEdit runat="server" ID="CstPXDateTimeEdit6" DataField="DateCreated"
  >
</px:PXDateTimeEdit>
    <px:PXDateTimeEdit runat="server" ID="CstPXDateTimeEdit5"
DataField="DateCompleted" >
</px:PXDateTimeEdit>
    <px:PXDropDown runat="server" ID="CstPXDropDown13" DataField="Priority" >
</px:PXDropDown>
    <px:PXLayoutRule runat="server" ID="CstPXLayoutRule16"
      StartColumn="True" ControlSize="XM" LabelsWidth="S" ></px:PXLayoutRule>
    <px:PXSegmentMask CommitChanges="True" runat="server"
      ID="CstPXSegmentMask4" DataField="CustomerID" >
</px:PXSegmentMask>
    <px:PXSelector runat="server" ID="CstPXSelector14" DataField="ServiceID" >
</px:PXSelector>
    <px:PXSelector runat="server" ID="CstPXSelector8" DataField="DeviceID" >
</px:PXSelector>
    <px:PXSelector runat="server" ID="CstPXSelector3" DataField="Assignee" >
</px:PXSelector>
    <px:PXLayoutRule runat="server" ID="CstLayoutRule18" ColumnSpan="2" >
</px:PXLayoutRule>
    <px:PXTextEdit runat="server" ID="CstPXTextEdit7" DataField="Description" >
</px:PXTextEdit>
    <px:PXLayoutRule runat="server" ID="CstPXLayoutRule17"
      StartColumn="True" ControlSize="M" LabelsWidth="S" >
    </px:PXLayoutRule>
    <px:PXNumberEdit runat="server" ID="CstPXNumberEdit12" DataField="OrderTotal" >
</px:PXNumberEdit>
    <px:PXTextEdit runat="server" ID="CstPXTextEdit10" DataField="InvoiceNbr" >
</px:PXTextEdit>
  </Template>
</px:PXFormView>

```

9. Publish the customization project.

10. In the Summary area of the Repair Work Orders form (shown in the following screenshot), make sure all of the following conditions are met:

- The controls are arranged into three columns.
- The **Description** box is expanded across two columns.
- The labels of all controls are fully displayed.

Figure: The layout of the Summary area

Related Links

- [Layout Rule \(PXLayoutRule\)](#)
- [Use of the StartRow and StartColumn Properties of PXLayoutRule](#)
- [Use of the ColumnWidth, ControlSize, and LabelsWidth Properties of PXLayoutRule](#)
- [Use of the ColumnSpan Property of PXLayoutRule](#)
- [Predefined Size Values](#)

Step 1.1.4: Configuring Form View Mode for the Grid

In this example, you will configure form view mode for the **Repair Items** tab of the Repair Work Orders (RS301000) form; this mode is shown in the following screenshot. The mode provides the capability to edit a single record that has been selected in the grid. If a user selects a record in the grid and then clicks the **Switch Between Grid and Form** button on the table toolbar, the selected record is displayed in the form view.

You will arrange controls in the form view of the grid by using layout rules. You will add controls inside the RowTemplate ASPX element and organize them into two groups by using the PXLayoutRule controls with the GroupCaption and StartGroup properties specified. The order of controls in form view mode may differ from the order of columns in grid mode. You will also use the AllowFormEdit property of Mode of PXGrid to make form view mode available and the SyncPosition of PXGrid to synchronize the values displayed in grid mode and in form view mode.

Figure: Form view mode

Configuring Form View Mode for the Repair Items Tab

Do the following to configure form view mode for the grid on the **Repair Items** tab:

1. In the Screen Editor for the Repair Work Orders form or in the ASPX code of the `Pages\RS\RS301000.aspx` file of the site, for the `PXGrid` control, set the `AllowFormEdit` property of `Mode` to `True`.



In the Screen Editor, select the **Grid: RepairItems** node, and in the **Mode** group of layout properties, set the `AllowFormEdit` property. In Visual Studio, add `<Mode AllowFormEdit="True" ></Mode>` inside the `PXGrid` element, and add the `RowTemplate` element inside the `PXGridLevel` element.

2. For the `PXGrid` control, set the `SyncPosition` property to `True` to display in form view mode the row that is selected in the grid.
3. For the form view mode, add a layout rule with `StartRow = "True"`.



To add a layout rule for the form view mode in the Screen Editor, you need to select the **Tab > Repair Items > Grid: RepairItems > Levels > RepairItems** node and create row on the **Add Controls** tab. To create a layout rule for form view mode in Visual Studio, you need to add the layout rule inside the `RowTemplate` element.

4. Create the controls in form view mode for the following fields and organize them in the following order:
 - `RepairItemType`
 - `InventoryID`
 - `InventoryID_description`
 - `BasePrice`



To create controls for the form view mode in the Screen Editor, you need to select the **Tab > Repair Items > Grid: RepairItems > Levels > RepairItems** node and create controls on the **Add Data Fields** tab. To create controls for the form view mode in Visual Studio, you need to add controls inside the `RowTemplate` element.

5. Configure form view mode of the grid as follows:
 - a. Add a group layout rule (`PXLayoutRule` with `StartGroup="True"`) before the `Price` field and before the `RepairItemType` field.
 - b. Specify the group captions in the `GroupCaption` property of the group layout rules:
 - For the first group, `Repair Item`
 - For the second group, `Price Info`
 - c. Enter the following properties for the first group layout rule:
 - `ControlSize:M`
 - `LabelsWidth:SM`
 - d. Enter the `S` value for the `LabelsWidth` property for the second group layout rule.
 - e. Specify `StartColumn = "True"` for the second group layout rule.

The following code shows the ASPX of the form view of the **Repair Items** tab.

```
<RowTemplate>
  <px:PXLayoutRule runat="server" ID="CstPXLayoutRule21" StartRow="True" >
    </px:PXLayoutRule>
  <px:PXLayoutRule ControlSize="M" LabelsWidth="SM" runat="server"
    ID="CstPXLayoutRule26" StartGroup="True" GroupCaption="Repair Item" >
    </px:PXLayoutRule>
  <px:PXDropDown runat="server" ID="CstPXDropDown25" DataField="RepairItemType" >
    </px:PXDropDown>
  <px:PXSegmentMask runat="server" ID="CstPXSegmentMask23" DataField="InventoryID" >
    </px:PXSegmentMask>
  <px:PXTextEdit runat="server" ID="CstPXTextEdit24"
    DataField="InventoryID_description" >
    </px:PXTextEdit>
  <px:PXLayoutRule LabelsWidth="S" StartColumn="True" GroupCaption="Price Info"
    runat="server" ID="CstPXLayoutRule27" StartGroup="True" ></px:PXLayoutRule>
  <px:PXNumberEdit runat="server" ID="CstPXNumberEdit22" DataField="BasePrice" >
    </px:PXNumberEdit>
</RowTemplate>
```

6. Save your changes.
7. Publish the customization project.
8. On the toolbar of the **Repair Items** tab of the Repair Work Orders form, click the **Switch Between Grid and Form** button, and review the layout of form view mode. It should look as shown in the following screenshot.

Figure: The layout of the form view mode

Related Links

- [Configuration of Grids](#)
- [Use of the GroupCaption, StartGroup, and EndGroup Properties of PXLLayoutRule](#)

Step 1.1.5: Adding the Substitute Form with a Shared Filter to the Project



In this step, you will import the substitute form that has been developed for the Repair Work Orders (RS301000) form, which is the related entry form.

A shared filter has been developed for and applied to this substitute form. The shared filter displays the repair work orders that have the *Ready for Assignment* status. (For details about shared filters, see [Saving of Filters for Future Use](#).)

You will then add the generic inquiry for the substitute form and the shared filter to the customization project.

Importing the Substitute Form and the Shared Filter

Do the following:

1. On the [Generic Inquiry](#) (SM208000) form, import the generic inquiry from the `RepairWorkOrders.xml` file provided with this course. The file contains the definition of the generic inquiry and the shared filter.
2. On the **Entry Point** tab, in the **Entry Screen** box, select the Repair Work Orders (RS301000) form. Once you selected the form, the **Replace Entry Screen with This Inquiry in Menu** check box becomes selected.
3. On the **Entry Point** tab, make sure the **Enable New Record Creation** check box is selected.
4. Save your changes.
5. On the Repair Work Orders form, make sure the substitute form with two tabs **All Records** and **To Assign** (shown below) is displayed when you open the form.

When the *To Assign* filter was saved, the system added the corresponding tab (with the name of the filter) to the substitute form. The **All Records** tab shows all records without any filter applied.

Repair Work Orders ☆

CUSTOMIZATION ▾ TOOLS ▾

↻ ↶ + ✎ ⏮ ⏭

ALL RECORDS **TO ASSIGN**

Status: = Ready for Assignment ▾

Order Nbr. Description Status Service Device

No records found as 'To Assign'.
Try to change filter to see records here.

< < > >

Figure: Substitute form

6. On the Generic Inquiries page of the Customization Project Editor, add the generic inquiry to the *PhoneRepairShop* customization project.
7. On the Shared Filters page of the Customization Project Editor, add the shared filter to the project as follows:
 - a. On the toolbar of the page, click **Add New Record**.
 - b. In the **Add Shared Filter** dialog box, which opens, select the check box in the **Selected** column for the *To Assign* filter.
 - c. Click **OK**.
 - d. Click **Save** on the page toolbar.

Related Links

- [Saving of Filters for Future Use](#)

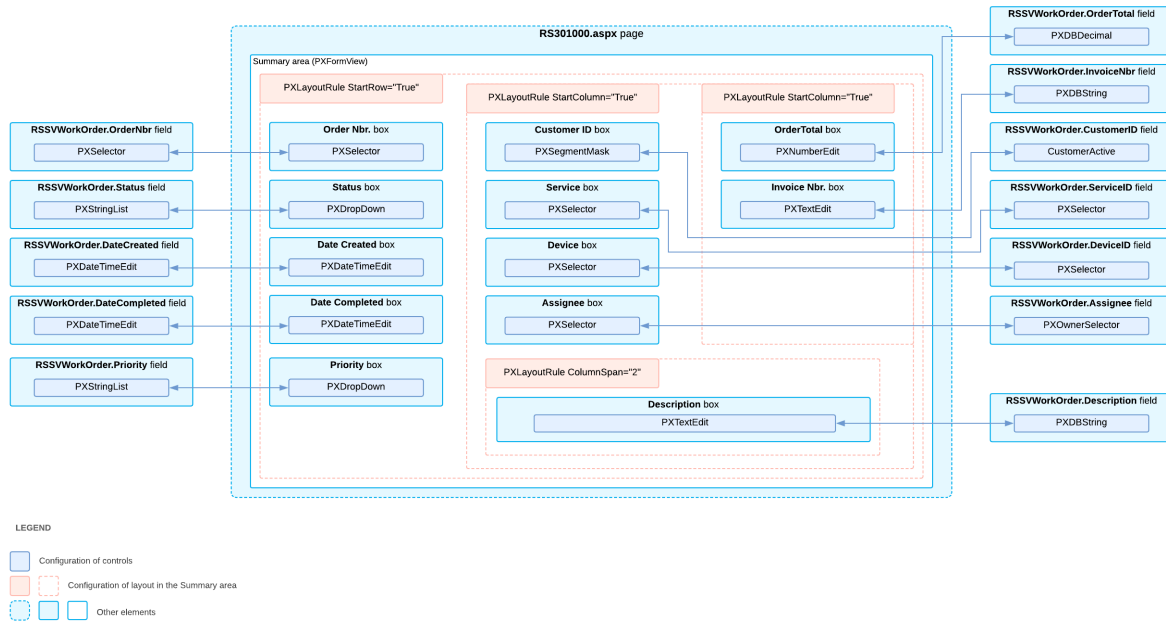
Lesson Summary

In this lesson, you have learned how to configure the layout of a form. You have used the `PXLayoutRule` controls added to ASPX page source code to configure the layout of the Repair Work Orders (RS301000) form as follows:

- Define rows of controls on the form and divide the controls into columns in each row
- Expand UI controls over multiple columns
- Adjust the sizes of controls and their labels
- Group UI controls on a form

The following diagram shows the elements that you have implemented for the configuration of the controls and the layout of the Repair Work Orders form.

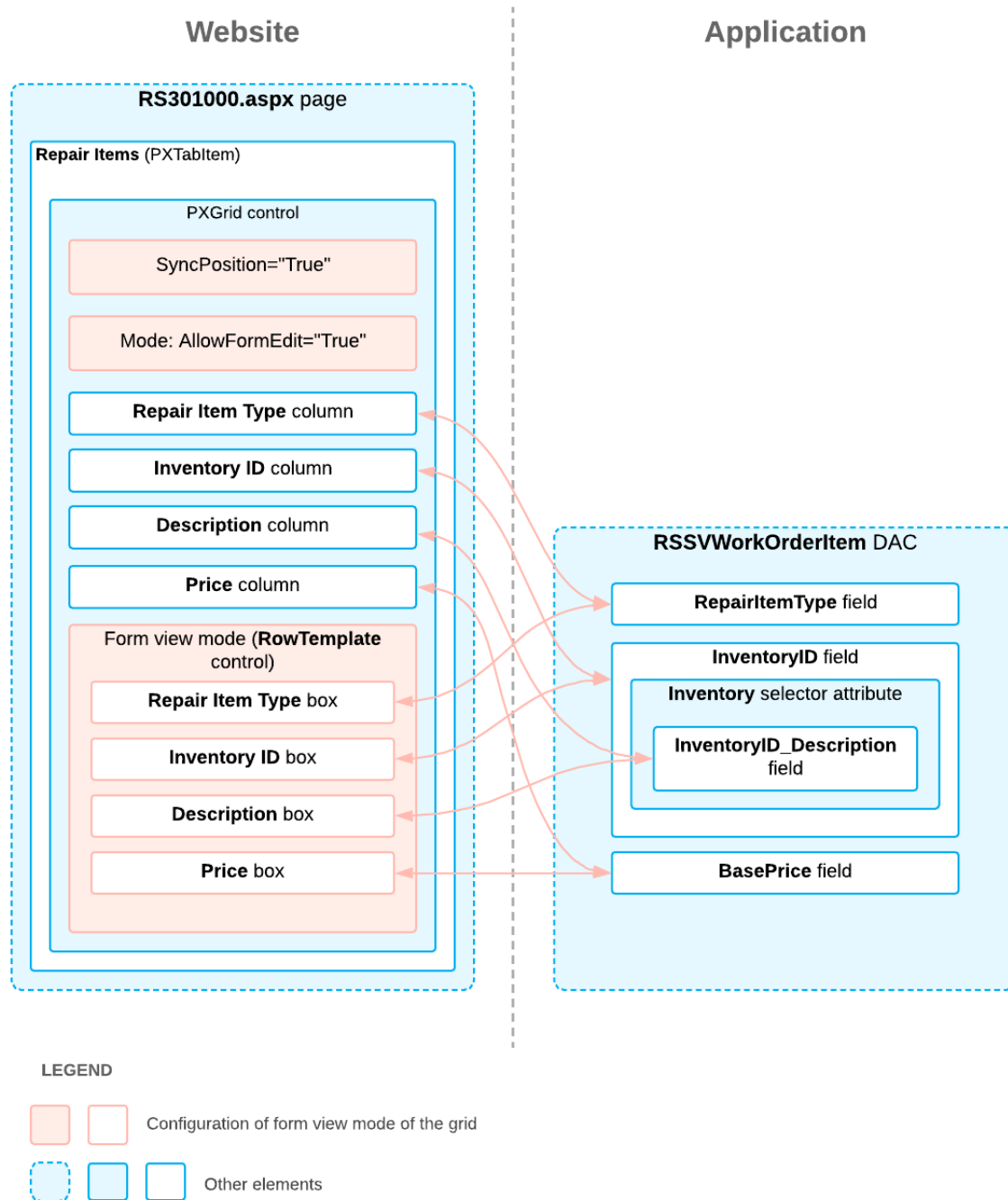
Configuration of Layout and Controls



You have also configured a grid by adding columns to it and configuring controls for the grid columns in form view mode. You have added input controls to form view of the grid, set their properties, and arranged them, as you would in any other area of the form that contains a form view (such as the Summary area or a form tab).

The elements that you have added to configure the form view mode of the grid are shown in the following diagram.

Configuration of Form View Mode of the Grid



You have added the substitute form, which has a shared filter, for the custom entry form and included this substitute form in the customization project.

Review Questions

1. Which property of the `PXLayoutRule` control would you use to divide the controls on a form into two columns?
 - a. `StartColumn`
 - b. `StartRow`
 - c. `StartGroup`
2. Which attribute would you assign to a DAC field to configure a drop-down list with `string` values?
 - a. `PXIntList`
 - b. `PXStringList`
 - c. `PXDropDown`

Answer Key

1. a
2. b

Additional Information: Configuration of Controls

In this lesson, you have configured various types of controls on the Repair Work Orders (RS301000) form. However, particular scenarios of the configuration of controls, such as modification of a drop-down list at runtime and replacement of the displayed key value at runtime, are outside of the scope of the course but may be useful to some readers.

Modifying a Drop-Down List in Runtime

You can modify a drop-down list at runtime by using the `SetList<>()` static method of the `PXStringList` attribute. You can do this in the `RowSelected` event handler or graph constructor.

For details about this and other options to configure a drop-down list, see [Configuration of Drop-Down Lists](#) in the documentation.

Replacing the Displayed Key Value

The `SubstituteKey` property specifies the field whose value should be shown in the control in the UI instead of the field that is specified in the `Search<>` command.

For details about the configuration of selectors in code, see [Configuration of Selector Controls](#). For more information about how to change the external presentation in run time, see [Internal and External Presentation of Values](#).

Replacing Attributes of DAC Fields in CacheAttached

The attributes that you add to a data field in the DAC are initialized once, during the startup of the domain. You can replace attributes for a particular field by defining the `CacheAttached` event handler for this field in a graph. These attributes are also initialized once, on the first initialization of the graph where you define this method.

For details about replacement of attributes of DAC fields, see [Replacement of Attributes for DAC Fields in CacheAttached](#).

Additional Information: Layout Configuration

In this lesson, you have learned how to configure the layout of the form. Two scenarios of layout configuration, aligning controls horizontally and hiding the labels of controls, are outside of the scope of this course but may be useful to some readers.

Horizontal Alignment of Controls

Horizontal alignment is performed for the controls that are placed between a layout rule with the `Merge` property set to `True` and the next layout rule. Therefore, to cancel merging for all of the following controls, you have to add a `PXLayoutRule` component with or without the `Merge` property specified.

For more information about the `Merge` property, see [Use of the Merge Property of PXLayoutRule](#).

Hiding of Labels of Controls

To hide the labels of the controls placed within a column, you should set the `SuppressLabel` property value of the `PXLayoutRule` component of the column to `True`. Then within the column, all check boxes are placed without any space to the left of the input control, and the labels of other controls are hidden.

For details about the `SuppressLabel` property, see [Use of the SuppressLabel Property of PXLayoutRule](#).

Lesson 1.2: Copying Field Values from One Record to Another

In this lesson, for the Repair Work Orders (RS301000) form, you will implement the business logic that includes the following requirements:

- When values have been selected in the **Service** and **Device** boxes of the Summary area of the form, these values correspond to those in a record entered on the Services and Prices (RS203000) form, and no records have been added on the **Repair Items** and **Labor** tabs, the default records that have been configured on the Services and Prices form are inserted on the tabs.
- For a particular row, if a value is selected in the **Inventory ID** column, the values in the **Repair Item Type** and **Price** columns will be changed to the repair item type and base price (respectively) of the selected stock item as specified on the [Stock Items](#) (IN202500) form. (This business logic replicates the business logic that have been defined for the **Repair Items** tab of the Services and Prices form. You will implement this logic on your own.)

Lesson Objectives

You will learn how to copy the field values from one record to another record by using event handlers.

Step 1.2.1: Creating a Work Order from a Template (with RowUpdated)

Managers of the Smart Fix company define the prices for particular services and devices on the Services and Prices (RS203000) form. These prices are used as templates for the creation of work orders on the Repair Work Orders (RS301000) form. When a customer comes to an office of the Smart Fix company to repair a phone, a consultant in the office creates a repair work order on the Repair Work Orders (RS301000) form. When the consultant selects a

particular service and device for a new repair work order, the default information about the price for the service, which a manager has entered on the Services and Prices form, should be copied to the new work order.

In this step, you will implement the copying of the default records configured on the Services and Prices (RS203000) form to the **Repair Items** and **Labor** tabs of the Repair Work Orders (RS301000) form. The default record or records should be inserted when a user creates a new work order on the form and the following criteria are met:

- In the **Service** and **Device** boxes of the Summary area of the form, the user selects the values for which a record has been defined on the Services and Prices form.
- No records have been added on the **Repair Items** and **Labor** tabs.

Because you need to insert details on the Repair Work Orders form when values of two fields of one record are specified, you will implement the `RowUpdated` event handler for the `RSSVWorkOrder` DAC. In this handler, you will do the following:

- To check that a new `RSSVWorkOrder` record has been inserted, you will use the `Cache.GetStatus` method of the `WorkOrders` data view. An inserted record maintains the `Inserted` status until it is saved to the database even if it has been updated. When the inserted record is deleted, it is assigned the specific `InsertedDeleted` status.
- To display the data records in the UI, you will insert them in `PXCache` by using the `Insert` method of the `RepairItems` data view. The `Insert` method is invoked on the `PXCache` object of the first DAC specified in the data view type (which is the main DAC of the data view). Once you inserted the record, it has the default values specified for the fields.



In Acumatica ERP, before a new record can be inserted in `PXCache`, it is required that all key fields of this record are assigned some values. Since both `OrderNbr` and `InventoryID` fields must be the key fields for the `RSSVWorkOrderLabor` DAC, you need to assign a value to the `InventoryID` field. You do not need to explicitly assign a value to the `OrderNbr` field because it has `PXDBDefault` attached to it. The `PXDBDefault` attribute is responsible for generating and assigning a default value when the `FieldDefaulting` event is raised for this field, which always happens before the record is inserted in `PXCache`.

- Because you update the values of the inserted records, you need to call the `Update` method of `PXCache` to trigger all the events related to the update of the fields of a detail record.

For details about modifications in `PXCache`, see [Modification of Data in a PXCache Object](#) in the documentation.

Inserting the Default Repair Items and Labor to the Tabs

Do the following:

1. Add the following `RowUpdated` event handler to the `RSSVWorkOrderEntry` graph.

```
#region Events
//Copy repair items and labor items from the Services and Prices form.
protected virtual void _(Events.RowUpdated<RSSVWorkOrder> e)
{
    if (WorkOrders.Cache.GetStatus(e.Row) == PXEntryStatus.Inserted &&
        !e.Cache.ObjectsEqual<RSSVWorkOrder.serviceID, RSSVWorkOrder.deviceID>(e.Row,
        e.OldRow))
    {
        if (e.Row.ServiceID != null && e.Row.DeviceID != null &&
            !IsCopyPasteContext && RepairItems.Select().Count == 0 &&
            Labor.Select().Count == 0)
        {
            //Retrieve the default repair items
            var repairItems = SelectFrom<RSSVRepairItem>.
                Where<RSSVRepairItem.serviceID.IsEqual<RSSVWorkOrder.serviceID.FromCurrent>.
```

```

And<RSSVRepairItem.deviceID.IsEqual<RSSVWorkOrder.deviceID.FromCurrent>>>
    .View.Select(this);
//Insert default repair items
foreach (RSSVRepairItem item in repairItems)
{
    RSSVWorkOrderItem orderItem = RepairItems.Insert();
    orderItem.RepairItemType = item.RepairItemType;
    orderItem.InventoryID = item.InventoryID;
    orderItem.BasePrice = item.BasePrice;
    RepairItems.Update(orderItem);
}

//Retrieve the default labor items
var laborItems = SelectFrom<RSSVLabor>.

Where<RSSVLabor.serviceID.IsEqual<RSSVWorkOrder.serviceID.FromCurrent>>.

And<RSSVLabor.deviceID.IsEqual<RSSVWorkOrder.deviceID.FromCurrent>>>
    .View.Select(this);
//Insert the default labor items
foreach (RSSVLabor item in laborItems)
{
    RSSVWorkOrderLabor orderItem = new RSSVWorkOrderLabor();
    orderItem.InventoryID = item.InventoryID;
    orderItem = Labor.Insert(orderItem);
    orderItem.DefaultPrice = item.DefaultPrice;
    orderItem.Quantity = item.Quantity;
    orderItem.ExtPrice = item.ExtPrice;
    Labor.Update(orderItem);
}
}
}
}
}
#endregion

```

2. Rebuild the project.
3. In the RS301000.aspx file or in the Screen Editor, for the ServiceID and DeviceID fields, set CommitChanges to True.
4. Save your changes.
5. Publish the customization project.

Testing the Logic

On the Repair Work Orders form, do the following:

1. Create a work order with the following settings:
 - **Order Nbr.:** 000001
 - **Customer ID:** C000000001
 - **Service:** Battery Replacement
 - **Device:** Nokia 3310
 - **Description:** Battery replacement, Nokia 3310

During the *T210 Customized Forms and Master-Detail Relationship* course, the price was defined on the Services and Prices (RS203000) form for this pair of service and device.

2. Make sure the detail lines are inserted on the **Repair Items** and **Labor** tabs.
3. Change the value in the **Service** box to *Liquid Damage*.
4. Make sure the detail lines on the **Repair Items** and **Labor** tabs remain the same. The new detail lines must be inserted only if the **Repair Items** and **Labor** tabs contain no records.
5. Change the value in the **Service** box back to *Battery Replacement*.
6. Save the record.

Related Links

- [Modification of Data in a PXCache Object](#)
- [RowUpdated Event](#)

Step 1.2.2: Updating Fields of the Same Record on Update of a Field (with FieldUpdated and FieldDefaulting)—Self-Guided Exercise

For a particular row on the **Repair Items** tab of the Repair Work Orders (RS301000) form, if a user selects a value in the **Inventory ID** column, the values in the **Repair Item Type** and **Price** columns should be changed to the repair item type and base price (respectively) of the selected stock item as specified on the [Stock Items](#) (IN202500) form.

In this step, you will add on your own code that copies the `RSSVWorkOrderItem.BasePrice` and `RSSVWorkOrderItem.RepairItemType` values from the stock item record when the `RSSVWorkOrderItem.InventoryID` value is changed.

Updating Fields of the Same Record on Update of the InventoryID Field

As you implement the logic, add the `FieldUpdated` event handler for the `RSSVWorkOrderItem.InventoryID` field and the `FieldDefaulting` event handler for the `RSSVWorkOrderItem.BasePrice` field in the `RSSVWorkOrderEntry` class. Also, you should set the `CommitChanges` property of the `RSSVWorkOrderItem.InventoryID` in the form ASPX.

You can use the instructions provided in [Step 2.2.2: Updating Fields of the Same Record on Update of a Field \(with FieldUpdated and FieldDefaulting\)](#) of the *T210 Customized Forms and Master-Detail Relationship* training course.

Testing the Logic

On the Repair Work Orders (RS301000) form, do the following:

1. Open the 000001 work order.
2. On the **Repair Items** tab, add a row for a repair item of the *Battery* type, and select the *BAT3310EX* stock item in the **Inventory ID** column. Shift the focus away from the box. Make sure the system has filled in values in the **Description** and **Price** columns.
3. Remove the *BAT3310EX* repair item from the list.
4. Save the record.

Lesson Summary

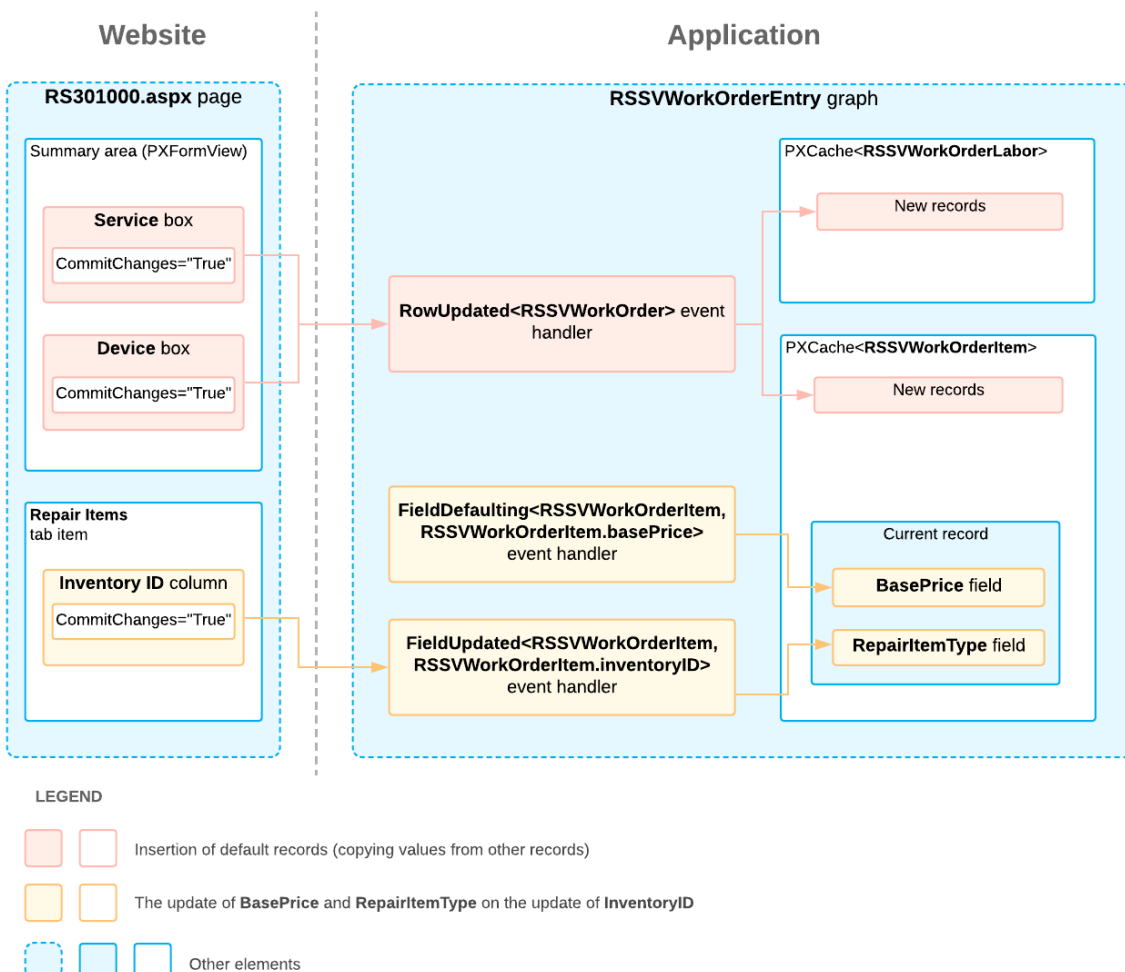


In this lesson, you have learned how to implement basic business logic in a data entry form based on the example of the Repair Work Orders (RS301000) form. You have used the following event handlers to implement changes in the business logic:

- **RowUpdated:** To copy the values of fields of detail records from other records.
- **FieldUpdated and FieldDefaulting:** To modify the values of a detail record on update of the **Inventory ID** column of this detail record.

The following diagram summarizes the implementation.

Copying of Values from Other Records



Review Question

1. Suppose that a user is being created a record on the [Stock Items](#) (IN202500) form. The user has specified the values in **Inventory ID**, **Description**, and **Item Status** of the new record and has not yet saved the record. Now the user is changing the value in the **Description** field. Which status has the new record in PXCACHE?
 - a. Updated
 - b. InsertedUpdated
 - c. Inserted
 - d. InsertedDeleted

Answer Key

1. c

Lesson 1.3: Validating the Field Values

In this lesson, you will implement the validation of particular field values on the Repair Work Orders (RS301000) form. The validation business logic includes the following requirements:

- For each row on the **Labor** tab, the value in the **Quantity** column must satisfy the following conditions:
 - The value must be greater than or equal to 0.
 - The value must be greater than or equal to the value in the **Quantity** column specified for the corresponding record on the **Labor** tab of the Services and Prices form (that is, the record that has the same inventory ID, service ID, and device ID on the Services and Prices form as the current row on the **Labor** tab of the Repair Work Orders form.)
- For a service that requires a preliminary check, the priority of a work order must be at least *Medium*. The preliminary check requirement is specified on the Repair Services (RS201000) form. When a user selects a service in the Summary area, the system must check whether the priority is high enough for the service. If the priority is too low, the system must display an error and cancel the update of the record.

Lesson Objectives

You will learn how to validate the following values:

- The value of a field that does not depend on the values of other fields of the same record
- The value of a field that depends on the values of other fields of the same record

Step 1.3.1: Validating an Independent Field Value (with FieldVerifying)

In this step, you will implement the validation of the value in the **Quantity** column on the **Labor** tab of the Repair Work Orders (RS301000) form. For each row on this tab, the value in the **Quantity** column must be greater than or equal to 0. The value also must be greater than or equal to the value specified for the corresponding record on the **Labor** tab of the Services and Prices (RS203000) form (that is, the record that has the same inventory ID, service ID, and device ID on the Services and Prices form as the current row on the **Labor** tab of the Repair Work Orders form). Thus, a nonnegative quantity must be specified for each row, and the value specified for the labor on the **Labor** tab

of the Services and Prices form (for the same service and device as those selected on the Repair Work Orders form) will function as a minimum quantity.

You will implement the `FieldVerifying` event handler for the `Quantity` field of the `RSSVWorkOrderLabor` DAC. This event handler is intended for field validation that is independent of other fields in the same data record. For details about the validation of independent field values, see [Validation of Field Values](#).

In the event handler, you will do the following:

- When the new value in the **Quantity** column is negative, you will throw an exception (by using `PXSetPropertyException`) to cancel the assignment of the new value to the `Quantity` field.
- When the value is not negative but is smaller than the default quantity specified on the Services and Prices form (in the `RSSVLabor.Quantity` field), you will attach the exception to the field by using the `RaiseExceptionHandling` method and exit the method normally. This method will display a warning for the validated data field but will not raise an exception, so that the method finishes normally and `e.NewValue` is set.

To attach a warning to the control, you will specify `PXErrorLevel.Warning` in the `PXSetPropertyException` constructor.



`RaiseExceptionHandling`, which is used to prevent the saving of a record or to display an error or warning on the form, cannot be invoked on a `PXCache` instance in the following event handlers: `FieldDefaulting`, `FieldSelecting`, `RowSelecting`, and `RowPersisted`. For details, see [RaiseExceptionHandling](#) in the API Reference.

To select the default data record from the `RSSVLabor` DAC, you will configure a fluent BQL query with three required parameters. In the `Select()` method that executes the query, as the parameters, you will pass the values of `RSSVWorkOrder.ServiceID`, `RSSVWorkOrder.DeviceID`, and `RSSVWorkOrderLabor.InventoryID` from the row for which the event is triggered. To use parameters in a fluent BQL query, you need to add the `PX.Data.BQL` using directive to the code. For details about the parameters in fluent BQL, see [Parameters in Fluent BQL](#).

Validating the Value of the Quantity Field

To validate the value of the `Quantity` field, do the following:

1. In the `Messages.cs` file, add the following constants to the `Messages` class.

```
public const string QuantityCannotBeNegative =
    "The value in the Quantity column cannot be negative.";
public const string QuantityTooSmall = @"The value in the Quantity column
    has been corrected to the minimum possible value.";
```

2. In the `RSSVWorkOrderEntry.cs` file, add the following using directive (if it has not been added yet).

```
using PX.Data.BQL;
```

3. Add the following `FieldVerifying` event handler to the `RSSVWorkOrderEntry` graph.

```
//Validate that Quantity is greater than or equal to 0 and
//correct the value to the default if the value is less than the default.
protected virtual void _(Events.FieldVerifying<RSSVWorkOrderLabor,
    RSSVWorkOrderLabor.quantity> e)
{
    if (e.Row == null || e.NewValue == null) return;

    if ((decimal)e.NewValue < 0)
    {
        //Throwing an exception to cancel the assignment of the new value to the field
```

```

        throw new PXSetPropertyException(Messages.QuantityCannotBeNegative);
    }

    var workOrder = WorkOrders.Current;
    if (workOrder != null)
    {
        //Retrieving the default labor item related to the work order labor
        RSSVLabor labor = SelectFrom<RSSVLabor>.
            Where<RSSVLabor.serviceID.IsEqual<@P.AsInt>.
                And<RSSVLabor.deviceID.IsEqual<@P.AsInt>>.
                And<RSSVLabor.inventoryID.IsEqual<@P.AsInt>>>
            .View.Select(this, workOrder.ServiceID, workOrder.DeviceID,
e.Row.InventoryID);
        if (labor != null && (decimal)e.NewValue < labor.Quantity)
        {
            //Correcting the LineQty value
            e.NewValue = labor.Quantity;
            //Raising the ExceptionHandling event for the Quantity field
            //to attach the exception object to the field
            e.Cache.RaiseExceptionHandling<RSSVWorkOrderLabor.quantity>(e.Row,
e.NewValue,
                new PXSetPropertyException(Messages.QuantityTooSmall,
PXErrorLevel.Warning));
        }
    }
}

```

4. Rebuild the project.
5. In the Screen Editor or in the ASPX code in Visual Studio, make sure that `CommitChanges` is set to `True` for the `Quantity` field in the grid of the **Labor** tab on the Repair Work Orders (RS301000) form.
6. Save your changes on the page.
7. Publish the customization project.

Testing the Validation

To check the validation, on the Repair Work Orders (RS301000) form, do the following:

1. Select the work order with the `000001` order number.
2. On the **Labor** tab, in the row for the *CONSULT* labor item, change the value in the **Quantity** column to `-1` and press Enter. Make sure the error is displayed, as shown in the following screenshot.

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 error for a negative value

- Change the value to 0.5, which is smaller than the default value of 1. Make sure that the warning message is generated on the control and the value is corrected to 1, as shown in the following screenshot.

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.

Figure: The warning message

- Change the value to 2. Make sure no warning or error is displayed.
- Save the changes.

Related Links

- [Validation of Field Values](#)
- [Validation of a Data Record](#)

- [Parameters in Fluent BQL](#)
- [PXCACHE.RaiseExceptionHandling Method](#)

Step 1.3.2: Validating Dependent Fields of Records (with RowUpdating)

For a service that requires a preliminary check, the priority of a work order must be at least *Medium*. The preliminary check requirement is specified on the Repair Services (RS201000) form. When a user selects a service in the Summary area, the system must check whether the priority is high enough for the service. If the priority is too low, the system must display an error and cancel the update of the record.

In this step, you will implement validation of the `Priority` field of a work order record in the `RowUpdating` event handler. Because the `Priority` field depends on the `ServiceID` field of a work order record, you will use the `RowUpdating` event handler. The `RowUpdating` event happens during the update of a data record after all field-related events. At this moment, the modifications haven't been applied to the data record stored in the cache yet, and you can cancel the update process.

The event arguments give you access to:

- `e.NewRow`: The modified version of the work order record, which contains all changes made by field-related events
- `e.Row`: The copy of the original work order record stored in the cache

You will use the `ObjectsEqual<>()` method of the cache to compare these two records to find out if the `Priority` or the `ServiceID` field has changed.

You will mark the `Priority` field whose value doesn't pass validation with an error message—by calling the `RaiseExceptionHandling<>()` method of the cache. You will also assign the proper `Priority` field value.



If you want to cancel the changes which have not been saved in the cache, set the `Cancel` property of the event arguments to `true`.

Validating a Work Order Record

To validate a work order record, do the following:

1. In the `Messages.cs` file, add the following constant to the `Messages` class.

```
public const string PriorityTooLow =
    @"The priority must be at least Medium for
    the repair service that requires preliminary check.";
```

2. Add the following `RowUpdating` event handler to the `RSSVWorkOrderEntry` graph.

```
//Display an error if the priority is too low for the selected service
protected virtual void _(Events.RowUpdating<RSSVWorkOrder> e)
{
    // The modified data record (not in the cache yet)
    RSSVWorkOrder row = e.NewRow;
    // The data record that is stored in the cache
    RSSVWorkOrder originalRow = e.Row;

    if (!e.Cache.ObjectsEqual<RSSVWorkOrder>(row, originalRow))
    {
        if (row.Priority == WorkOrderPriorityConstants.Low)
        {
            //Obtain the service record
```

```

RSSVRepairService service = SelectFrom<RSSVRepairService>.
    Where<RSSVRepairService.serviceID.IsEqual<@P.AsInt>>.
    View.Select(this, row.ServiceID);

if (service != null && service.PreliminaryCheck == true)
{
    //Display the error for the Priority field
    WorkOrders.Cache.RaiseExceptionHandling<RSSVWorkOrder.priority>(row,
        originalRow.Priority,
        new PXSetPropertyException(Messages.PriorityTooLow));

    //Assign the proper priority
    e.NewRow.Priority = WorkOrderPriorityConstants.Medium;
}
}
}
}

```

3. Rebuild the project.
4. In the Screen Editor or in ASPX code in Visual Studio, set `CommitChanges` to `True` for the `Priority` field in the Summary area of the Repair Work Orders (RS301000) form and make sure the `Service` box has the same setting.
5. Publish the customization project.

Testing the Logic

To check the validation, on the Repair Work Orders (RS301000) form, do the following:

1. Select the work order with the 000001 order number.
2. In the **Priority** box, select *Low*.
3. In the **Service** box, select the *Liquid Damage* repair service which requires preliminary check. Make sure the error is displayed as the following screenshot shows.

The screenshot shows the 'Repair Work Orders' form for order 000001 - Liquid Damage. The form includes fields for Order Nbr., Status, Date Created, Date Completed, Customer ID, Service, Device, Assignee, and Description. The Priority field is set to 'Low', which has triggered a validation error. A red error message box is displayed over the Priority field, stating: 'The priority must be at least Medium for the repair service that requires preliminary check.' Below the form, there is a table listing repair items:

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

Figure: The error on the page

4. Click **Cancel** on the form toolbar. The changes are discarded and the error is no longer displayed.
5. In the **Service** box, select the *Liquid Damage* service without changing the priority.
6. Save your changes. The saving is performed without errors.
7. In the **Service** box, select the *Battery Replacement* service, and in the **Priority** box, select *Low*.
8. Save your changes. The saving is performed without errors.

Related Links

- [Validation of a Data Record](#)
- [PXRowUpdating Event](#)
- [PXStringListAttribute](#)
- [PXCACHE.RaiseExceptionHandling Method](#)
- [PXCACHE.ObjectsEqual Method](#)

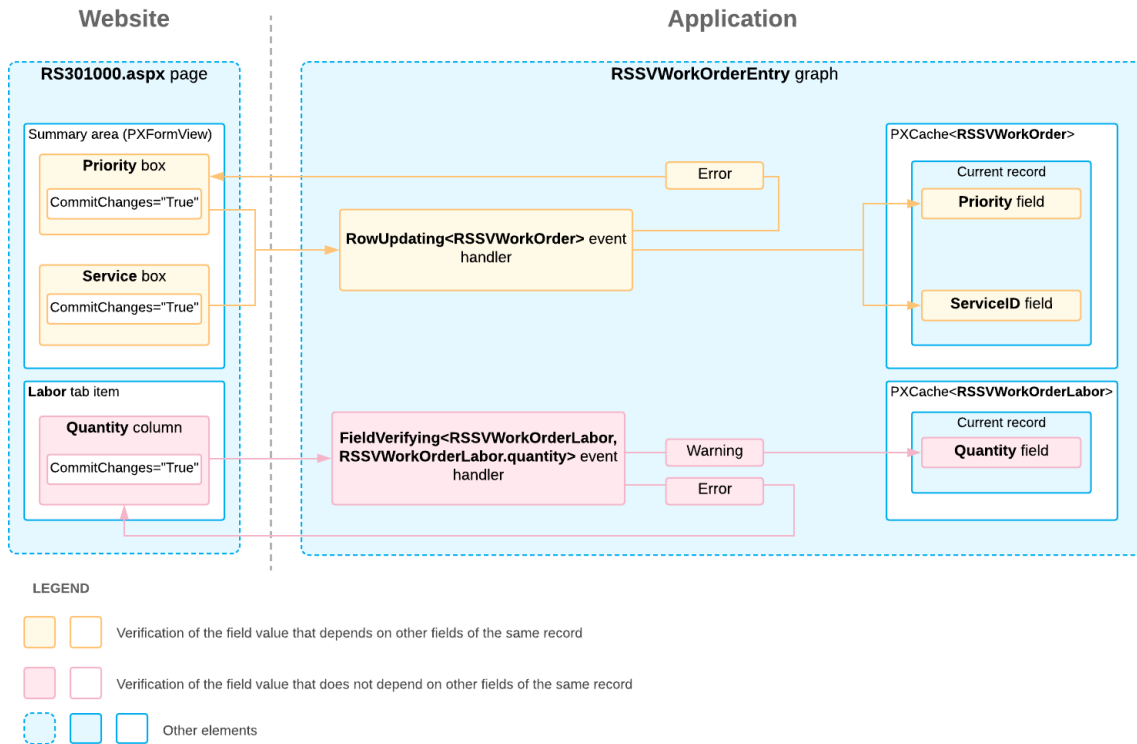
Lesson Summary

In this lesson, you have learned how to implement validation business logic in a data entry form based on the example of the Repair Work Orders (RS301000) form. You have used the following event handlers to implement changes in the business logic:

- **FieldVerifying:** To verify the value of a field that does not depend on other fields of the same record. 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.
- **RowUpdating:** To verify the value of a field that depends on another field of the same record. In this event handler, you have attached the `PXSetPropertyException` exception to the field by using the `RaiseExceptionHandling` method and canceled the update of the record.

The following diagram summarizes the implementation.

Implementation of Verification of Field Values



Review Questions

- Which of the following objects would you use to throw an exception to cancel the assignment of a new value to a field?
 - e.Cancel of the FieldVerifying event handler
 - PXSetPropertyException
 - RaiseExceptionHandling
- Which event handler should be used to validate an independent field value?
 - FieldDefaulting
 - FieldSelecting
 - FieldVerifying
 - FieldUpdated
 - RowSelected
- Which event handler is used to update a value of a dependent field within a particular data record?
 - FieldDefaulting
 - FieldSelecting
 - FieldVerifying
 - FieldUpdated
 - RowSelected

4. How would you specify a required integer parameter in a fluent BQL query?
- a. `@P.AsInt`
 - b. `Argument.AsInt`
 - c. `@P`
 - d. `Argument`

Answer Key

- 1. b
- 2. c
- 3. d
- 4. a

Part 2: Setup Form (Repair Work Order Preferences)

In Acumatica ERP, administrators use setup forms to provide particular configuration parameters for the application. A set of configuration parameters is stored in a single record in the corresponding setup table of the database. By using a setup form, a user can edit this record: for example, turn on or off particular functionality, specify default values, or specify the numbering settings to be used to number documents. Setup forms are used very rarely, usually in the very beginning of application implementation and use.

The names of ASPX pages for setup start with a two-letter abbreviation (indicating the functional area of the form) followed by *10*. For instance, `RS101000.aspx` will be used as the name of the ASPX page for the Repair Work Order Preferences form, which will provide the configuration for the auto-numbering of repair work orders and the default identifier for a walk-in customer.

The names of the graphs for setup forms have the *Maint* suffix (as maintenance forms do).

After you complete the lessons of this part, you will be able to test the auto-numbering functionality.

Lesson 2.1: Configuring the Auto-Numbering of a Field Value

In this lesson, you will create a setup form and learn how to configure auto-numbering of a field value on a data entry form. You will create the Repair Work Order Preferences (RS101000) setup form and configure the auto-numbering of repair work order numbers by using the `AutoNumber` attribute defined in the `PX.Objects.CS` namespace.

Description of the Form Elements

The form will contain the following elements (which are shown in the following screenshot):

- **Numbering Sequence:** A box to contain the numbering sequence that should be used to auto-number repair work order records. By default, the value is set to the *WORKORDER* numbering sequence, which has been preconfigured for this course on the [Numbering Sequences](#) (CS201010) form.
- **Walk-In Customer:** A box to hold the customer ID that should be used by default for the work orders for walk-in repair services—that is, repair services that have the **Walk-In Service** check box selected on the Repair Services (RS201000) form. This logic will not be implemented in this training course.
- **Default Employee:** A box to hold the default assignee for repair work orders. This logic will not be implemented in this training course.
- **Prepayment Percent:** A box to contain the percent of prepayment that a customer should pay for a service that requires prepayment—that is, the service that has the **Requires Prepayment** check box selected on the Repair Services (RS201000) form. This logic will not be implemented in this training course.

Repair Work Order Preferences

NOTES FILES CUSTOMIZATION TOOLS ▾

📁 ↺

* Numbering Sequence:	WORKORDER - Rep 🔍 ✎
* Walk-In Customer:	C000000001 - Jersey 🔍
Default Employee:	Becher, Joseph 🔍
* Prepayment Percent:	10.00

Figure: Repair Work Order Preferences form

Configuration of Auto-Numbering

The `PX.Objects.CS.AutoNumberAttribute` attribute inserts a new number into each new document created by a user before the record is saved to the database. This attribute is designed to use a numbering sequence that has been defined on the [Numbering Sequences](#) (CS201010) form. You will configure the attribute to retrieve the numbering sequence ID from the `RSSVSetup` table, which you have created in [Initial Configuration](#).

The `RSSVSetup` DAC consists of data fields that represent configuration parameters and standard system fields. The corresponding columns in the database should not be null. The `RSSVSetup` class in the application does not contain a primary key field, because each setup form is configured so that it always works with the only record retrieved from the corresponding setup table. However, to support multitenant configurations, the setup table should contain the `CompanyID` column as the primary key in the database. On the application level, this `CompanyID` field is handled automatically by Acumatica Framework. For more information on the `CompanyID` column in the database, see [Multitenancy Support \(CompanyID, CompanyMask\)](#) in the documentation.

Lesson Objectives

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

- Create and use setup forms where users enter the configuration settings of the application
- Configure the auto-numbering of a field value

Step 2.1.1: Creating the Form—Self-Guided Exercise

In this step, you will create the Repair Work Order Preferences (RS101000) form on your own. Although this is a self-guided exercise, you can use the details and suggestions in this topic as you create the form. The creation of a form is described in detail in the *T200 Maintenance Forms* training course.

If you are using the Customization Project Editor to complete the self-guided exercise, you can follow this instruction:

1. Create the form and graph as follows:
 - a. On the toolbar of the Customized Screens page of the Customization Project Editor, click **Create New Screen**.
 - b. In the **Create New Screen** dialog box, which opens, specify the following values:
 - **Screen ID:** RS.10.10.00
 - **Graph Name:** RSSVSetupMaint

- **Graph Namespace:** PhoneRepairShop
 - **Page Title:** Repair Work Order Preferences
 - **Template:** Form (FormView)
- c. Move the generated `RSSVSetupMaint` graph to the extension library.
2. Create and configure the `RSSVSetup` DAC as specified below:
 - a. In Code Editor, generate the `RSSVSetup` DAC and move it to the extension library.
 - b. Configure the `RSSVSetup` DAC in Visual Studio as follows:
 - Define attributes for the system fields. (For details about the definition of the attributes of the system fields, see [Step 1.4.2: Configure the Attributes of the New DAC](#) in the *T200 Maintenance Forms* training course or see [Audit Fields](#), [Concurrent Update Control \(TStamp\)](#), and [Attachment of Additional Objects to Data Records \(NoteID\)](#) in the documentation.)
 - Define the attributes for the `WalkInCustomerID` field as shown below.

```
#region WalkInCustomerID
[CustomerActive(DisplayName = "Walk-In Customer", DescriptionField =
typeof(Customer.acctName))]
[PXDefault]
public virtual int? WalkInCustomerID { get; set; }
public abstract class walkInCustomerID :
PX.Data.BQL.BqlInt.Field<walkInCustomerID> { }
#endregion
```

The `CustomerActive` attribute is defined in the `PX.Objects.AR` namespace.

- Define the attributes for the `DefaultEmployee` field as follows.

```
#region DefaultEmployee
[Owner(DisplayName = "Default Employee")]
[PXDefault]
public virtual int? DefaultEmployee { get; set; }
public abstract class defaultEmployee :
PX.Data.BQL.BqlInt.Field<defaultEmployee> { }
#endregion
```

The `Owner` attribute is defined in the `PX.TM` namespace.

- Define the attributes for the `PrepaymentPercent` field as shown in the following code.

```
#region PrepaymentPercent
[PXDBDecimal()]
[PXDefault(TypeCode.Decimal, "0.0")]
[PXUIField(DisplayName = "Prepayment Percent", Required = true)]
public virtual Decimal? PrepaymentPercent { get; set; }
public abstract class prepaymentPercent :
PX.Data.BQL.BqlDecimal.Field<prepaymentPercent> { }
#endregion
```



You will configure the attributes of the `NumberingId` field and the attributes of the DAC further in this lesson.

3. Define a data view in the generated `RSSVSetupMaint` graph and make the **Save** and **Cancel** standard system buttons available on the form toolbar as the following code shows.

```
public class RSSVSetupMaint : PXGraph<RSSVSetupMaint>
{
```

```

public PXSave<RSSVSetup> Save;
public PXCancel<RSSVSetup> Cancel;

public SelectFrom<RSSVSetup>.View Setup;
}

```

4. Build the project in Visual Studio and publish the customization project.
5. Configure the `RS101000.aspx` page as follows:
 - Set the `PrimaryView` property value of the `DataSource` control to `Setup`.
 - Set the `DataMember` property value of the header form to `Setup`.
 - Do not configure controls on the form. You will create controls in [Step 2.1.3: Configuring the Auto-Numbering of Records \(with CS.AutoNumberAttribute\)](#) further in this lesson.
6. Publish the customization project.
7. Include a link to the form in the **Preferences** group of the **Phone Repair Shop** workspace.
8. Update the `SiteMapNode` item for the Repair Work Order Preferences form in the customization project.

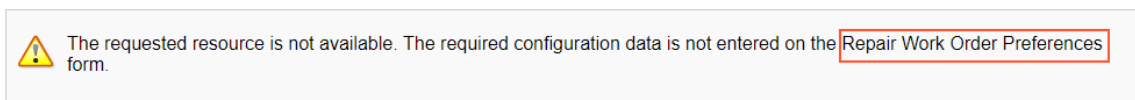
Step 2.1.2: Configuring the DAC for the Setup Form (with `PXPrimaryGraph` and `PXCacheName`)

In this step, you will configure the `RSSVSetup` DAC with the `PXPrimaryGraph` and `PXCacheName` attributes, which are necessary for the link to the setup form to be displayed when no setup data record exists (see the screenshot below).

You use the `PXPrimaryGraph` attribute to specify the graph that corresponds to the default editing form for records of the DAC. The attribute enables support for links to the Repair Work Order Preferences form from other forms.

You use the `PXCacheName` attribute to specify a user-friendly name for a DAC. In particular, this name is used in an error message that is displayed when no setup data records exist. Without the `PXCacheName` attribute, the error message would use the DAC name, `RSSVSetup`, for the link.

Error #0



Next step:

Navigate to the [Repair Work Order Preferences](#) form and enter the required configuration data.

Figure: The form when the setup data has not been specified

Configuring the `RSSVSetup` DAC

Perform the following steps to configure the `RSSVSetup` DAC:

1. In the `RSSVSetup.cs` file, add the `PXPrimaryGraph` attribute to the `RSSVSetup` DAC as shown in the following code.

```
[PXPrimaryGraph(typeof(RSSVSetupMaint))]  
public class RSSVSetup : IBqlTable  
{  
    ...  
}
```

2. Add the `PXCacheName` attribute to the `RSSVSetup` DAC as shown in the following code.

```
[PXCacheName("Repair Work Order Preferences")]
```

3. Rebuild the project.

Related Links

- [PXPrimaryGraphAttribute](#)
- [Configuration Parameters of the Application \(Setup Forms\)](#)

Step 2.1.3: Configuring the Auto-Numbering of Records (with `CS.AutoNumberAttribute`)

In this step, you will configure the auto-numbering of repair work orders.

Configuration of the Numbering Sequence Box on the Setup Form

The numbering sequences for the auto-numbering of records are defined on the [Numbering Sequences](#) (CS201000) form. This form displays the data of the `Numbering` DAC from the `PX.Objects.CS` namespace. You will use this DAC to configure the selector for the **Numbering Sequence** box on the Repair Work Order Preferences (RS101000) form.

For the auto-numbering of repair work orders, you will use the `WORKORDER` numbering sequence, which has been preconfigured for this course. You will set this numbering sequence as the default value for the **Numbering Sequence** box. To display the **Edit** button to the right of the box, you will set `AllowEdit` to `True` for the control that corresponds to the box. When a user clicks this button, the system opens the [Numbering Sequences](#) form, where the user can view and possibly edit the settings of the numbering sequence.

The Attribute for the Auto-Numbering of Repair Work Orders

You will assign the `AutoNumber` attribute from `PX.Objects.CS` to the `OrderNbr` field of the `RSSVWorkOrder` DAC. In the first parameter of the attribute constructor, you will pass the numbering sequence that should be used to auto-number work orders.



Acumatica ERP includes a number of attributes derived from `PX.Objects.CS.AutoNumberAttribute`. In your application, you can use a predefined attribute that suits your needs or implement your own attribute, as described in [Custom Attributes](#) in the documentation.

Changes in the Graph of the Data Entry Form

To make the `RSSVWorkOrderEntry` graph use the numbering sequence specified on the Repair Work Order Preferences (RS101000) form, you will add a data view of the `PXSetup` type and the graph constructor to retrieve setup data from the database. If the current record in this view is null, the server returns the specific error with the link to the setup form. (Acumatica Framework defines the form based on the `PXPrimaryGraph` attribute on the `RSSVSetup` DAC.)



Instead of checking the current record in the graph constructor, you can assign the `PXCheckCurrent` attribute to the data view of the `PXSetup` type in the graph.

Instructions for Configuring the Auto-Numbering of Repair Work Orders

To set up the automatic numbering of repair work orders, complete the following steps:

1. In the `RSSVSetup.cs` file, configure the attributes of the `NumberingID` field as follows:
 - a. Specify the `PXDBString` and `PXUIField` attributes as follows.

```
[PXDBString(10, IsUnicode = true)]
[PXUIField(DisplayName = "Numbering Sequence")]
```

- b. Add the using `PX.Objects.CS;` directive and define the selector for the field as shown in the following code.

```
[PXSelector(typeof(Numbering.numberingID),
    DescriptionField = typeof(Numbering.descr))]
```

- c. Specify the `WORKORDER` numbering sequence as the default value for the field, as shown in the following code.

```
[PXDefault("WORKORDER")]
```

2. Build the project.
3. Edit ASPX of the Repair Work Order Preferences (RS101000) form in the Screen Editor or in Visual Studio as follows:
 - a. Create controls for the `NumberingID`, `WalkInCustomerID`, `DefaultEmployee`, and `PrepaymentPercent` fields of the Repair Work Order Preferences (RS101000) form.
 - b. Adjust the width of control labels and the control sizes by setting the properties of the first layout rule as shown in the following code.

```
<px:PXLayoutRule ControlSize="SM" LabelsWidth="SM"
    ID="PXLayoutRule1" runat="server" StartRow="True">
</px:PXLayoutRule>
```

- c. For the `NumberingID` control, specify the `AllowEdit` property as `true` to provide redirection to the [Numbering Sequences](#) (CS201000) form.
4. In the `RSSVWorkOrder.cs` file, add the using `PX.Objects.CS;` directive and specify the `AutoNumber` attribute for the `OrderNbr` field, as shown in the following code.

```
#region OrderNbr
[PXDBString(15, IsKey = true, IsUnicode = true, InputMask =
">CCCCCCCCCCCCCCCC")]
[PXDefault(PersistingCheck = PXPersistingCheck.NullOrBlank)]
[PXUIField(DisplayName = "Order Nbr.", Visibility =
PXUIVisibility.SelectorVisible)]
[AutoNumber(typeof(RSSVSetup.numberingID), typeof(RSSVWorkOrder.dateCreated))]
[PXSelector(typeof(Search<RSSVWorkOrder.orderNbr>))]
public virtual string OrderNbr { get; set; }
public abstract class orderNbr : PX.Data.BQL.BqlString.Field<orderNbr> { }
#endregion
```

5. In the `RSSVWorkOrderEntry` graph, add the `AutoNumSetup` data view and the constructor as follows.

```

#region Views
...
//The view for the auto-numbering of records
public PXSetup<RSSVSetup> AutoNumSetup;
#endregion

#region Graph constructor
public RSSVWorkOrderEntry()
{
    RSSVSetup setup = AutoNumSetup.Current;
}
#endregion

```

6. Rebuild the project.
7. Publish the customization project.

Testing the Auto-Numbering

Do the following to test the auto-numbering:

1. Open the Repair Work Orders (RS301000) form and see the error displayed on the form, as the following screenshot shows.

Error #0



The requested resource is not available. The required configuration data is not entered on the [Repair Work Order Preferences](#) form.

Next step:

Navigate to the [Repair Work Order Preferences](#) form and enter the required configuration data.

Figure: The form with the error

2. Click the link to open the Repair Work Order Preferences (RS101000) form.
3. On the form, make sure the *WORKORDER* numbering sequence is selected in the **Numbering Sequence** box, and click the **Edit** button to the right of the box.
4. On the [Numbering Sequences](#) (CS201000) form, which opens for the *WORKORDER* numbering sequence, do the following:
 - a. Type *000001* in the **Last Number** column. (You have already created a work order with number *000001* manually.)
 - b. Make sure the other settings of the sequence are the same as those shown in the screenshot below.

Numbering Sequences

NOTES FILES CUSTOMIZATION TOOLS

* Numbering ID: WORKORDER
 * Description: Repair Work Orders
☐ Manual Numbering
 * New Number Symbol: <NEW>

Branch	* Start Number	* End Number	* Start Date	* Last Number	* Warning Number	* Numbering Step
	000000	999999	1/1/1900	000001	999899	1

Figure: The WORKORDER numbering sequence

- c. Save your changes.
5. Close the Numbering Sequences form.
6. On the Repair Work Order Preferences form, specify the following settings and save your changes:
 - **Walk-in Customer:** C000000001
 - **Default Employee:** Becher, Joseph
 - **Prepayment Percent:** 10
7. Open the Repair Work Orders form. Now the error is no longer displayed for the form, and you can create a new record.
8. On the form toolbar, click **Add New Record**. Notice that the <NEW> placeholder appears in the **Order Nbr.** box.
9. Specify the following settings for the new work order:
 - **Customer ID:** C000000001
 - **Service:** Screen Repair
 - **Device:** iPhone 6
 - **Description:** Screen repair, iPhone 6
10. Save your changes. Make sure the new repair work order has the 000002 order number assigned to it, as shown in the following screenshot.

Repair Work Orders

000002 - Screen Repair

NOTESFILESCUSTOMIZATIONTOOLS

←↻+🗑️📄⌂⏪⏩⏴⏵

Order Nbr.:000002🔍

* Customer ID:C000000001 - Jersey Central Office E🔍

Order Total:0.00

Status:On Hold

* Service:SCREENREPAIR - Screen Repair🔍

Invoice Nbr.:

* Date Created:3/4/2022

* Device:IPHONE6 - iPhone 6🔍

Date Completed:

Assignee:

Priority:Medium

Description:Screen repair, iPhone 6

REPAIR ITEMSLABOR

🔄+✎️×

📄🔍🗑️

Repair Item Type	Inventory ID	Description	Price
------------------	--------------	-------------	-------

Figure: A work order with automatically assigned number

Related Links

- *Configuration Parameters of the Application (Setup Forms)*

Lesson Summary

In this lesson, you have created a setup form and learned how to configure the auto-numbering of data records on a data entry form.

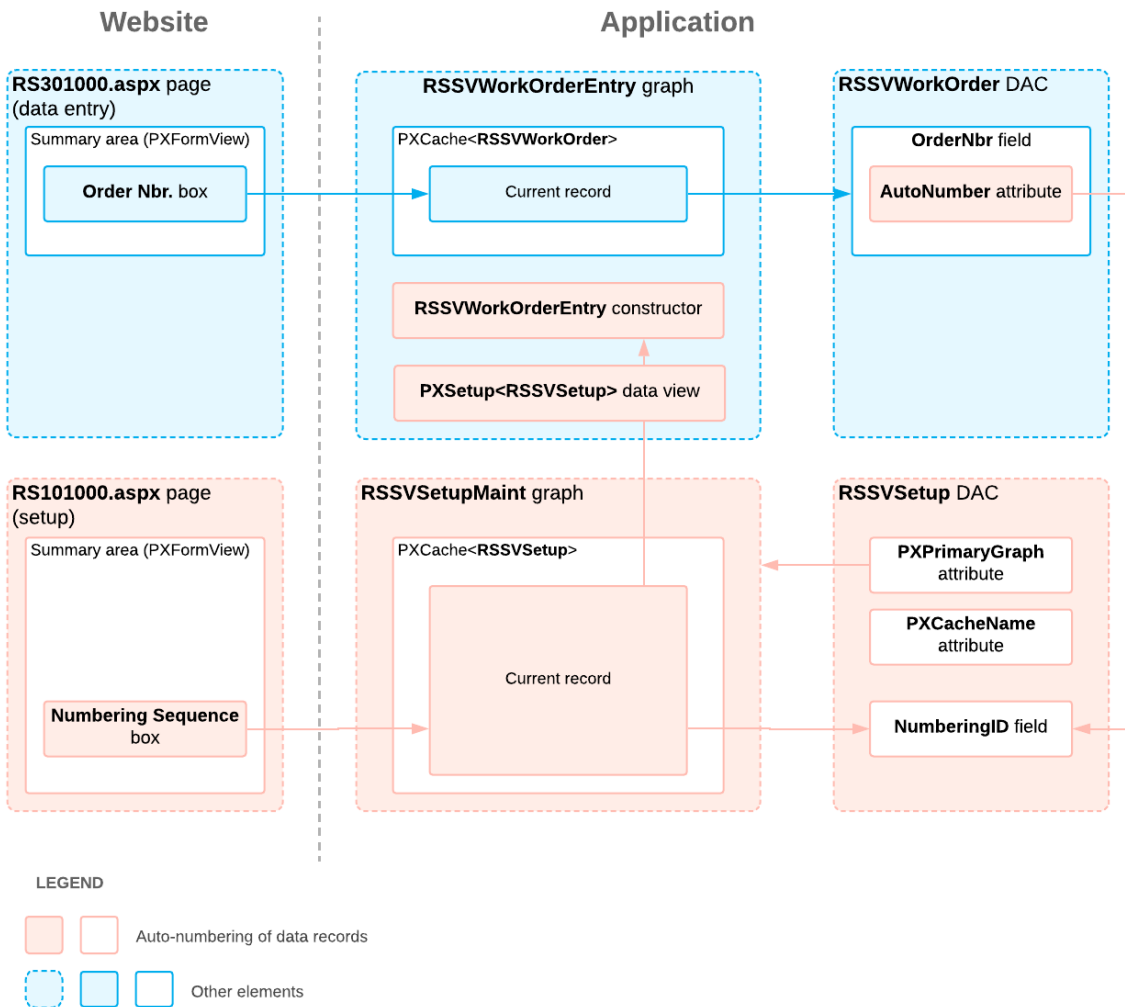
To configure a link to the setup form (which is displayed when the configuration parameters are not specified), you have assigned the `PXPrimaryGraph` and `PXCacheName` attributes to the `RSSVSetup` DAC.

To configure the auto-numbering of repair work order records, you have done the following:

- Added a box to the setup form that contains the numbering sequence to be used for the auto-numbering of records
- Assigned the `AutoNumber` attribute from `PX.Objects.CS` to the field of the DAC for the data entry form
- Added the data view of the `PXSetup` type and the graph constructor that checks the current record in this data view to the graph of the data entry form

The following diagram summarizes the implementation of automatic numbering.

Implementation of Auto-Numbering of Data Records



Review Questions

- Which of the following classes would you use to define the primary data view for a setup form?
 - PXSetup
 - SelectFrom<>.View
 - PXSave
- In a database of a multitenant Acumatica ERP instance, what is the maximum number of records that the database table that corresponds to the primary DAC of a setup form can contain?
 - 0
 - 1
 - A number that is equal to the number of tenants
 - Any number

Answer Key

1. b
2. c

Additional Information: Custom Feature Switches

In this training course, you have created two custom forms. You may want to make custom forms available for a user only if a custom feature is enabled on the [Enable/Disable Features](#) (CS100000) form. The creation of custom feature switches is outside of the scope of this course but may be useful to some readers.

For details about custom feature switches, see [To Add a Custom Feature Switch](#) in the documentation.

Appendix: Use of Event Handlers

This topic lists the scenarios in which particular event handlers have been used in this course.

Table: Use of Event Handlers

Event	Scenario	Examples in the Guide
FieldDefaulting	Set a default value of a field depending on other field values	Step 1.2.2: Updating Fields of the Same Record on Update of a Field (with FieldUpdated and FieldDefaulting)—Self-Guided Exercise
FieldUpdated	Update of a field of a data record when another field of this record is updated	Step 1.2.2: Updating Fields of the Same Record on Update of a Field (with FieldUpdated and FieldDefaulting)—Self-Guided Exercise
FieldVerifying	Validation of an independent field value	Step 1.3.1: Validating an Independent Field Value (with FieldVerifying)
RowUpdated	Insertion of detail lines when particular fields of the master record are updated	Step 1.2.1: Creating a Work Order from a Template (with RowUpdated)
RowUpdating	Validation of a field value that depends on another field of the same record	Step 1.3.2: Validating Dependent Fields of Records (with RowUpdating)

Appendix: Reference Implementation

You can find the reference implementation of the customization described in this course in the `Customization\T220` folder of the [Help-and-Training-Examples](#) repository in Acumatica GitHub.

Appendix: Deploying the Needed Acumatica ERP Instance for the Training Course



If for some reason you cannot complete the instructions in [Step 2: Preparing the Needed Acumatica ERP Instance for the Training Course](#), you can create an Acumatica ERP instance as described in this topic and manually publish the needed customization project as described in [Appendix: Publishing the Required Customization Project](#).

You deploy an Acumatica ERP instance and configure it as follows:

1. To deploy a new application instance, open the Acumatica ERP Configuration Wizard, and do the following:
 - a. On the Database Configuration page, type the name of the database: `PhoneRepairShop`.
 - b. On the Tenant Setup page, set up a tenant with the `I100` data inserted by specifying the following settings:
 - **Login Tenant Name:** `MyTenant`
 - **New:** Selected
 - **Insert Data:** `I100`
 - **Parent Tenant ID:** `1`
 - **Visible:** Selected
 - c. On the **Instance Configuration** page, in the **Local Path of the Instance** box, select a folder that is outside of the `C:\Program Files (x86)` or `C:\Program Files` folder. We recommend that you store the website folder outside of these folders to avoid an issue with permission to work in these folders when you perform customization of the website.

The system creates a new Acumatica ERP instance, adds a new tenant, and loads the selected data to it.

2. Sign in to the new tenant by using the following credentials:

- Username: `admin`
- Password: `setup`

Change the password when the system prompts you to do so.

3. In the top right corner of the Acumatica ERP screen, click the username and then click **My Profile**. On the **General Info** tab of the [User Profile](#) (SM203010) form, which the system has opened, select `YOGIFON` in the **Default Branch** box; then click **Save** on the form toolbar.

In subsequent sign-ins to this account, you will be signed in to this branch.

4. Optional: Add the [Customization Projects](#) (SM204505) and [Generic Inquiry](#) (SM208000) forms to your favorites. For details about how to add a form to favorites, see [Managing Favorites: General Information](#).

Appendix: Publishing the Required Customization Project



If for some reason you cannot complete the instructions in [Step 2: Preparing the Needed Acumatica ERP Instance for the Training Course](#), you can create an Acumatica ERP instance as described in [Appendix: Deploying the Needed Acumatica ERP Instance for the Training Course](#) and manually publish the needed customization project as described in this topic.

Load the customization project with the results of the *T210 Customized Forms and Master-Detail Relationship* training course and publish this project as follows:

1. On the Customization Projects (SM204505) form, create a project with the name *PhoneRepairShop*, and open it.
2. In 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\T210\PhoneRepairShop` folder, which you have downloaded from Acumatica GitHub, and click **OK**.
4. Bind the customization project to the source code of the extension library as follows:
 - a. Copy the `Customization\T210\PhoneRepairShop_Code` folder to the `App_Data\Projects` folder of the website.



By default, the system uses the `App_Data\Projects` folder of the website as the parent folder for the solution projects of extension libraries.

If the website folder is outside of the `C:\Program Files (x86)` and `C:\Program Files` folders, we recommend that you use the `App_Data\Projects` folder for the project of the extension library.

- b. Open the solution, and build the `PhoneRepairShop_Code` project.
 - c. Reload the Customization Project Editor.
 - d. In the menu of the Customization Project Editor, click **Extension Library > Bind to Existing**.
 - e. In the dialog box that opens, specify the path to the `App_Data\Projects\PhoneRepairShop_Code` folder, and click **OK**.
5. In the menu of the Customization Project Editor, click **Publish > Publish Current Project**.



The **Modified Files Detected** dialog box opens before publication because you have rebuilt the extension library in the `PhoneRepairShop_Code` Visual Studio project. The `Bin\PhoneRepairShop_Code.dll` file has been modified and you need to update it in the project before the publication.

The published customization project contains all changes to the Acumatica ERP website and database that have been performed in the *T200 Maintenance Forms* and *T210 Customized Forms and Master-Detail Relationship* training courses. This project also contains the customization plug-ins that fill in the tables created in the *T200 Maintenance Forms* and *T210 Customized Forms and Master-Detail Relationship* training courses with the custom data entered in these training courses. For details about the customization plug-ins, see [To Add a Customization Plug-In to a Project](#). Creation of customization plug-ins is outside of the scope of this course.