Developer Course

Customization

T240 Processing Forms 2022 R2

Revision: 10/12/2022

Acumatica
The Cloud ERP

Contents

Copyright

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 R2

Last Updated: 10/12/2022

Introduction

The *T240 Processing Forms* training course teaches you how you can create processing forms by using Acumatica Framework and the customization tools of Acumatica ERP. A processing form is a form on which users can invoke an operation on multiple selected records at once.

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 previous training courses of the *T* series (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 used to define Acumatica ERP processing 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 the *Course Prerequisites*, perform the *Initial Configuration*, and carefully read the *Company Story and Customization Description*.
2. Complete the lessons in all parts of the training guide.
3. In Partner University, take *T240 Certification Test: Processing 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 two types of processing forms: a form without filtering parameters, and a form with filtering parameters.

The second part of the course shows how to implement the update of the frequently edited fields (by using a custom `PXAccumulator` attribute) and use these fields on a processing form.

The third part of the course shows the implementation of redirection to a report at the end of processing.

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 the Documentation Resources Are

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.

Which License You Should Use

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. Before you begin this course, we recommend that you complete the following training courses:

- *T200 Maintenance Forms*
- *T210 Customized Forms and Master-Detail Relationship*
- *T220 Data Entry and Setup Forms*
- *T230 Actions*
- *T270 Workflow API*

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.

Step 1: Preparing the Environment

If you have completed any of the training courses of the *T* series 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 conforms to the *System Requirements for Acumatica ERP 2022 R2*.
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

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)`, `C:\Program Files`, and `C:\Users` 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**. The *User Profile* (SM203010) form opens. On the **General Info** tab, 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 Table

Create the database table that is necessary for the *T240 Processing Forms* training course and include the script for table creation in the customization project as follows:

1. In SQL Server Management Studio, execute the `T240_DatabaseTables.sql` script to create the database tables that are necessary for the *T240 Processing Forms* training course.

   This script creates the `RSSVEmployeeWorkOrderQty` table, which is new for this course.
2. On the Database Scripts page of the Customization Project Editor, for the 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 previous training courses of the *T* series.

> You have loaded and published the customization project with the results of these courses as described in *Initial Configuration*.

In the previous training courses of the *T* series, you have created the following forms:

- The Repair Services (RS201000) custom maintenance form, which the Smart Fix company uses to manage the lists of repair services that the company provides
- The Serviced Devices (RS202000) custom maintenance form, which the Smart Fix company uses to manage the lists of devices that can be serviced
- The Services and Prices (RS203000) custom maintenance form, which provides users with the ability to define and maintain the price for each provided repair service
- The Repair Work Orders (RS301000) custom data entry form, which is used to create and manage work orders for repairs
- The Repair Work Order Preferences (RS101000) custom setup form, which an administrative user uses to specify the company's preferences for the repair work orders

In the previous training courses of the *T* series, you have also customized the *Stock Items* (IN202500) form 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 Assign Work Orders (RS501000) custom processing form, which users will use to assign multiple repair work orders at the same time. You will implement the functionality of the form in stages. First, you will implement this form as a simple processing form without any filtering parameters for user selection. Then you will add a filter to the form so that only the records that satisfy the filtering parameters are displayed in the table. Also, you will implement the selection of the default assignee, which depends on the number of already assigned work orders for the employees. You will use a custom `PXAccumulator` attribute to update the number of assigned orders in the database for each employee. Finally, you will implement redirection to a report at the end of the processing.

Assign Work Orders Form

The following screenshot shows how the Assign Work Orders (RS301000) form will look at the end of the course.

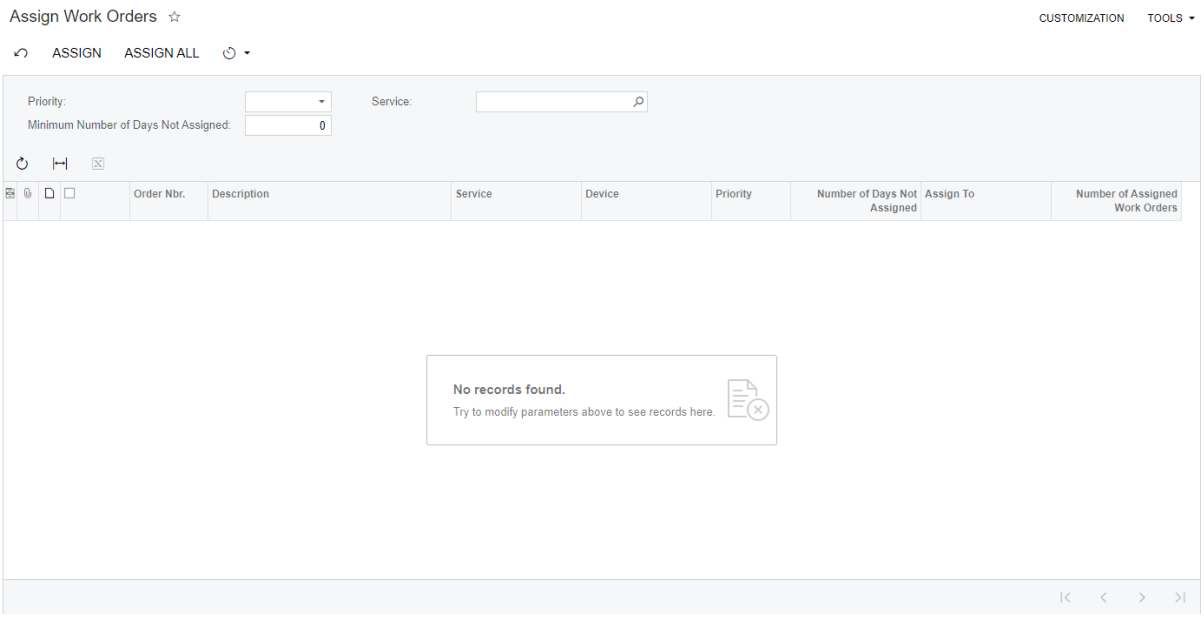


Figure: Assign Work Orders form

The form will contain the following elements:

- Two processing buttons on the toolbar: **Assign** and **Assign All**, which a user will use to assign only the selected work orders (that is, those for which the user has selected the unlabeled check boxes) or all of the listed work orders, respectively, in the table.
- The filtering UI elements in the Summary area, which a user can use to filter the list of repair work orders by the priority, the number of days the work order is not assigned, or the service that should be provided.
- The table that displays the list of work orders that have the *Ready for Assignment* status and meet the other filtering criteria specified. Each row of the table lists a work order along with additional information about it, such as the number of days the order has been unassigned, the assignee to which the order will be assigned, and the number of orders that this assignee is currently is working on. A user can change the assignee for any work order in the table.

This form will use the following custom tables:

- `RSSVWorkOrder`: The data of this table will be displayed in the table on the form. The table has been added to the application database in *Step 2: Preparing the Needed Acumatica ERP Instance for the Training Course*.
- `RSSVEmployeeWorkOrderQty`: The data of this table will be used to display the number of assigned work orders of an employee in the table on the form. The table has been added to the application database in *Step 3: Creating the Database Table*.

The filtering elements in the Summary area will use the `RSSVWorkOrderToAssignFilter` DAC, which contains only unbound fields. Therefore, no table corresponds to this DAC in the database.

Part 1: Processing Form (Assign Work Orders)

The Smart Fix company needs to have a custom Acumatica ERP form that the managers of the company will use to assign repair work orders to particular employees. For this purpose, in this part of the course, you will create the Assign Work Orders (RS501000) processing form, which is described in *Company Story and Customization Description*.

On a processing form, users can invoke an operation on multiple selected records at once. For instance, a processing operation can be a procedure that modifies the status of documents.

Processing forms have IDs that start with a two-letter abbreviation (indicating the functional area of the form) followed by *50* (indicating a processing form), such as *RS501000*. The names of the graphs that work with processing forms have the *Process* suffix. For instance, `RSSVAssignProcess` will be the name of the graph for the Assign 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 you have created.

Lesson 1.1: Creating a Simple Processing Form

In this lesson, you will create a simple processing form that displays the records to be processed and does not have any filtering parameters. You will create the Assign Work Orders (RS501000) custom processing form, which you will modify for expanded functionality in future lessons.

In a table, the form will display the repair work orders that have the *Ready for Assignment* status. To give users the ability to process these work orders, the form will have two buttons on the toolbar (**Assign** and **Assign All**). The processing operation will change the status of each processed work order to *Assigned* and assign the work order to the employee specified in the **Assignee** column, if one has been specified. For the work orders for which no assignee has been specified, the default employee defined on the Repair Work Order Preferences (RS101000) form is inserted as the assignee of the work order. (The default assignee has been specified on the Repair Work Order Preferences form in the *T220 Data Entry and Setup Forms* training course.) As the processing operation, you will use the **Assign** action that you published with the customization package in *Initial Configuration*.

At the end of the lesson, the form will look as shown in the following screenshot.

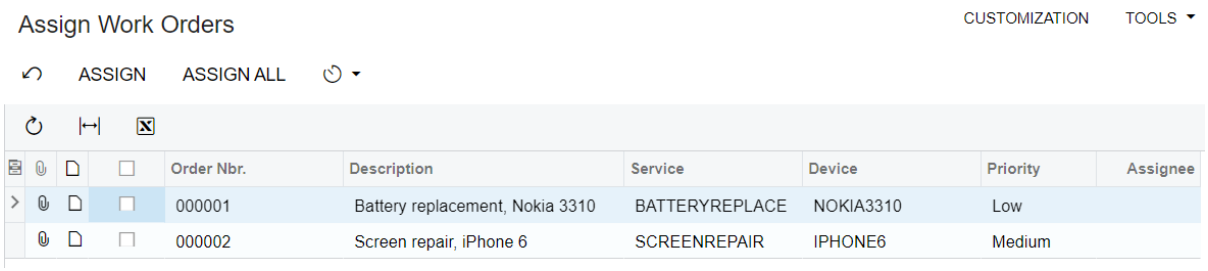


Figure: Assign Work Orders form

Lesson Objectives

In this lesson, you will learn how to create a simple processing form (that is, one that does not have any filtering parameters defined).

Step 1.1.1: Creating the Form—Self-Guided Exercise

In this step, you will create the Assign Work Orders (RS501000) form on your own. Although this is a self-guided exercise, this topic provides details and suggestions you can use 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.50.10.00`
      - **Graph Name**: `RSSVAssignProcess`
      - **Graph Namespace**: `PhoneRepairShop`
      - **Page Title**: `Assign Work Orders`
      - **Template**: *Grid (GridView)*
   c. Move the generated `RSSVAssignProcess` graph to the extension library.
2. Make sure that the `RSSVWorkOrder` DAC is defined in the `PhoneRepairShop_Code` Visual Studio project.
3. Do not make any standard system actions available during the initial definition of the `RSSVAssignProcess` graph. You will define the actions in *Step 1.1.3: Configuring the Processing Graph and Data View (with ProcessingView and RowSelected)*.
4. Do not define any data views of the `RSSVAssignProcess` graph at this time. You will define the data view in *Step 1.1.3: Configuring the Processing Graph and Data View (with ProcessingView and RowSelected)*.
5. Build the project in Visual Studio.
6. Update the customization project with a new version of the `PhoneRepairShop_Code.dll` and publish the customization project.
7. Include a link to the Assign Work Orders form in the **Processes** category of the **Phone Repair Shop** workspace.
8. Update the *SiteMapNode* item for the Assign Work Orders form in the customization project.

Step 1.1.2: Changing the Processing Action

In this step, you will modify the **Assign** action of the Repair Work Orders (RS301000) form, which assigns a repair work order to an employee and changes the status of the order to *Assigned*. You will make the following changes to this action so that it can be also used on the Assign Work Orders (RS501000) processing form (the way the action works on the Repair Work Orders form will not be modified):

- You will modify the `assign()` action handler as follows:
  - You will move the code from the `assign()` action handler to the separate `AssignOrders()` static method.
  - You will change the signature of the action handler so that it returns `IEnumerable`. If you use the `void` action handler instead, the processing of the long-running operation and its result will not be displayed in the UI.

- You will replace the `PXButton` attribute with the `PXProcessButton` attribute to indicate that the action will be used on the processing form.
- To run the `AssignOrders()` processing method within the `assign()` action handler, you will invoke the `PXLongOperation.StartOperation()` method, which starts execution of the processing method in a separate thread. The use of the `PXLongOperation.StartOperation()` method is the only way to execute the processing method asynchronously in Acumatica Framework.

  Before you run the operation, you invoke the `Save.Press()` method to save the last changes made on the data entry page, to be sure to process the latest version of the work order.

  You need to call `Save.Press()` instead of `Actions.PressSave()` in an action that is used in a workflow and starts a long-running operation.

- In the separate `AssignOrders()` static method, you will do the following changes to the code of the action:
  - You will modify the code so that it works with the list of repair work orders obtained from the input parameter of the method.
  - You will add the `isMassProcess` parameter to the `AssignOrders()` method. If `isMassProcess = true` is passed in the method parameters (which means that the method is invoked from a processing form), you will return a successful processing message to the UI by using the static `PXProcessing<T>.SetInfo()` method.
  - To handle errors that might occur during the processing, you will enclose the processing code in the `try` statement. If any error occurs, in the `catch` statement, by using the static `PXProcessing<T>.SetError()` method, you will return the processing result for each repair work order to the UI.

Changing the Assign Action

Make the changes to the action as follows:

1. In the `Messages` class, add the following string. This message will be returned to the UI after the successful processing of each work order on a processing form.

```
public const string WorkOrderAssigned =
    "The {0} work order has been successfully assigned.";
```

2. In the `RSSVWorkOrderEntry` graph, define the `AssignOrders()` static method as follows.

```
public static void AssignOrders(List<RSSVWorkOrder> list,
    bool isMassProcess = false)
{
    var workOrderEntry = PXGraph.CreateInstance<RSSVWorkOrderEntry>();
    for (int i = 0; i < list.Count; i++)
    {
        if (list[i] == null)
            continue;

        RSSVWorkOrder workOrder = list[i];
        try
        {
            workOrderEntry.Clear();
            workOrderEntry.WorkOrders.Current = workOrder;
            //If the assignee is not specified,
            //specify the default employee.
            if (workOrder.Assignee == null)
            {
                //Retrieve the record with the default setting
```

```
                    RSSVSetup setupRecord =
                        workOrderEntry.AutoNumSetup.Current;
                    workOrder.Assignee = setupRecord.DefaultEmployee;
                }

                //Update the work order in the cache.
                workOrderEntry.WorkOrders.Update(workOrder);

                //Trigger the Save action to save the changes
                //to the database
                workOrderEntry.Actions.PressSave();

                //Display the message to indicate successful processing.
                if (isMassProcess)
                {
                    PXProcessing<RSSVWorkOrder>.SetInfo(i,
                        string.Format(Messages.WorkOrderAssigned,
                        workOrder.OrderNbr));
                }
            }
            catch (Exception e)
            {
                PXProcessing<RSSVWorkOrder>.SetError(i, e);
            }
        }
    }
```

3. Invoke the `AssignOrders()` method in the action handler for the `Assign` action of the `RSSVWorkOrderEntry` graph as follows.

```
    public PXAction<RSSVWorkOrder> Assign;
    [PXProcessButton]
    [PXUIField(DisplayName = "Assign")]
    protected virtual IEnumerable assign(PXAdapter adapter)
    {
        bool isMassProcess = adapter.MassProcess;
        // Populate a local list variable.
        List<RSSVWorkOrder> list = new List<RSSVWorkOrder>();
        foreach (RSSVWorkOrder order in adapter.Get<RSSVWorkOrder>())
        {
            list.Add(order);
        }
        // Trigger the Save action to save changes in the database.
        Save.Press();

        PXLongOperation.StartOperation(this, delegate ()
        {
            AssignOrders(list, isMassProcess);
        });

        // Return the local list variable.
        return list;
    }
```

4. Rebuild the project.

~~Testing the Modified Action~~

The modified action will work with both the Repair Work Orders (RS301000) form and the Assign Work Orders (RS501000). You have not added the action to the `RSSVAssignProcess` graph, which works with the Assign Work Orders form, so that form cannot be tested yet. You can, however, make sure that the modified **Assign** action on the Repair Work Orders form works correctly. Proceed as follows:

1. On the Repair Work Orders form, create a new work order with the following settings:
   - **Customer ID**: *C000000001*
   - **Service**: *Battery Replacement*
   - **Device**: *Nokia 3310*
   - **Description**: `Battery replacement, Nokia 3310`
2. Save the work order.
3. On the form toolbar, click **Remove Hold** and make sure the work order has the *Ready for Assignment* status.
4. On the form toolbar, click **Assign**. Make sure the work order has the *Assigned* status and the **Assignee** box is not empty, as shown in the following screenshot. If assignee is not specified in the repair work order, the system fills in the value specified in the **Default Assignee** box on the Repair Work Order Preferences (RS101000) form.

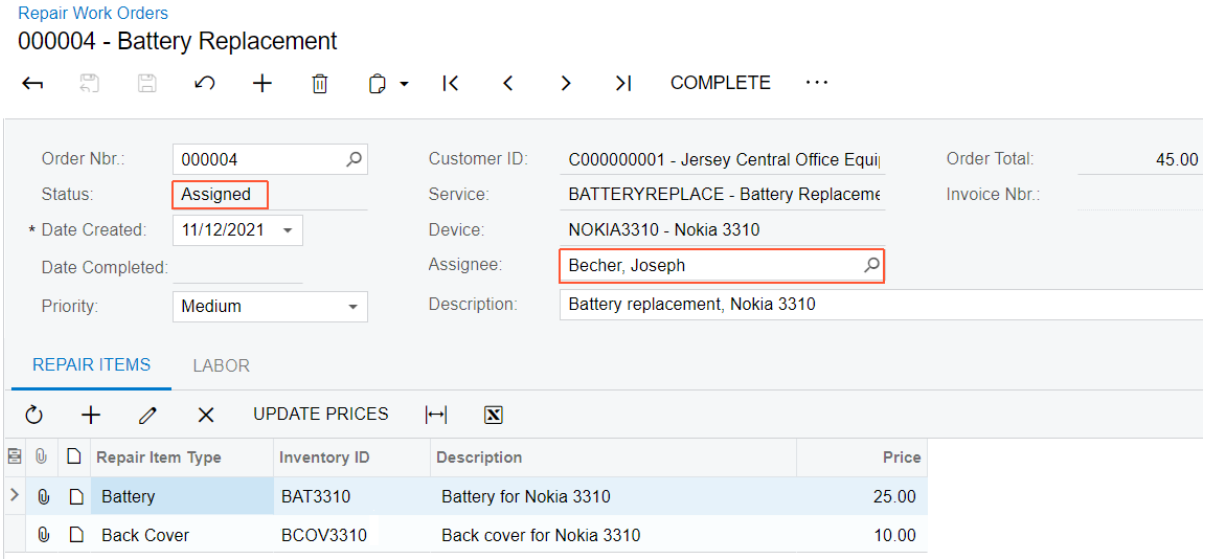


Figure: Assigned work order

Related Links

- *PXProcessing.SetInfo Method*
- *PXLongOperation.StartOperation Method*
- *Asynchronous Execution*
- *PXProcessButtonAttribute Class*

Step 1.1.3: Configuring the Processing Graph and Data View (with ProcessingView and RowSelected)

In this step, you will configure the `RSSVAssignProcess` graph that works with the Assign Work Orders (RS501000) form to be a processing graph as follows:

- You will define the data view for the Assign Work Orders form.

  To define the data view for the processing form, you will use the `SelectFrom<Table>.ProcessingView` class. This class is derived from the `PXProcessingBase<Table>` class, which is a base class for the data views of processing forms. Inside the `Where` condition, you will use a fluent BQL statement that selects only the repair work orders with the *Ready for Assignment* status.
- You will add this data view and the processing actions to the `RSSVAssignProcess` graph.

  The processing form will have one system action (**Cancel**) and two custom processing actions (**Assign** and **Assign All**). By default, any form that has a data view of a type derived from `PXProcessingBase<Table>` has the **Process** and **Process All** buttons on the form toolbar. You will replace the names of the default buttons to **Assign** and **Assign All** in the graph constructor. To override the button captions, you will use the `SetProcessCaption()` and `SetProcessAllCaption()` methods.
- You will specify the workflow action to be used for processing.

  In the `RowSelected` event handler, you will specify the workflow action that the processing form should use for processing. You will invoke the `SetProcessWorkflowAction<>()` method of the data view.

  - We recommend that you do not call the `SetProcessWorkflowAction<>()` method in the graph constructor because this can cause incorrect initialization of the workflow.
  - For the forms that do not use workflow actions for processing, you must specify the processing delegate by using the `SetProcessDelegate()` method. For details about processing delegates, see *Implementation of Processing Operations*.
- You will modify the action definition in the workflow so that the action can be used on the processing form.

  In the action definition in the workflow, you will call the `MassProcessingScreen<>()` method with the `RSSVAssignProcess` graph as the type parameter. You will also call the `InBatchMode()` method because the `Assign` action works with the list of records.

Configuring the RSSVAssignProcess Graph

Do the following:

1. In the `RSSVAssignProcess.cs` file, add the following `using` directives.

```
using PhoneRepairShop.Workflows;
using PX.Data.BQL.Fluent;
```

2. In the `RSSVAssignProcess` graph, use the following code to define the `Cancel` action for the toolbar and the `WorkOrders` data view that provides data records to be processed on the form.

```
    public class RSSVAssignProcess : PXGraph<RSSVAssignProcess>
    {
        public PXCancel<RSSVWorkOrder> Cancel;
        public SelectFrom<RSSVWorkOrder>.
            Where<RSSVWorkOrder.status.
                IsEqual<RSSVWorkOrderWorkflow.States.readyForAssignment>>.
            ProcessingView WorkOrders;    }
```

3. In the `RSSVAssignProcess` graph, define the constructor of the graph as follows.

```
        public RSSVAssignProcess()
        {
            WorkOrders.SetProcessCaption("Assign");
            WorkOrders.SetProcessAllCaption("Assign All");
        }
```

4. In the `RSSVAssignProcess` graph, define the following `RowSelected` event handler.

```
protected virtual void _(Events.RowSelected<RSSVWorkOrder> e)
{
    WorkOrders.SetProcessWorkflowAction<RSSVWorkOrderEntry>(
        g => g.Assign);
}
```

5. In the `RSSVWorkOrderWorkflow` class, in the lambda expression for the `WithActions` method, modify the `Assign` action definition as follows.

```
actions.Add(g => g.Assign,
    c => c.WithCategory(
        processingCategory, g => g.PutOnHold)
    .MassProcessingScreen<RSSVAssignProcess>()
    .InBatchMode());
```

6. Rebuild the project.

Related Links

- *PXProcessingBase<Table> Class*

Step 1.1.4: Creating Controls for the Processing Form

In this step, you will create controls for the Assign Work Orders (RS501000) processing form.

You will add the unbound `Selected` data field of the Boolean type to the `RSSVWorkOrder` DAC and then add the column for this field to the form. If a user doesn't want to process all listed records, the user will use this column to select the work order records to be assigned during processing. You will define the `Selected` data field as unbound by using the `PXBool` type attribute. (Unlike the `PXDBBool` attribute, the `PXBool` attribute does not have the *DB* part in its name. The presence of the *DB* part indicates a bound data type.)

You will make all columns in the grid (except for the column that corresponds to the `Selected` field) unavailable for editing by specifying `SkinID="Inquire"` for the grid. For the `Selected` column, you will set the `AllowCheckAll` property of the corresponding control to `True` to make it possible for the users to select all work orders listed on the current page of the table for assignment.

Creating Controls

To create the needed controls for the form, perform the following instructions:

1. In the `RSSVWorkOrder` DAC, add the unbound `Selected` data field, as shown in the following code.

```
#region Selected
public abstract class selected : PX.Data.BQL.BqlBool.Field<selected> { }
[PXBool]
[PXUIField(DisplayName = "Selected")]
public virtual bool? Selected { get; set; }
#endregion
```

2. Rebuild the project.
3. For the `RS501000.aspx` page, specify the following settings:
   - `PrimaryView` of the datasource control: `WorkOrders`
   - `DataMember` of the grid control: `WorkOrders`

- `SkinID`: `Inquire`

4. Create grid columns for the following fields of the `RSSVWorkOrder` DAC, and arrange them in the following order:
   - `Selected`
   - `OrderNbr`
   - `Description`
   - `ServiceID`
   - `DeviceID`
   - `Priority`
   - `Assignee`

   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.

5. For the grid control, specify the following required properties:
   - `AllowPaging`: `True`
   - `AdjustPageSize`: `Auto`
6. Adjust the size and appearance of the columns as follows:
   a. For the `Selected` column, set the following property values:
      - `Type`: `CheckBox`
      - `AllowCheckAll`: `True`
      - `TextAlign`: `Center`
   b. Specify `Width="140"` for the columns that correspond to the `Service` and `Device` fields.
7. Save your changes.
8. Publish the customization project.

Step 1.1.5: Testing the Processing Form

In this step, you will test the Assign Work Orders (RS501000) form.

Testing the Form

Do the following to test the processing form:

1. On the Repair Work Orders (RS301000) form, open the *000001* repair work order and click **Remove Hold**. Do the same with the *000002* repair work order.
2. On the Assign Work Orders (RS501000) form, make sure that two work orders are displayed on the form, as shown in the following screenshot. These are the work orders that have been created with the publication of the customization project in *Initial Configuration*. Notice that these work orders do not have assignees specified.

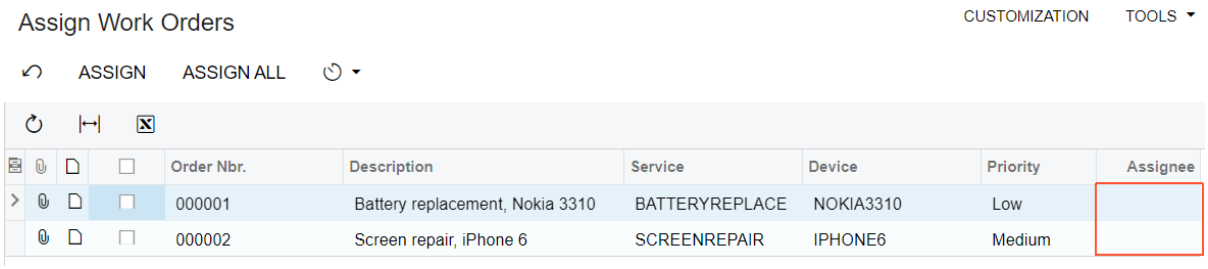


Figure: Two work orders

3. On the Repair Work Orders (RS301000) form, specify the following assignees for the *000001* and *000002* repair work orders, and save your changes to each order:
   - For the *000001* repair work order: *Beauvoir, Layla*
   - For the *000002* repair work order: *Baker, Maxwell*
4. Create a repair work order with the following settings:
   - **Customer ID**: *C000000001*
   - **Service**: *Battery Replacement*
   - **Device**: *Nokia 3310*
   - **Description**: `Battery replacement, Nokia 3310`
5. Save the work order, click **Remove Hold**, and make sure the work order has the *Ready for Assignment* status.
6. Return to the Assign Work Orders (RS501000) form. Select all data records in the table by selecting the check box in the header of the first column (with the unlabeled check box). (In a table with multiple pages of records, selecting the check box in the column header selects the check box of only the rows on the current page.)
7. On the form toolbar, click the **Cancel** button, which clears the selected check boxes and refreshes the list of work orders on the form.

   When you click the **Cancel** button, the system displays the message that is shown in the following screenshot. Click **OK** to close the message and proceed. You will cause the system to suppress this message in *Lesson 1.2: Adding Filtering Parameters to the Processing Form*.

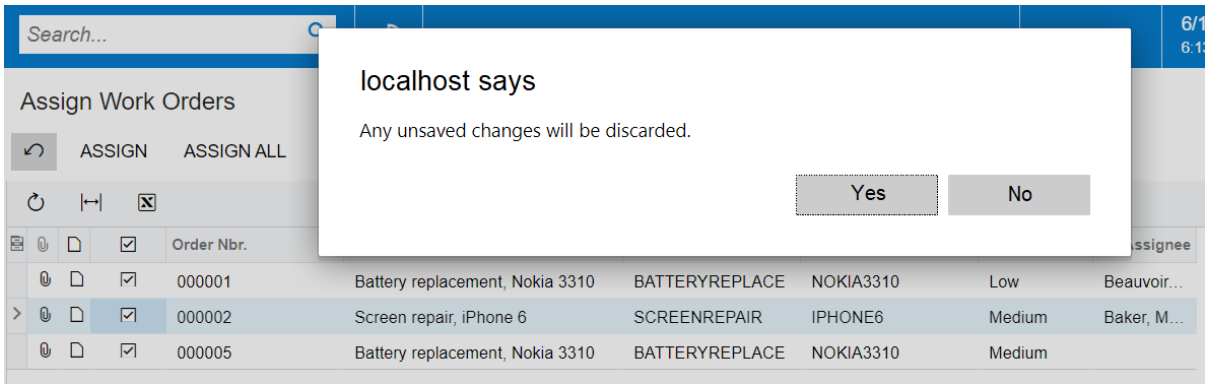


Figure: Message

8. Select the check box in the first column for the *000001* work order and click **Assign** on the form toolbar. The **Processing** dialog box is displayed, which shows the progress and then the result of the operation, as shown in the following screenshot. Close the dialog box.

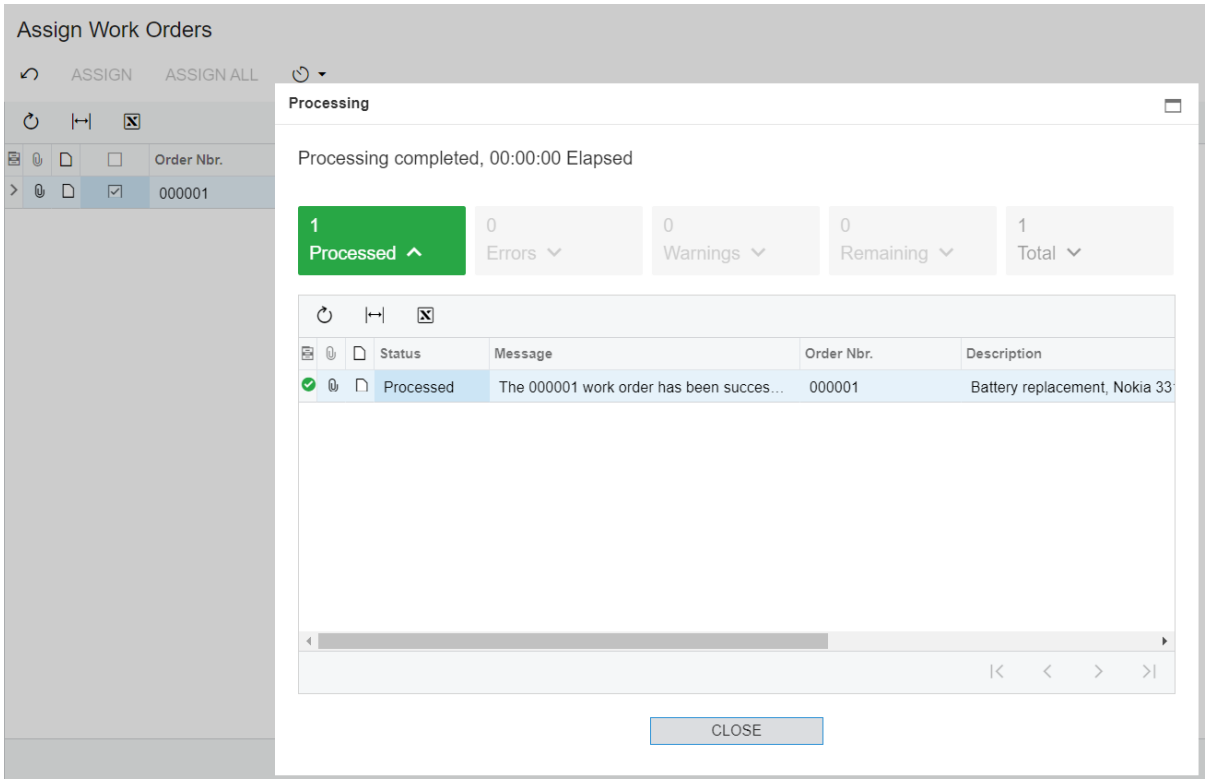


Figure: The Processing dialog box

9. On the Repair Work Orders form, make sure that the *000001* work order now has the *Assigned* status and that the assignee remains *Beauvoir, Layla* (it has not been changed to the default assignee), as shown in the following screenshot.

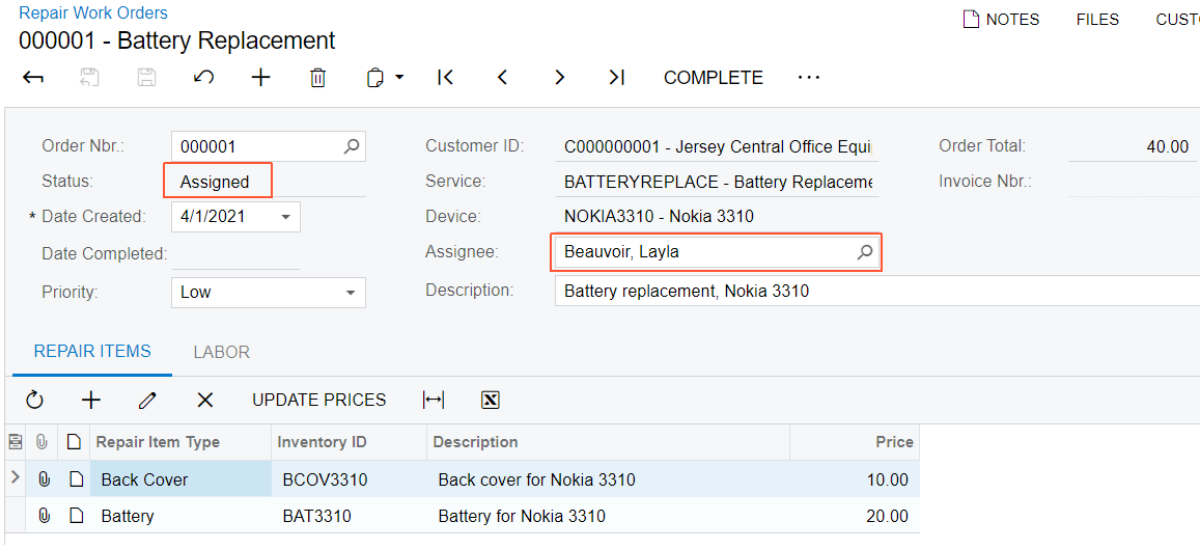


Figure: Assigned work order

10. On the Assign Work Orders form, make sure that two work orders (*000002* and *000005*) are displayed on the form. Click **Assign All** on the form toolbar. The **Processing** dialog box shows that two records have been processed (see the following screenshot).

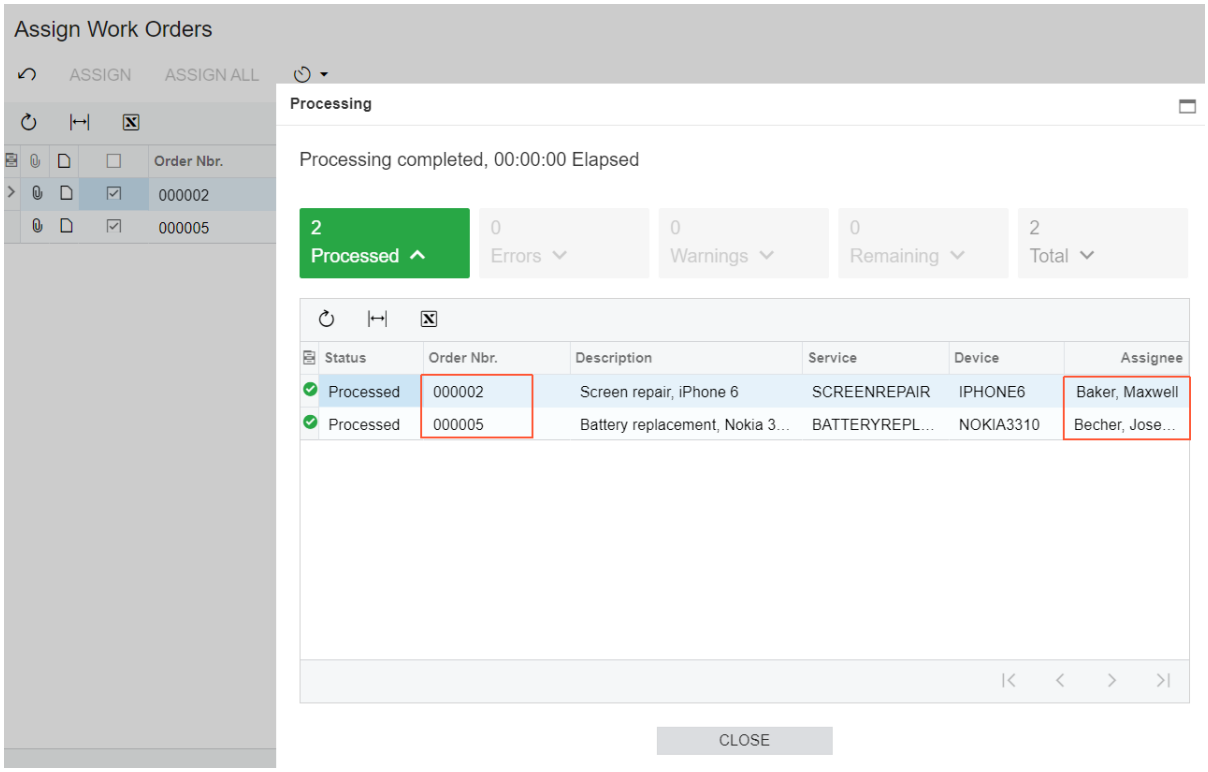


Figure: Assigned work orders

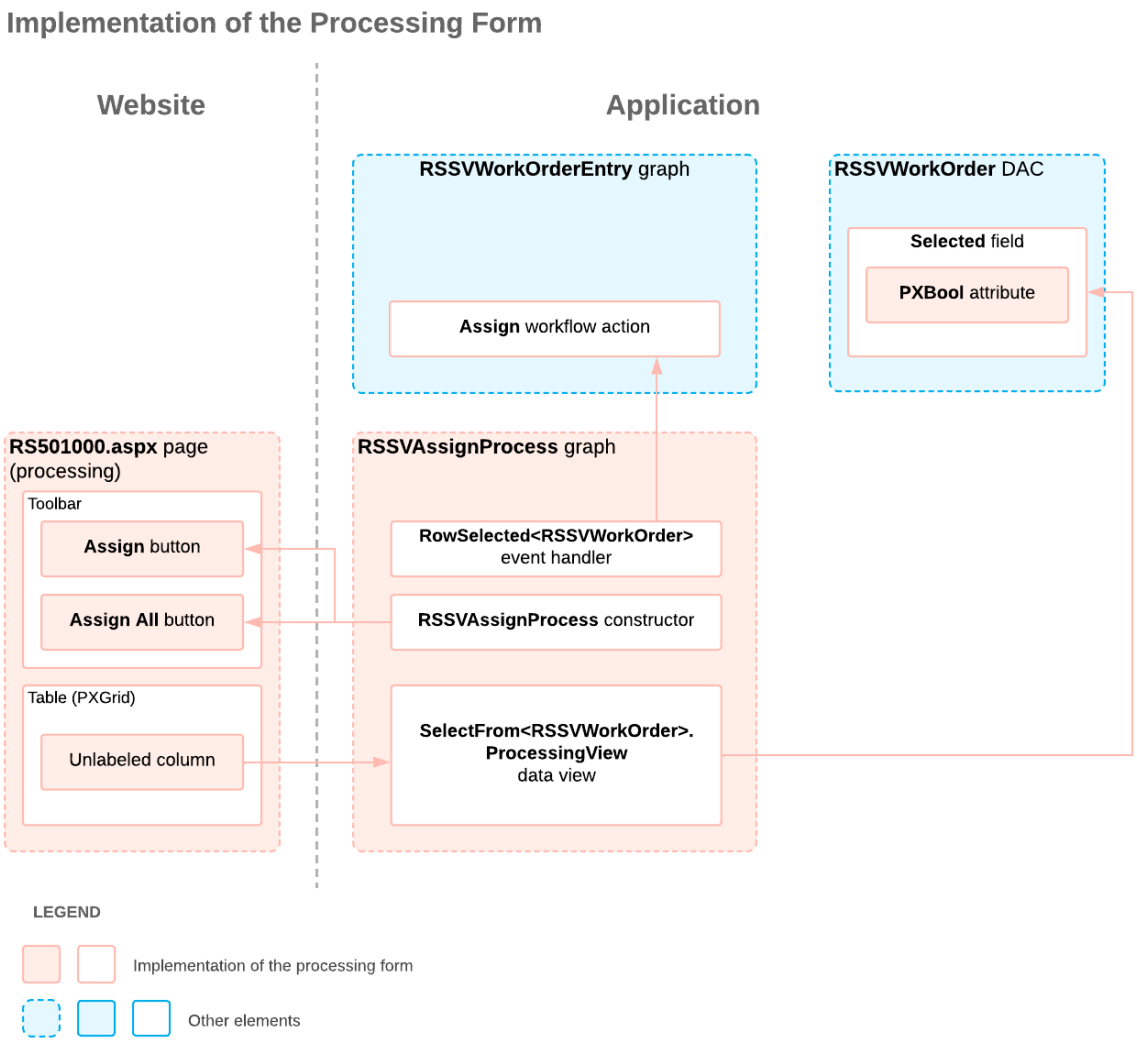
Make sure that for the *000002* work order, the assignee is *Baker, Maxwell*, which has been specified in the work order. For the *000005* work order, the assignee is *Becher, Joseph*, which is the default assignee specified on the Repair Work Order Preferences (RS101000) form.

Lesson Summary

In this lesson, you have learned how to create a simple processing form that displays data to be processed and provides processing actions. For the processing form, you have defined:

- In the processing graph, the specific `SelectFrom<Table>.ProcessingView` (derived from `PXProcessingBase`) data view type to provide data records for the form.
- In the graph constructor, the names of the processing buttons.
- In the `RowSelected` event handler, the workflow action to be used for processing.
- In the DAC, the unbound `Selected` data field, which is used to indicate the records to be processed.
- In the ASPX page, the column in the grid for the `Selected` data field.

The following diagram shows the elements that you have implemented or modified for the processing form.



Review Questions

1. Which type of data view can you use to define a data view for the records to be processed in a processing graph?
   a. `SelectFrom<Table>.View`
   b. `SelectFrom<Table>.ProcessingView`
   c. `PXProcessing<Table>.View`
2. Select all correct statements about the graph constructor of a processing form.
   a. You must specify the processing delegate in the graph constructor of any processing graph.
   b. In the graph constructor, you can specify the custom names of the processing actions.
   c. In the graph constructor, you should not specify the workflow action by using the `SetProcessWorkflowAction<>()` method.

Answer Key

1. **b**
2. **b**, **c**

Additional Information: Processing Dialog Box

In this lesson, you have defined the Assign Work Orders (RS501000) processing form. This form displays the **Processing** dialog box during the assignment operation. Making changes to this dialog box, such as adding a custom button to this dialog box, is outside of the scope of this course but may be useful to some readers.

Adding a Button to the Processing Dialog Box

When a processing operation is started, all elements of the processing form become unavailable. If you need to make a button from the processing form available during processing, you have to add this button to the processing dialog box, as described in *To Add a Button to the Processing Dialog Box*.

Hiding the Processing Dialog Box

You can turn off the displaying of the processing dialog box and instead display the progress and the result of the processing on the form toolbar. For details, see *To Not Display the Processing Dialog Box*.

Additional Information: Parallel Processing

Suppose that you need to implement processing of items on a custom processing form. On this form, users process large lists of items and all of these items can be processed independently. To speed up processing of these items, you can implement parallel processing with Acumatica Framework. If you turn on the parallel processing, the list of records that should be processed is split into batches and is processed in multiple threads.

Acumatica Framework has its own threading subsystem. We do not recommend that you mix it with the default one from .Net.

For details about parallel processing, see *PXParallelProcessingOptions Class*.

Lesson 1.2: Adding Filtering Parameters to the Processing Form

In this lesson, you will modify the Assign Work Orders (RS501000) processing form so that it has filtering parameters, which a user can use to filter the repair work orders in the table on the form. You will also define the **Assignee** column of the table to be editable, so that a user of the form can use this column to select an assignee for any listed repair work order.

Description of the Form Elements

The Summary area of the form will contain the following filtering parameters:

- **Priority**: If a user selects a value in this box, the table on the form displays only the repair work orders with this priority. If no value is selected, repair work orders with all priority values are displayed in the table.

- **Minimum Number of Days Unassigned**: If a user types a number in this box, the table on the form displays only the repair work orders that have been unassigned for a number of days that is greater than or equal to the specified value.
- **Service**: If a user selects a value in this box, the table on the form displays only the repair work orders in which the specified service is selected.

You will also add the **Number of Days Unassigned** column to the table on the form. The column will display the number of days the repair work order has been unassigned. This value will not be stored in the database; it will instead be calculated from the date when the repair work order has been created with the `PXDBCalced` attribute.

In the end of the lesson, the form will look as shown in the following screenshot.

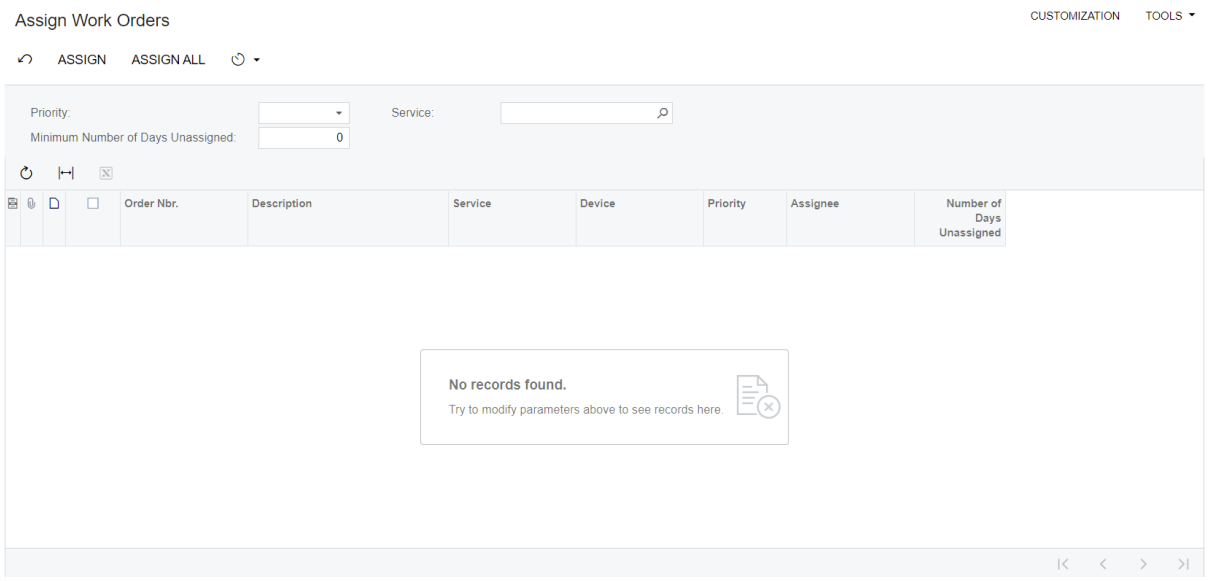


Figure: The Assign Work Orders form with the filter

Lesson Objectives

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

- Create processing pages with filtering parameters
- Use the `PXDBCalced` attribute

Step 1.2.1: Extending the DAC with a New Field (Using PXDBCalced)

In this step, you will extend the `RSSVWorkOrder` class with an additional DAC field that is specific to the processing form.

You will add the `TimeWithoutAction` field. The value of this field is calculated during the retrieval of each `RSSVWorkOrder` record from the database as the difference between the value of the `RSSVWorkOrder.DateCreated` field and the current date, for which you will use the `Now` BQL constant. To calculate the value, you will use the `PXDBCalced` attribute. In the expression calculated by this attribute, you can use only the fields of the same DAC. For more information on the `PXDBCalced` attribute, see *Ad Hoc SQL for Fields*.

Extending the RSSVWorkOrder DAC with the New Field

In the `RSSVWorkOrder.cs` file, add the new field as follows:

1. In the `RSSVWorkOrder` class, define the `TimeWithoutAction` field, which holds the number of days that has passed from the date when the repair work order was created.

```
#region TimeWithoutAction
[PXInt]
[PXDBCalced(
    typeof(RSSVWorkOrder.dateCreated.Diff<Now>.Days),
    typeof(int))]
[PXUIField(DisplayName = "Number of Days Unassigned")]
public virtual int? TimeWithoutAction { get; set; }
public abstract class timeWithoutAction :
    PX.Data.BQL.BqlInt.Field<timeWithoutAction>
{ }
#endregion
```

2. Build the project.

Related Links

- *Ad Hoc SQL for Fields*

Step 1.2.2: Defining the Filter DAC

In this step, you will define the `RSSVWorkOrderToAssignFilter` DAC, which will be used to display filtering parameters on the Assign Work Orders (RS501000) form. You will define the DAC as follows:

- The DAC will contain three fields (`ServiceID`, `TimeWithoutAction`, and `Priority`) that correspond to the filtering parameters.
- The DAC will contain only unbound fields because you do not read the values of the parameters from the database.
- You will not define any key fields in the DAC because the DAC will work with only one data record.

You will assign the `PXHidden` attribute to the filter DAC because you do not need this DAC to be used in generic inquiries and reports.

Defining the Filter DAC

To define the filter DAC, do the following:

1. In the `RSSVAssignProcess` graph, define the `RSSVWorkOrderToAssignFilter` data access class as follows.

```
[PXHidden]
public class RSSVWorkOrderToAssignFilter : IBqlTable
{
    #region Priority
    [PXString(1, IsFixed = true)]
    [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

            #region TimeWithoutAction
            [PXInt]
            [PXUnboundDefault(0)]
            [PXUIField(DisplayName = "Minimum Number of Days Unassigned")]
            public virtual int? TimeWithoutAction { get; set; }
            public abstract class timeWithoutAction :
                PX.Data.BQL.BqlInt.Field<timeWithoutAction>
            { }
            #endregion

            #region ServiceID
            [PXInt()]
            [PXUIField(DisplayName = "Service")]
            [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. Build the project.

Step 1.2.3: Defining the Data Views (with PXFilter and ProcessingView.FilteredBy)

In this step, you will prepare the graph members that provide data for the form.

To add filtering capabilities to the Assign Work Orders (RS501000) processing form, you will define two data views:

- The data view of the `PXFilter` type, which provides the filtering parameters for the processing form. For more information on defining filtering parameters, see *Data View for the Filtering Parameters*.

  Avoid using the `PXFilter` data view type with DACs that have at least one key field defined—that is, DACs that contain fields having the `IsKey=true` parameter in the type attribute.

- The data view of the `ProcessingView.FilteredBy<Table>` type, which selects the repair work orders that meet the criteria specified by the filtering parameters. For more information on data view types for processing forms, see *Creation of Processing Forms*.

In the graph constructor, you will make the values in the **Assignee** column of the table editable. Because you have specified the `Inquiry` skin ID for the table in ASPX (in *Step 1.1.4: Creating Controls for the Processing Form*), the

columns of the table are not defined as being editable. You will enable the editing of the column in the graph constructor (instead of in `RowSelected` event handler) because the UI presentation logic of this column doesn't depend on the particular values of the data record.

You will also override the `IsDirty` property of the graph to make the `IsDirty` property always return `false`. This disables the dialog box that confirms that a user wants to leave the form. This dialog box appears when a user attempts to close the form if there are unsaved changes in the cache objects for the form. (You have seen this dialog box in *Step 1.1.5: Testing the Processing Form*.) Because you have the filtering parameters on the form, which a user can modify, and the editable **Assignee** column and the column with the unlabeled check box, you need to override this property to omit the dialog box. A `False` value in the `IsDirty` property of the graph means that there are no unsaved changes on the form, and the dialog box never appears. This behavior makes sense on processing forms, which are not intended for data entry or editing.

Defining the Data Views

Do the following:

1. In the `RSSVAssignProcess` graph, define the `Filter` data view of the `PXFilter` type (as shown below), which provides the filtering parameters for the processing form.

```
public PXFilter<RSSVWorkOrderToAssignFilter> Filter;
```

2. Replace the definition of the `Cancel` action so that the action uses the filter DAC.

```
public PXCancel<RSSVWorkOrderToAssignFilter> Cancel;
```

3. Replace the definition of the `WorkOrders` data view with the following of the `ProcessingView.FilteredBy<Table>` type, which selects repair work orders that match the values of the filtering parameters.

```
public SelectFrom<RSSVWorkOrder>.
    Where<RSSVWorkOrder.status.IsEqual<
        RSSVWorkOrderWorkflow.States.readyForAssignment>.
        And<RSSVWorkOrder.timeWithoutAction.IsGreaterEqual<
            RSSVWorkOrderToAssignFilter.timeWithoutAction.
                FromCurrent>.
        And<RSSVWorkOrder.priority.IsEqual<
            RSSVWorkOrderToAssignFilter.priority.FromCurrent>.
            Or<RSSVWorkOrderToAssignFilter.priority.FromCurrent.
                IsNull>>.
        And<RSSVWorkOrder.serviceID.IsEqual<
            RSSVWorkOrderToAssignFilter.serviceID.FromCurrent>.
            Or<RSSVWorkOrderToAssignFilter.serviceID.FromCurrent.
                IsNull>>>>.
   OrderBy<RSSVWorkOrder.timeWithoutAction.Desc,
       RSSVWorkOrder.priority.Desc>.
   ProcessingView.
   FilteredBy<RSSVWorkOrderToAssignFilter> WorkOrders;
```

4. Replace the definition of the `RowSelected` event handler so that the event handler uses the filter DAC.

```
protected virtual void _(Events.RowSelected<
    RSSVWorkOrderToAssignFilter> e)
{
    WorkOrders.SetProcessWorkflowAction<RSSVWorkOrderEntry>(
        g => g.Assign);
}
```

5. In the graph constructor, enable editing for the `Assignee` data field, as shown in the following code.

```
            PXUIFieldAttribute.SetEnabled<RSSVWorkOrder.assignee>(
                WorkOrders.Cache, null, true);
```

6. Override the `IsDirty` property of the graph, as the following code shows.

```
        public override bool IsDirty
        {
            get
            {
                return false;
            }
        }
```

7. Rebuild the project.

Related Links

- *Creation of Processing Forms*
- *Data View for the Filtering Parameters*

Step 1.2.4: Adjusting the ASPX Page (with SyncPosition and AutoRefresh)

In this step, you will adjust the ASPX page of the Assign Work Orders (RS501000) form to display the filter and data to be processed.

Because the grid includes the **Assignee** column, in which each cell is a selector control that displays the list of records that depends on the currently selected row in the grid, you will specify the `SyncPosition` property of the grid and the `AutoRefresh` property of the selector control. The `SyncPosition` property makes the system set the `Current` property of the cache object to a row selected by the user in the grid. The `AutoRefresh` property of a selector control causes the list of records in the control to be refreshed automatically every time it is opened by the user. These properties are required for the synchronization of the list of records in the selector control with the currently selected row in the grid because the data displayed in the selector control depends on the selected row.

Adjusting ASPX

Adjust the ASPX of the form as follows:

You can perform the following instructions in the Screen Editor of the Customization Project Editor or edit the ASPX code of the form directly in Visual Studio. For details on working with the Screen Editor or editing the ASPX code in Visual Studio, see the *T200 Maintenance Forms* training course. The instructions below are presented in general terms to accommodate both methods.

1. For the datasource control of `RS501000.aspx`, change the value of `PrimaryView` to `Filter`.
2. Add the form control, set its `DataMember` property to `Filter`, and its `Width` property to `100%`.
3. In the form control, add input controls for the `Priority`, `TimeWithoutAction`, and `ServiceID` fields, and set the `CommitChanges` property to `True` for these controls.
4. Split the controls into two columns and adjust the size of controls so that the labels are fully visible. You can use `LabelsWidth="XM"` for the first column.
5. For the grid control, specify the following values of the properties:
   - `DataMember`: `WorkOrders`
   - `SyncPosition`: `True`

> ⚠ In this case, setting the `SyncPosition` property to True is optional because the `PXSelector` attribute attached to the `Assignee` field does not depend on the values of other `RSSVWorkOrderToAssign` fields.

6. Add a column to the grid for the `TimeWithoutAction` data field and specify `Width="100"` for the column.
7. For the `Assignee` column, specify the following values of the properties:
   - `CommitChanges`: `True`
   - `AutoRefresh`: `True`

   > ⓘ This property is specified for the `PXSelector` control inside `RowTemplate`. For details about how to specify the `AutoRefresh` property, see *Step 2.2.1: Restricting the Values of a Field (with PXRestrictor)* in the *T210 Customized Forms and Master-Detail Relationship* training course.
8. Publish the customization project.

Step 1.2.5: Testing the Filter

In this step, you will test the filtering parameters of the Assign Work Orders (RS501000) form. Do the following:

1. On the Repair Work Orders (RS301000) form, create three repair work orders with the settings specified in the following table. Save each order and click **Remove Hold**.

	Work Order *000006*	**Work Order *000007***	**Work Order *000008***
Customer ID	*C000000001*	*C000000002*	*C000000001*
Service	*Battery Replacement*	*Screen Repair*	*Battery Replacement*
Device	*Nokia 3310*	*Samsung Galaxy S4*	*Motorola RAZR V3*
Assignee	*Beauvoir, Layla*	Empty	*Baker, Maxwell*
Priority	*High*	*Medium*	*Medium*
Description	`Test order`	`Test order`	`Test order`

   The created work orders have the *Ready for Assignment* status and have been assigned the *000006*, *000007*, and *000008* order numbers (if you have created work orders only by following the instructions in the training guides of the *T* courses).
2. On the Assign Work Orders form, test the filtering parameters as follows:
   a. Make sure that the three work orders you have created are displayed on the form.
   b. In the **Priority** box in the Summary area, select *High*. Make sure the *000006* work order is displayed in the table, as shown in the following screenshot.

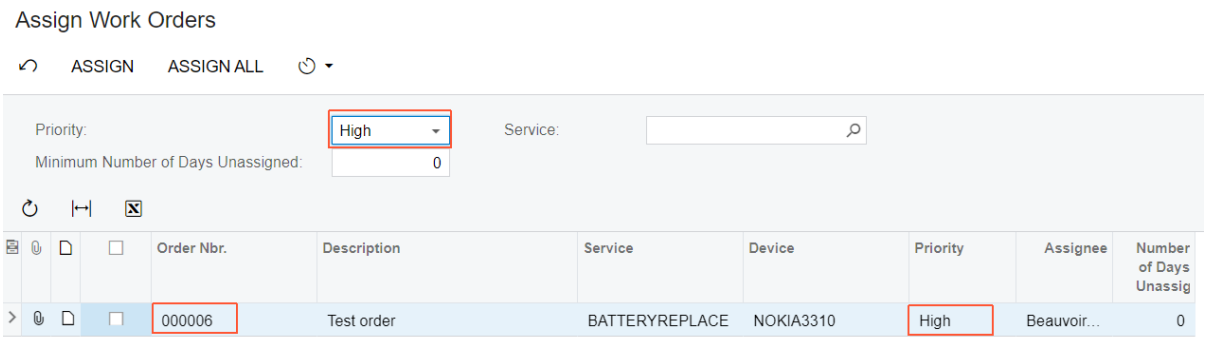


Figure: Work orders with the High priority

c. Clear the filter by clicking Cancel on the form toolbar.

d. In the **Minimum Number of Days Unassigned** box, type `1`. No work orders are displayed in the table if you have not created work orders apart from the instructions of the guide.

e. Change the value in the **Minimum Number of Days Unassigned** box to `0`. Three work orders are displayed.

f. In the **Service** box, select *Battery Replacement*. The *000006* and *000008* work orders are displayed in the table.

g. In the **Priority** box, select *Medium*. Only the *000008* work order is displayed in the table.

h. On the form toolbar, click **Assign All**. The processing dialog box indicates that the *000008* work order is processed. Make sure the work order has the *Assigned* status and is assigned to *Baker, Maxwell* (which you have selected during creation), as shown in the following screenshot.

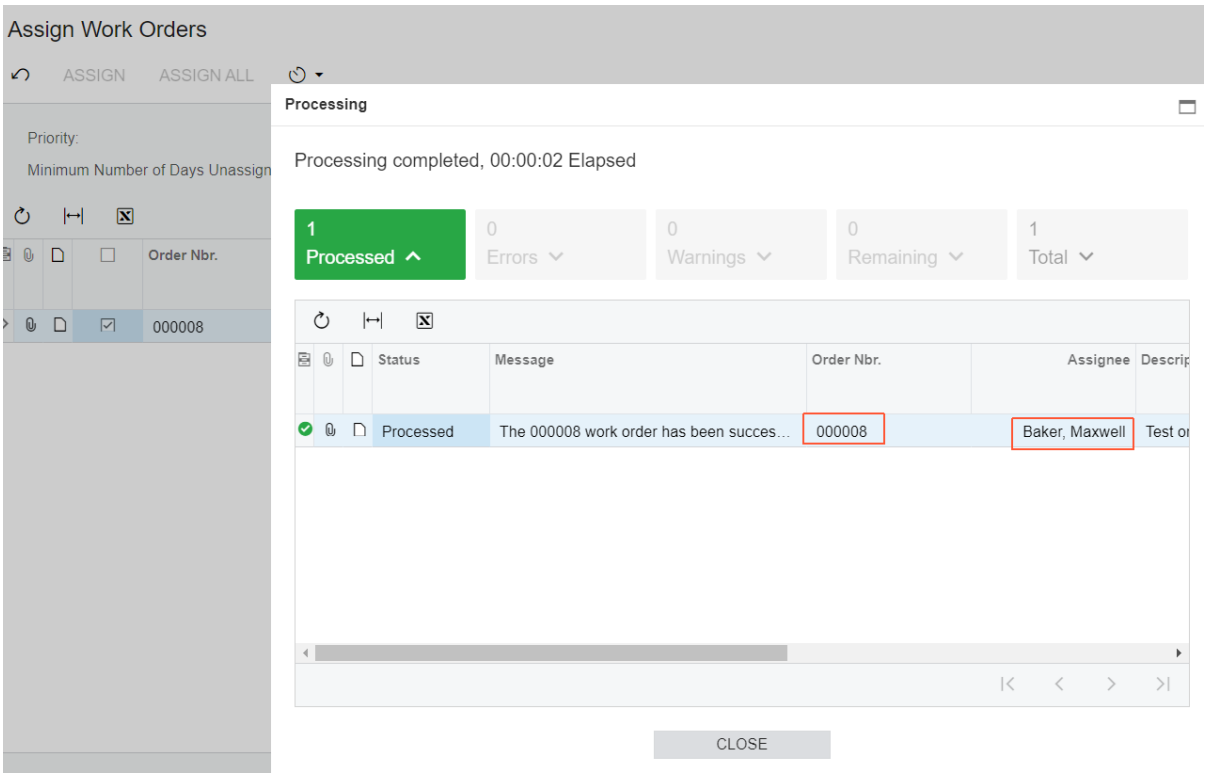


Figure: The assigned work order

3. Test the **Assignee** box on the Assign Work Orders form as follows:

   a. Clear all filters. Two repair work orders (*000006* and *000007*) are displayed.

b. For the *000007* work order, in the **Assignee** box, select the *Beauvoir, Layla* employee.

c. On the form toolbar, click **Assign All**. The processing dialog box shows that two repair work orders are processed. Make sure that the assignees are as shown in the following screenshot.

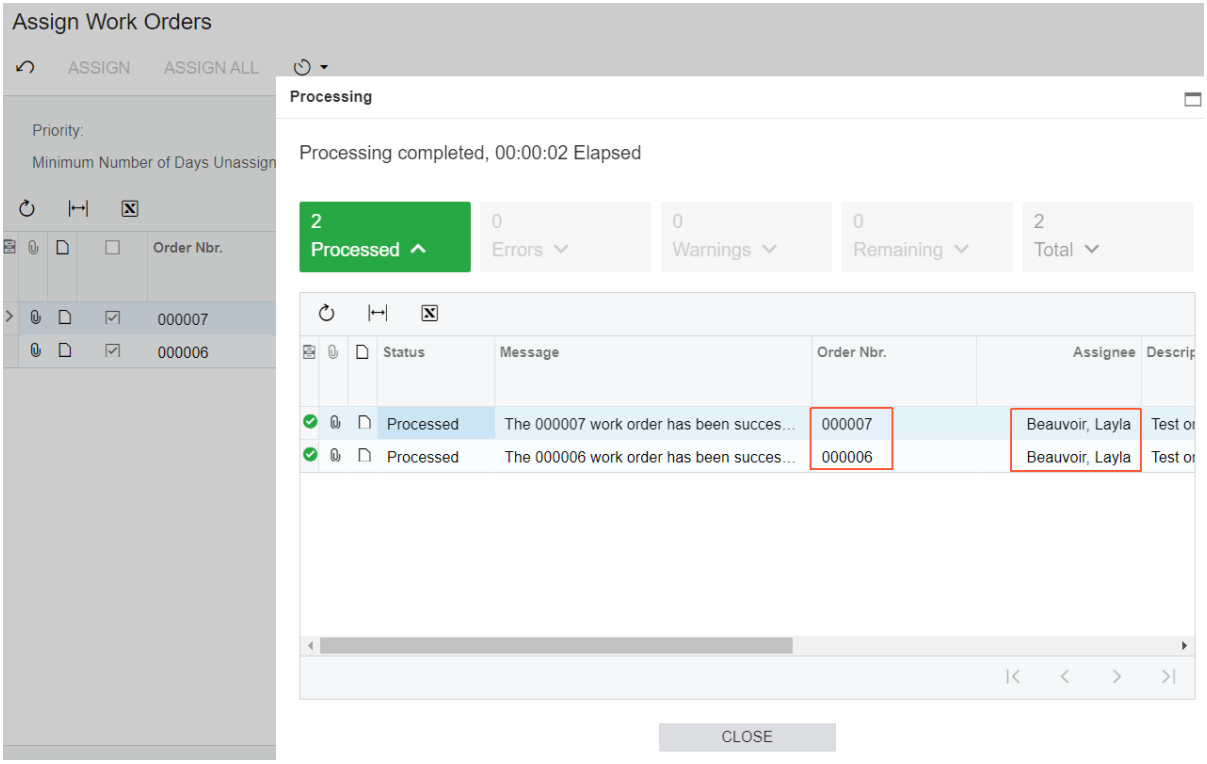


Figure: Two assigned work orders

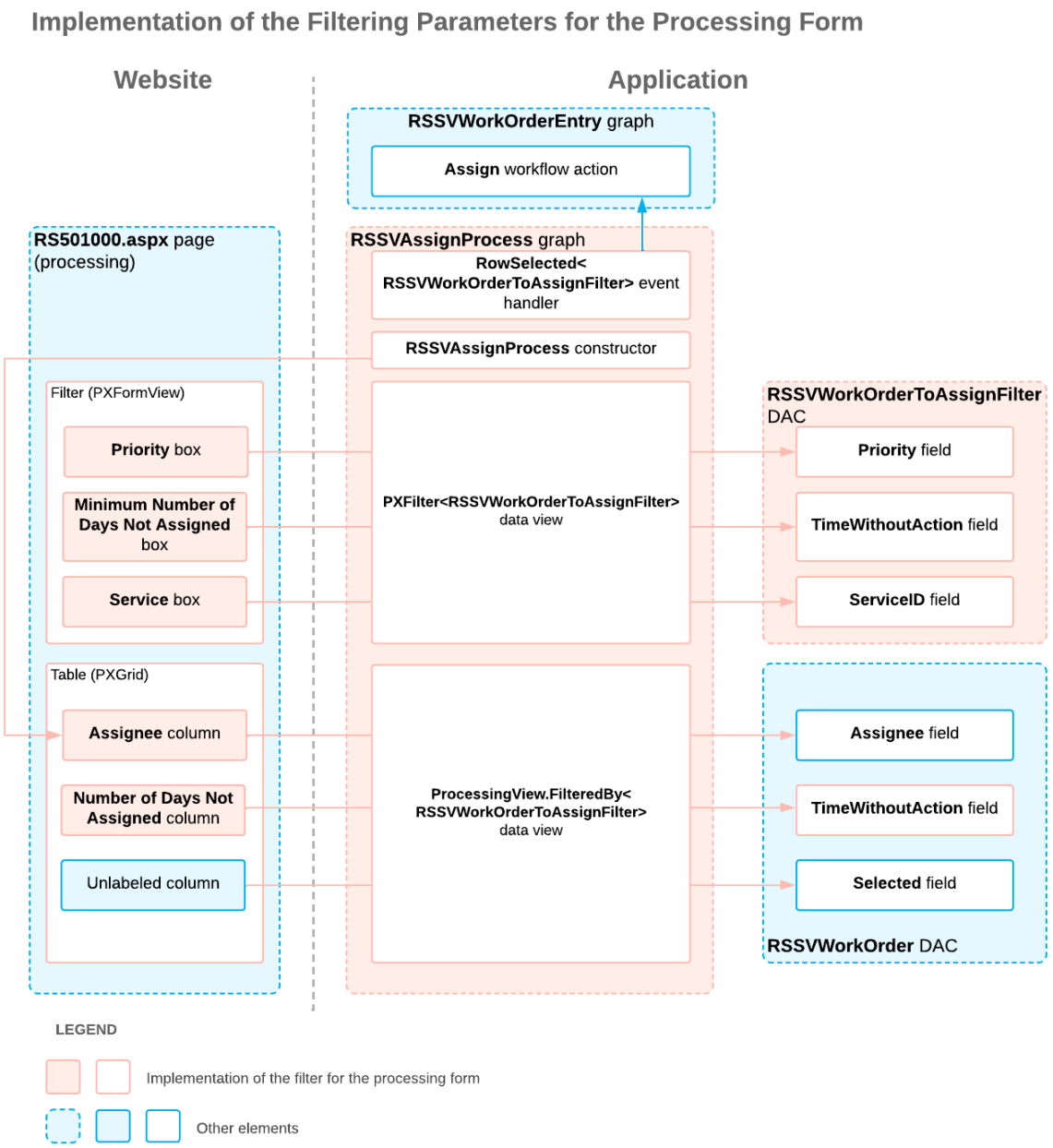
Lesson Summary

In this lesson, you have learned how to define processing forms with filtering parameters. Because you have already implemented the processing method in the previous lesson, to add a filter to the processing form, you have completed the following steps:

1. Prepared the DAC that provides records for processing.
2. Defined the DAC that provides the filtering parameters for the processing form.
3. In the graph, defined the following data views:
   - The data view of the `PXFilter` type, which provides data for the filter
   - The data view of the `ProcessingView.FilteredBy<Table>` type, which retrieves records for possible processing
4. In the graph, modified the `RowSelected` event handler so that it uses the primary DAC of the primary data view, which is the `RSSVWorkOrderToAssignFilter` DAC.

You have also used the `PXUIFieldAttribute.SetEnabled<>()` method in the graph constructor to enable editing for the `Assignee` data field.

The following diagram shows the changes implemented in this lesson.



Review Questions

1. Which data view type should you use for the grid on a processing form with filtering parameters?
   a. `SelectFrom<Table>.ProcessingView`
   b. `SelectFrom<Table>.ProcessingView.FilteredBy<FilterTable>`
   c. `SelectFrom<Table>.View`
   d. `PXFilter<Table>`
2. Which data view type should you use for the filter on a processing form?
   a. `SelectFrom<Table>.ProcessingView`

b. `SelectFrom<Table>.ProcessingView.FilteredBy<FilterTable>`
c. `SelectFrom<Table>.View`
d. `PXFilter<Table>`

Answer Key

1. **b**
2. **d**

Part 2: Update of Data with a Custom Accumulator Attribute

The functionality of the Assign Work Orders (RS501000) form that has been implemented so far is not enough for the Smart Fix company. In accordance with the specifications of the managers of the company, the repair work orders that are processed on the Assign Work Orders form should be automatically assigned to the employee with the smallest number of repair work orders assigned. If there are multiple employees with the smallest number of repair work orders assigned, the work orders will be assigned to the first of these employees selected from the database.

To bring the form closer to these specifications, in this part of the guide, you will do the following:

- Modify the Assign Work Orders form so that it contains information about the number of work orders assigned to potential assignees listed in the table
- Implement a custom `PXAccumulator` attribute that will count the number of repair work orders assigned to each employee and update this number in the database during processing on the Assign Work Orders form

The use of accumulator attributes is a specific Acumatica Framework technique for fields that are updated frequently (and often concurrently by multiple users). An accumulator attribute changes the SQL query that is executed when the data is updated in the database. You can use an accumulator attribute in either of the following cases:

- To update a field or multiple fields of a data record without checking for the data record version in the database. (In an ordinary update, the framework generates the SQL statement that checks the timestamp column, if this column exists in the table.)
- To define a specific update policy for a column—for instance, to calculate the sum of values in a column on every update. You can also specify restrictions for a column that will be checked by the database during update.

After you complete the lessons of this part, you will be able to test the updated functionality of the form and the way the custom `PXAccumulator` attribute works.

Lesson 2.1: Implementing a Custom PXAccumulator Attribute

In the Smart Fix company, the number of assigned work orders may be updated very often. Multiple users can assign repair work orders to the same employee at the same time. To avoid problems that can be caused by concurrent updates in the database, you will implement a custom attribute derived from the `PXAccumulator` attribute. In this attribute, you will implement the calculation of the number of repair work orders assigned to a particular employee. This attribute will compute the total of the number of repair work orders assigned to each employee and restrict this number of orders for the employee so that the number may be no more than 10. The accumulator attribute modifies the SQL query and guarantees that one repair work order assigned to an employee is not counted multiple times.

Lesson Objectives

In this lesson, you will learn how to implement a custom attribute derived from the `PXAccumulator` attribute.

Step 2.1.1: Preparing the Data

In this step, you will complete—that is, indicate completion in the system of—all repair work orders that have the *Assigned* status on the Repair Work Orders (RS301000) form.

In the database, you have the `RSSVEmployeeWorkOrderQty` table, which you have added in *Initial Configuration* of this course and which will store the number of repair work orders assigned to a particular employee. This table currently contains no data because you have not yet implemented the logic to update data in this table. However, you have a number of repair work orders assigned to employees. Therefore, you need to complete these repair work orders so that none of the employees has repair work orders assigned.

Completing the Repair Work Orders

To complete the work orders, on the Repair Work Orders (RS301000) form, do the following for each of the repair work orders that have the *Assigned* status:

1. Open the repair work order.
2. Click **Complete** on the form toolbar.

The repair work orders you have completed now have the *Completed* status.

Step 2.1.2: Creating a DAC—Self-Guided Exercise

As you completed the *Initial Configuration*, you created the `RSSVEmployeeWorkOrderQty` database table, whose `NbrOfAssignedOrders` column will be updated by the custom accumulator attribute. In this step, you will create a data access class for this table. The ways to create a DAC are described in detail in the *T200 Maintenance Forms* training course.

As you add the `RSSVEmployeeWorkOrderQty` DAC, you will perform the following general actions:

1. You create the `RSSVEmployeeWorkOrderQty` data access class and define its single system field: `LastModifiedDateTime`.

   Database tables that are used exclusively for storing accumulated values usually do not contain audit, timestamp, or `NoteID` columns. The base `PXAccumulatorAttribute` class (on which you are going to base a custom attribute for this DAC) is only capable of handling fields of the `DateTime` type that are decorated with one of the following attributes:

   - `PXDBLastModifiedDateTimeAttribute`
   - `PXDBLastChangeDateTimeAttribute`
   - `PXDBLastModifiedByScreenIDAttribute`
   - `PXDBLastModifiedByIDAttribute`

   Thus the `RSSVEmployeeWorkOrderQty` table which you created in *Step 3: Creating the Database Table* contains only one audit column, `LastModifiedDateTime`.

   For more information about definition of the `LastModifiedDateTime` system field, see *Audit Fields* in the documentation.

2. For the DAC, you specify the `PXHidden` attribute, which indicates that the DAC will not be used for reports or generic inquiries.
3. In the `RSSVEmployeeWorkOrderQty` DAC, define the `UserID` and `NbrOfAssignedOrders` fields and their attributes as follows:
   - Mark the `UserID` field as the key field, as shown in the following code.

```
#region UserID
[PXDBInt(IsKey = true)]
public virtual int? UserID { get; set; }
public abstract class userID : PX.Data.BQL.BqlInt.Field<userID> { }
#endregion
```

- Do not specify any display names for the fields because they will not be displayed in the UI.

Step 2.1.3: Implementing the Accumulator Attribute

In this step, you will create the custom `RSSVEmployeeWorkOrderQtyAccumulator` accumulator attribute for the `RSSVEmployeeWorkOrderQty` DAC. For each employee, the custom attribute will compute the total of the number of assigned work orders and save the value in the `RSSVEmployeeWorkOrderQty.NbrOfAssignedOrders` field. The attribute will be derived from the `PXAccumulator` system attribute. Although the base attribute can also be configured to summarize the values in the `RSSVEmployeeWorkOrderQty.NbrOfAssignedOrders` field, you will use the custom attribute instead of the base one because you need to specify a custom restriction for the number of work orders assigned to an employee (no more than 10 work orders for each employee).

To define the custom accumulator attribute, you will do the following:

- Add the attribute constructor.

  By setting the value of the `_SingleRecord` field in the constructor, you will make the system use single-record update mode. In this mode, the attribute updates the data record independently from the existing data records and does not add any restrictions to future data records. In single-record update mode, the framework generates a specific SQL statement that updates an independent record.
- Implement the `PrepareInsert()` method.

  In the `PrepareInsert()` method, you will define the updating policy for the `NbrOfAssignedOrders` data field of the `RSSVEmployeeWorkOrderQty` DAC. The `PrepareInsert()` method is invoked within the `Persist()` method before the framework generates SQL commands for inserted data records. The fields for which you invoke the `columns.Update()` method are the only fields updated by the attribute. The type parameter of the method specifies the data field to be updated; the first input parameter specifies the value, while the second input parameter defines the updating policy for the data field. You will specify the `Summarize` update policy for the field, which means that the new value is added to the value stored in the database. For detailed information on the update policies of `PXAccumulator` attributes, see *Update of Data with PXAccumulator Attributes*.

  In the `PrepareInsert()` method, you will also specify the restriction that an employee cannot be assigned more than 10 repair work orders. By using the `columns.AppendException()` method, you will specify the restriction and define an exception that is thrown when the restriction is violated. The condition you specify is checked against the resulting value the system gets after adding the new value to the one stored in the database. When you use the `AppendException()` method, the restriction works correctly for both the insertion of a new value and the update of the old one. You will use the `PXComp` enumerator value to specify the type of comparison in the restriction: `PXComp.LE` is less than or equal to. For details about the implementation of restrictions in accumulator attributes, see *Restrictions in the Accumulator Attribute*.

You will add the custom attribute directly to the `RSSVEmployeeWorkOrderQty` DAC, because this class is updated only from code and not through the UI.

If you have a DAC that users can edit through the UI, you cannot assign a `PXAccumulator` attribute directly to this DAC. Instead, you should derive a new DAC from the original one and assign the accumulator attribute to this derived DAC, so that the derived DAC and the original DAC implement the following alternative ways of updating the related table:

- All data fields are updated through the original DAC when a record is edited through the UI.
- The data fields specified in the accumulator attribute are updated through the derived DAC according to the updating policies defined in the accumulator attribute when a record is edited through the code.

Implementing the RSSVEmployeeWorkOrderQtyAccumulator Attribute

To implement the custom accumulator attribute, do the following:

1. In the `Messages.cs` file, add the following constant with the message, which is displayed when the restriction specified in the accumulator attribute is violated.

```
        public const string ExceedingMaximumNumberOfAssingedWorkOrders =
            @"Updating the number of assigned work orders for the employee
            will lead to exceeding of the maximum number of assigned work orders,
            which is 10.";
```

2. In the `RSSVEmployeeWorkOrderQty.cs` file, define the `RSSVEmployeeWorkOrderQtyAccumulator` attribute as follows.

```
    public class RSSVEmployeeWorkOrderQtyAccumulator :
        PXAccumulatorAttribute
    {
        //Specify the single-record mode of update in the constructor.
        public RSSVEmployeeWorkOrderQtyAccumulator()
        {
            _SingleRecord = true;
        }
        //Override the PrepareInsert method.
        protected override bool PrepareInsert(PXCache sender, object row,
            PXAccumulatorCollection columns)
        {
            if (!base.PrepareInsert(sender, row, columns)) return false;
            RSSVEmployeeWorkOrderQty newQty = (RSSVEmployeeWorkOrderQty)row;
            if (newQty.NbrOfAssignedOrders != null)
            {
                // Add the restriction for the value of
                // RSSVEmployeeWorkOrderQty.NbrOfAssignedOrders.
                columns.AppendException(
                    Messages.ExceedingMaximumNumberOfAssingedWorkOrders,
                new PXAccumulatorRestriction<
                    RSSVEmployeeWorkOrderQty.nbrOfAssignedOrders>(
                    PXComp.LE, 10));
            }
            // Update NbrOfAssignedOrders by using Summarize.
            columns.Update<RSSVEmployeeWorkOrderQty.nbrOfAssignedOrders>(
                newQty.NbrOfAssignedOrders,
                PXDataFieldAssign.AssignBehavior.Summarize);
            return true;
        }
    }
```

3. Add the `RSSVEmployeeWorkOrderQtyAccumulator` attribute to the `RSSVEmployeeWorkOrderQty` class, as shown below.

```
    [PXHidden]
    [RSSVEmployeeWorkOrderQtyAccumulator]
    public class RSSVEmployeeWorkOrderQty : IBqlTable
    {        ...
    }
```

4. Build the project.

Related Links

- *Update of Data with PXAccumulator Attributes*
- *Restrictions in the Accumulator Attribute*

Lesson Summary

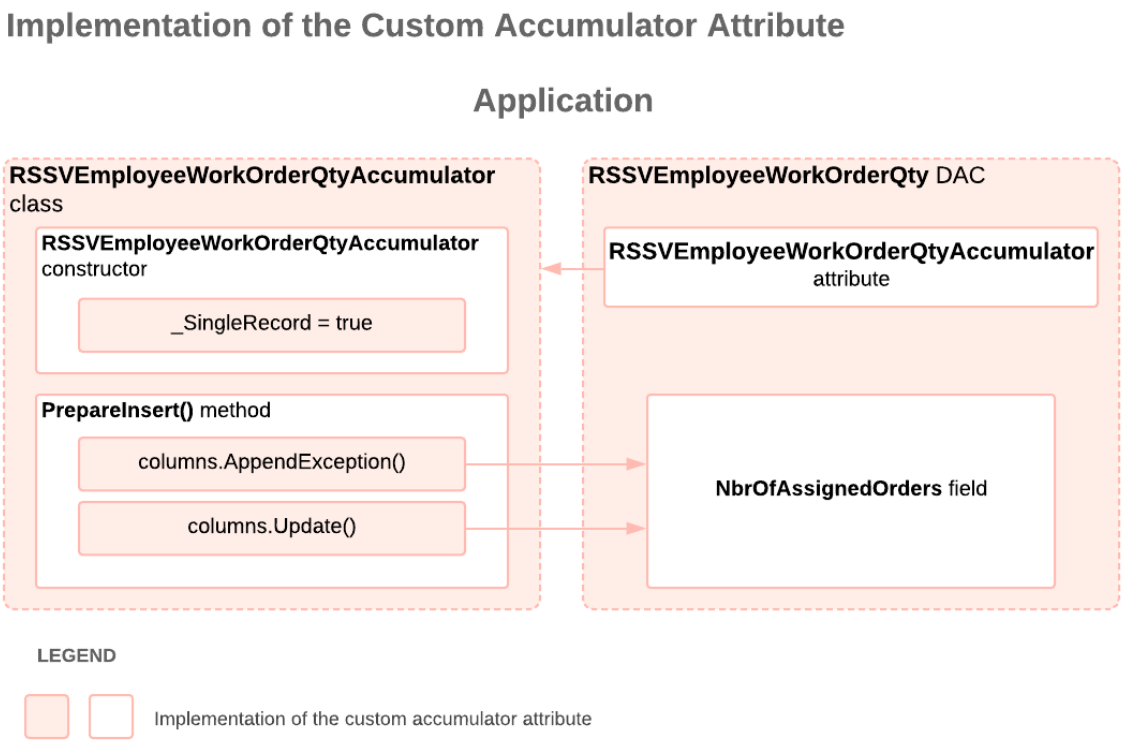
In this lesson, you have learned how to create a custom accumulator attribute to summarize the numbers of assigned work orders for each employee during the assignment or completion of work orders.

In the custom attribute, you have defined the following elements:

- The constructor, in which you have specified the update mode for the records
- The `PrepareInsert()` method, in which you have defined the updating policy for the particular field (the values of this field are summarized) and specified the restriction for the values of this field

You have also assigned the custom accumulator attribute to the DAC that stores the field to be updated by the accumulator attribute.

The following diagram illustrates the implementation of the custom accumulator.



Review Questions

1. Where would you assign a `PXAccumulator` attribute?
   a. To the fields that should be updated by the attribute

   b. To the DAC that contains the field that should be updated by the attribute
   c. To the DAC that contains the field that should be updated by the attribute if this DAC is editable only through the code; otherwise, to a derived DAC
2. How would you specify the fields that should be updated by a `PXAccumulator` attribute?
   a. Assign the attribute to these fields.
   b. Define the fields in the `PrepareInsert()` method by using the `columns.Update()` method.
   c. Define the fields in the attribute constructor by using the `columns.Update()` method.
   d. Define the fields in the `PrepareInsert()` method by using the `columns.AppendException()` method.
   e. Define the fields in the attribute constructor by using the `columns.AppendException()` method.

Answer Key

1. **c**
2. **b**

Lesson 2.2: Modifying the Processing Form to Use the Field Updated by PXAccumulator

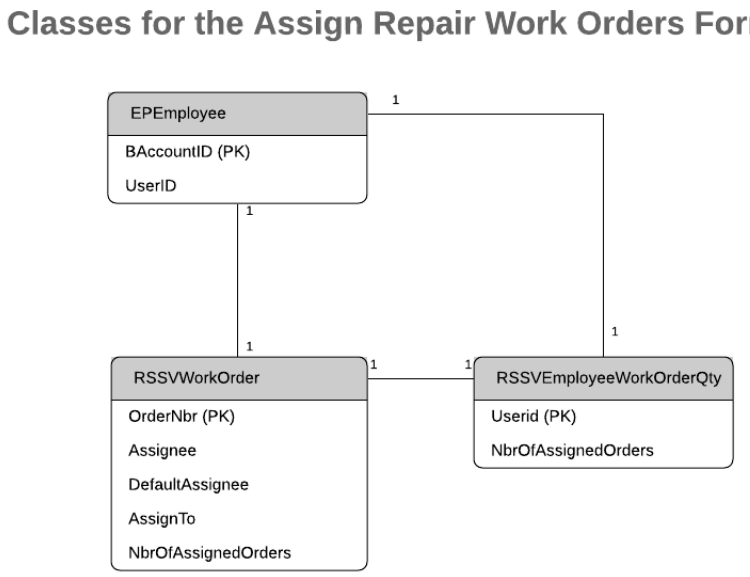
In this lesson, you will modify the Assign Work Orders (RS501000) form so that if no assignee is specified for a repair work order on the Repair Work Orders (RS301000) form, the system detects the default assignee for this repair work order as the employee that has the fewest work orders assigned.

You will also modify the **Assign** and **Complete** actions on the Repair Work Orders (RS301000) form. When a user clicks **Assign**, the number of assigned work orders for the corresponding assignee will be increased. When a user clicks **Complete**, the number of assigned work orders for the assignee will be decreased. These calculations will be performed by the custom accumulator attribute, which you have implemented in the previous lesson.

Database Tables and DACs Used for the Form

In this lesson, you will add the `DefaultAssignee`, `AssignTo`, and `NbrOfAssignedOrders` fields to the `RSSVWorkOrder` DAC (see the diagram below). The values of `DefaultAssignee` and `NbrOfAssignedOrders` are calculated based on the values in the `RSSVEmployeeWorkOrderQty` table, which holds the numbers of repair work orders assigned to employees.

The `RSSVEmployeeWorkOrderQty` table is linked to the `EPEmployee` table by `UserID`. (You have added the `RSSVEmployeeWorkOrderQty` table to the application database in *Initial Configuration* and the corresponding DAC to the customization code in *Lesson 2.1: Implementing a Custom PXAccumulator Attribute*.)



Lesson Objectives

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

- Specify the values of the fields updated by a `PXAccumulator` attribute
- Use the `PXDBScalar` attribute
- Append and replace attributes on a certain DAC field within a particular graph
- Define the external presentation of field values

Step 2.2.1: Extending the DAC with New Fields

In this step, you will add the following new fields to the `RSSVWorkOrder` DAC:

- `DefaultAssignee`: The employee that has the lowest number of assigned repair work orders. You will define the behavior of this field in the next step. For testing purposes (to make sure that the `AssignTo` field is calculated correctly), the `DefaultAssignee` field will be displayed in the table on the Assign Work Orders (RS501000) form and will not be editable. (You will delete the added column later, after testing.)
- `AssignTo`: The employee to which the repair work order will be assigned. You will define the behavior of this field in the next step. The `AssignTo` field will be displayed in the table on the Assign Work Orders form. A user can change the value in this box. In this step, you will modify the constructor of the `RSSVAssignProcess` graph to make the column editable.
- `NbrOfAssignedOrders`: The number of repair work orders that are assigned to the employee specified in the `AssignTo` field. The value that is displayed in this field will be defined from the `RSSVEmployeeWorkOrderQty.NbrOfAssignedOrders` field in the `FieldSelecting` event handler (which will be implemented in *Step 2.2.4: Defining the External Presentation of Field Values (in FieldSelecting)*). This field will be displayed in the UI and its corresponding column cannot be edited.

The **Assignee** column, which displays the value specified for the work order on the Repair Work Orders (RS301000) form, will temporarily remain in the table on the Assign Work Orders form for testing purposes and will not be editable.

Extending the RSSVWorkOrder DAC

Add the new fields to the `RSSVWorkOrder` DAC and edit the other code as follows:

1. In the `RSSVWorkOrder` class, define the `DefaultAssignee` field, as shown in the following code.

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

2. Define the `AssignTo` field, as shown below.

```
#region AssignTo
[PXInt]
[PXUIField(DisplayName = "Assign To")]
public virtual int? AssignTo { get; set; }
public abstract class assignTo : PX.Data.BQL.BqlInt.Field<assignTo> { }
#endregion
```

3. Define the `NbrOfAssignedOrders` field, as the following code shows.

```
#region NbrOfAssignedOrders
[PXInt]
[PXUIField(DisplayName = "Number of Assigned Work Orders")]
public virtual int? NbrOfAssignedOrders { get; set; }
public abstract class nbrOfAssignedOrders :
    PX.Data.BQL.BqlInt.Field<nbrOfAssignedOrders>
{ }
#endregion
```

4. In the constructor of the `RSSVAssignProcess` graph, replace the `Assignee` field with the `AssignTo` field of the `RSSVWorkOrder` DAC. The resulting code of the constructor is shown in the following code.

```
public RSSVAssignProcess()
{
    WorkOrders.SetProcessCaption("Assign");
    WorkOrders.SetProcessAllCaption("Assign All");
    PXUIFieldAttribute.SetEnabled<RSSVWorkOrder.assignTo>(
        WorkOrders.Cache, null, true);
}
```

5. Build the project.

Related Links

- *Ad Hoc SQL for Fields*

Step 2.2.2: Replacing Field Attributes (with PXDBScalar and PXUnboundDefault in CacheAttached)

In this step, you will add attributes that calculate values of the `DefaultAssignee` and `AssignedTo` fields of the `RSSVWorkOrder` DAC. Because you need these calculations only for the Assign Work Orders (RS501000) form, you will add these attributes by using the `CacheAttached` event handler.

Field Attributes

For the system to calculate the value of the `DefaultAssignee` field, you need to use the `PXDBScalar` attribute. The `PXDBScalar` attribute selects the first record that matches the query specified in the attribute. In the query, you will select records ordered by the number of assigned work orders ascending.

The system sets the value of the `AssignedTo` field to the employee selected for the work order on the Repair Work Orders (RS301000) form (if the value is not `null`) or to the default assignee specified in the `DefaultAssignee` field (if the value selected on the Repair Work Orders form is `null`). You will define this behavior by using the `PXUnboundDefault` attribute.

To display the employee name instead of its ID (which is an integer value) and display the selector for the column if it is editable, you will assign the `Owner` attribute to the `DefaultAssignee` and `AssignTo` fields.

Replacement of Attributes

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.

In the `RSSVAssignProcess` graph, you will add the attributes to the `RSSVWorkOrder` DAC fields by using the `CacheAttached` event handlers of these fields. These attributes will be used for the `RSSVWorkOrder` DAC fields only on the Assign Work Orders (RS501000) form.

Instead of complete replacement of attributes, you will add the needed attributes to the fields by including the `PXMergeAttributes` attribute in the list of assigned attributes.

Instructions for Replacement of Attributes

To implement calculations of field values for the `RSSVAssignProcess` graph, do the following:

1. In the `RSSVAssignProcess.cs` file, add the `PX.TM` `using` directives.
2. To add the `PXDBScalar` attribute to the `DefaultAssignee` field, add the following event handler to the `RSSVAssignProcess` graph.

```
        [PXMergeAttributes(Method = MergeMethod.Append)]
        [Owner(IsDBField = false, DisplayName = "Default Assignee")]
        [PXDBScalar(typeof(SelectFrom<OwnerAttribute.Owner>.
            LeftJoin<RSSVEmployeeWorkOrderQty>.
            On<OwnerAttribute.Owner.contactID.IsEqual<
                RSSVEmployeeWorkOrderQty.userID>>.
            Where<OwnerAttribute.Owner.acctCD.IsNotNull>.
            OrderBy<RSSVEmployeeWorkOrderQty.nbrOfAssignedOrders.Asc,
                RSSVEmployeeWorkOrderQty.lastModifiedDateTime.Asc>.
            SearchFor<OwnerAttribute.Owner.contactID>))]
        protected virtual void _(
            Events.CacheAttached<RSSVWorkOrder.defaultAssignee> e)
```

```
    { }
```

> Since the `DefaultAssignee` field does not exist in the database, in the `Owner` attribute, you specify `IsDBField = false`.

3. To add the `PXUnboundDefault` attribute to the `AssignedTo` field, add the following event handler to the `RSSVAssignProcess` graph.

```
    [PXMergeAttributes(Method = MergeMethod.Append)]
    [Owner(IsDBField = false, DisplayName = "Assign To")]
    [PXUnboundDefault(typeof(RSSVWorkOrder.assignee.When<
        RSSVWorkOrder.assignee.IsNotNull>.
        Else<RSSVWorkOrder.defaultAssignee>))]
    protected virtual void _(
        Events.CacheAttached<RSSVWorkOrder.assignTo> e)
    { }
```

4. Build the project.

Related Links

- *Replacement of Attributes for DAC Fields in CacheAttached*

Step 2.2.3: Modifying the Assignment and Completion Operations

In this step, you will modify the `AssignOrders()` static method and the `complete()` action handler of the `RSSVWorkOrderEntry` graph so that they change the number of assigned work orders for each employee who is assigned a repair work order or who completed a repair work order. You will assign `1` or `-1` (depending on whether the work order is assigned or completed) to the `RSSVWorkOrder.NbrOfAssignedOrders` field; the custom accumulator attribute will add this value to the value stored in the database.

Modifying the Assignment and Completion Operations

Do the following to modify the `AssignOrders()` method and the `complete()` action handler:

1. In the `RSSVWorkOrderEntry` graph, define the data view for the calculation of the number of assigned work orders per employee, as shown in the following code.

```
    //The view for the calculation of the number of assigned work orders
    //per employee
    public SelectFrom<RSSVEmployeeWorkOrderQty>.View Quantity;
```

2. In the `AssignOrders()` method of the `RSSVWorkOrderEntry` graph, add the following line in the beginning of the `try` clause.

```
                    workOrder.Assignee = workOrder.AssignTo;
```

3. In the `AssignOrders()` method of the `RSSVWorkOrderEntry` graph, add the following code before the `workOrderEntry.Actions.PressSave()` call.

```
                    //Modify the number of assigned orders for the employee.
                    RSSVEmployeeWorkOrderQty employeeNbrOfOrders =
                        new RSSVEmployeeWorkOrderQty();
                    employeeNbrOfOrders.UserID = workOrder.Assignee;
                    employeeNbrOfOrders.NbrOfAssignedOrders = 1;
```

```
                    workOrderEntry.Quantity.Insert(employeeNbrOfOrders);
```

4. In `RSSVWorkOrderEntry` graph, modify the `complete()` action handler, as shown in the following code.

```
        public PXAction<RSSVWorkOrder> Complete;
        [PXButton]
        [PXUIField(DisplayName = "Complete")]
        protected virtual IEnumerable complete(PXAdapter adapter)
        {
            // Get the current order from the cache
            RSSVWorkOrder row = WorkOrders.Current;
            //Modify the number of assigned orders for the employee
            RSSVEmployeeWorkOrderQty employeeNbrOfOrders =
                new RSSVEmployeeWorkOrderQty();
            employeeNbrOfOrders.UserID = row.Assignee;
            employeeNbrOfOrders.NbrOfAssignedOrders = -1;
            Quantity.Insert(employeeNbrOfOrders);
            // Trigger the Save action to save changes in the database
            Actions.PressSave();
            return adapter.Get();
        }
```

5. Rebuild the project.

Related Links

- *Implementation of Processing Operations*

Step 2.2.4: Defining the External Presentation of Field Values (in FieldSelecting)

In this step, you will define the external presentation of values of the `NbrOfAssignedOrders` field of the `RSSVWorkOrder` DAC—that is, the values that are displayed in the **Number of Assigned Work Orders** column in the table on the Assign Work Orders (RS501000) form. For the configuration of the external presentation of values, you will use the `FieldSelecting` event handler. In the event handler, you will retrieve the number of assigned work orders for the employee selected in the `AssignTo` field of the `RSSVWorkOrder` DAC. If this value is `null`, the value in the **Number of Assigned Work Orders** column will be 0. You will assign the external presentation of the value to `e.ReturnValue`.

If you also need to set the internal presentation of the value, you need to assign it to `e.NewValue` in the `FieldUpdating` event handler. For unbound data fields that are only displayed in the UI, you can use only the `FieldSelecting` event that defines the UI presentation of the value. For details about the external and internal presentation of values, see *Internal and External Presentation of Values*.

Configuring the External Presentation of the NbrOfAssignedOrders Field

Modify the `RSSVAssignProcess` graph as follows:

1. In the `RSSVAssignProcess.cs` file, add the `PX.Data.BQL using` directive.
2. In the graph, define the following `FieldSelecting` event handler.

```
        protected virtual void _(Events.FieldSelecting<RSSVWorkOrder,
                        RSSVWorkOrder.nbrOfAssignedOrders> e)
        {
            if (e.Row == null) return;
```

```
            RSSVEmployeeWorkOrderQty employeeNbrOfOrders =
                SelectFrom<RSSVEmployeeWorkOrderQty>.
                Where<RSSVEmployeeWorkOrderQty.userID.IsEqual<@P.AsInt>>.
                    View.Select(this, e.Row.AssignTo);
            if (employeeNbrOfOrders != null)
            {
                e.ReturnValue = employeeNbrOfOrders.NbrOfAssignedOrders.
                    GetValueOrDefault();
            }
            else
            {
                e.ReturnValue = 0;
            }
        }
```

3. Build the project.

Related Links

- *Internal and External Presentation of Values*

Step 2.2.5: Adjusting the ASPX Page—Self-Guided Exercise

After completing this step, you will have the following columns related to the employees in the table on the Assign Work Orders (RS501000) form:

- **Assignee**: The assignee that is selected on the Repair Work Orders (RS301000) form for the work order. The value can be `null` if no value is selected on the Repair Work Orders form.
- **Default Assignee**: The default assignee, which is calculated from the database values as the employee that has the lowest number of assigned work orders. (You have implemented this logic by using the `PXDBScalar` attribute in *Step 2.2.2: Replacing Field Attributes (with PXDBScalar and PXUnboundDefault in CacheAttached)*.)
- **Assign To**: The assignee to which the repair work order will be assigned during the assignment operation. By default, for a work order, the system displays in this column the value from the **Assignee** column, if it is not `null`. If the value in the **Assignee** column is `null`, the system displays the default value from the **Default Assignee** column. (You have implemented this logic by using the `PXUnboundDefault` attribute in *Step 2.2.2: Replacing Field Attributes (with PXDBScalar and PXUnboundDefault in CacheAttached)*.) A user can override the default value in this column.

The table on the Assign Work Orders (RS501000) form already contains the **Assignee** column. In this step, you will add the **Default Assignee**, **Assign To**, and **Number of Assigned Work Orders** columns to the table.

> You will remove the **Assignee** and **Default Assignee** columns, which are not necessary for the users of the Assign Work Orders form, in *Step 2.2.6: Testing the Processing Form and the Accumulator Attribute* after you perform testing of the lesson.

Adjusting the RS501000.aspx Page

Do the following on your own:

1. Add the **Default Assignee**, **Assign To**, and **Number of Assigned Work Orders** columns to the table on the Assign Work Orders (RS501000) form, and adjust the width of the columns. (For the **Number of Assigned Work Orders** column, specify `Width="100"`.)

> ⚠ You can add the columns in the Screen Editor of the Customization Project Editor or edit the ASPX code of the form directly in Visual Studio. For details on working with the Screen Editor or editing the ASPX code in Visual Studio, see the *T200 Maintenance Forms* training course.

2. Remove `CommitChanges="True"` for the **Assignee** column.
3. For the **Assign To** column, set the following properties:
   - `CommitChanges: True`
   - `AutoRefresh: True`

   > ⚠ This property is specified for the `PXSelector` control inside `RowTemplate`. For details about how to specify the `AutoRefresh` property, see *Step 2.2.1: Restricting the Values of a Field (with PXRestrictor)* in the *T210 Customized Forms and Master-Detail Relationship* training course.
4. Publish the customization project.

Step 2.2.6: Testing the Processing Form and the Accumulator Attribute

In this step, you will test the Assign Work Orders (RS501000) form and the custom accumulator attribute; you will then remove the unnecessary UI elements from the form.

Testing the Form and the Attribute

To test the Assign Work Orders (RS501000) form, do the following:

1. On the Repair Work Orders (RS301000) form, create three repair work orders with the settings specified in the following table. Save each of them and click **Remove Hold**.

	Work Order *000009*	**Work Order *000010***	**Work Order *000011***
Customer ID	*C000000001*	*C000000002*	*C000000001*
Service	*Battery Replacement*	*Screen Repair*	*Battery Replacement*
Device	*Nokia 3310*	*Samsung Galaxy S4*	*Motorola RAZR V3*
Assignee	*Andrews, Michael*	Empty	*Beauvoir, Layla*
Description	`Test order`	`Test order`	`Test order`

   Notice that the created work orders have the *Ready for Assignment* status and have been assigned the *000009*, *000010*, and *000011* order numbers (if you have created work orders only by following the instructions in the training guides of the *T* courses).
2. On the Assign Work Orders (RS501000) form, make sure that the three repair work orders you have created are displayed and that these work orders have the specified values in the **Assignee**, **Default Assignee**, and **Assign To** columns, as shown in the screenshot below.

   For the *000009* work order, the **Assign To** setting is *Andrews, Michael*, which is the value specified in the **Assignee** column (that is, the value that you specified on the Repair Work Orders form).

   For the *000010* work order, the **Assign To** setting is *Baker, Maxwell*, which is the value specified in the **Default Assignee** column. The database currently does not have the information about the number of

repair work orders assigned to the employee. Therefore, this is the employee with the first `UserID` (which is the key field) in the database.

For the *000011* work order, the **Assign To** setting is *Beauvoir, Layla*, which is the value specified in the **Assignee** column.

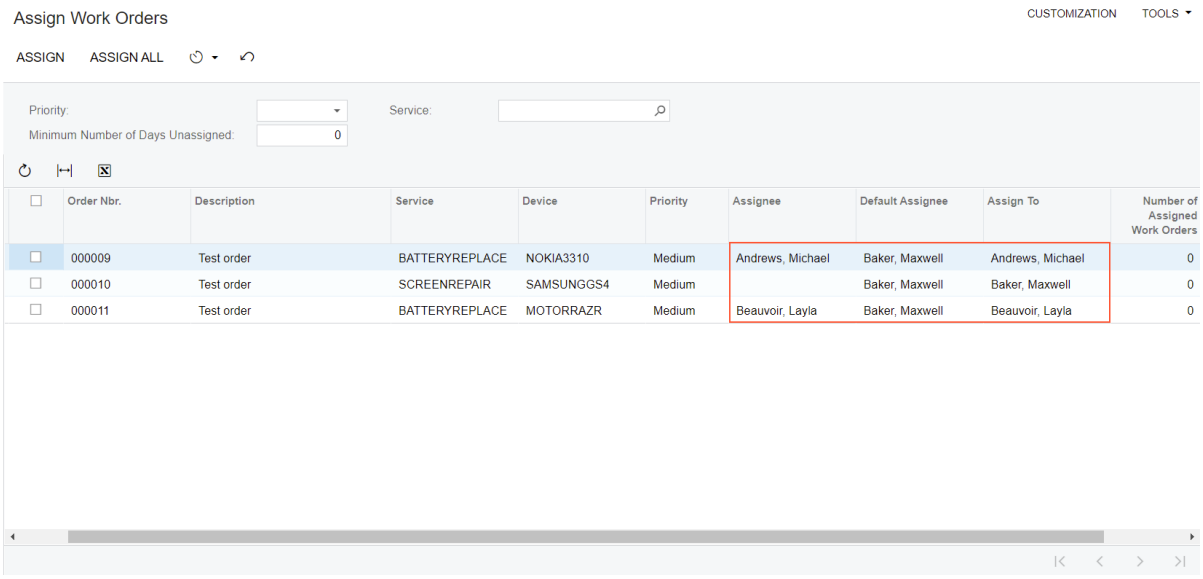


Figure: The assignees on the Assign Work Orders form

3. For the *000011* work order, change the value in the **Assign To** column to *Becher, Joseph*.
4. On the form toolbar, click **Assign All**. The work orders should be processed successfully.
5. In the **Processing** dialog box, make sure the processed repair work orders have the assignees specified as shown in the following table.

Work Order	Assignee
000009	*Andrews, Michael*
000010	*Baker, Maxwell*
000011	*Becher, Joseph*

6. Review the records in the `RSSVEmployeeWorkOrderQty` table by using Microsoft SQL Server Management Studio. The table contains three records (one for each employee to which repair work orders have been assigned during this testing). The value in the `NbrOfAssignedOrders` column is 1 for each row.
7. On the Repair Work Orders (RS301000) form, select the *000009* work order. Click **Complete** on the form toolbar.
8. In SQL Server Management Studio, review the records in the `RSSVEmployeeWorkOrderQty` table. Now for one of the rows, the value of `NbrOfAssignedOrders` is *0*.

Removing the Unnecessary Columns from the Form

You should now remove the **Assignee** and **Default Assignee** columns from the table on the Assign Work Orders (RS501000) form on your own.

You can remove the columns in the Screen Editor of the Customization Project Editor or edit the ASPX code of the form directly in Visual Studio. For details on working with the Screen Editor or editing the ASPX code in Visual Studio, see the *T200 Maintenance Forms* training course.

Lesson Summary

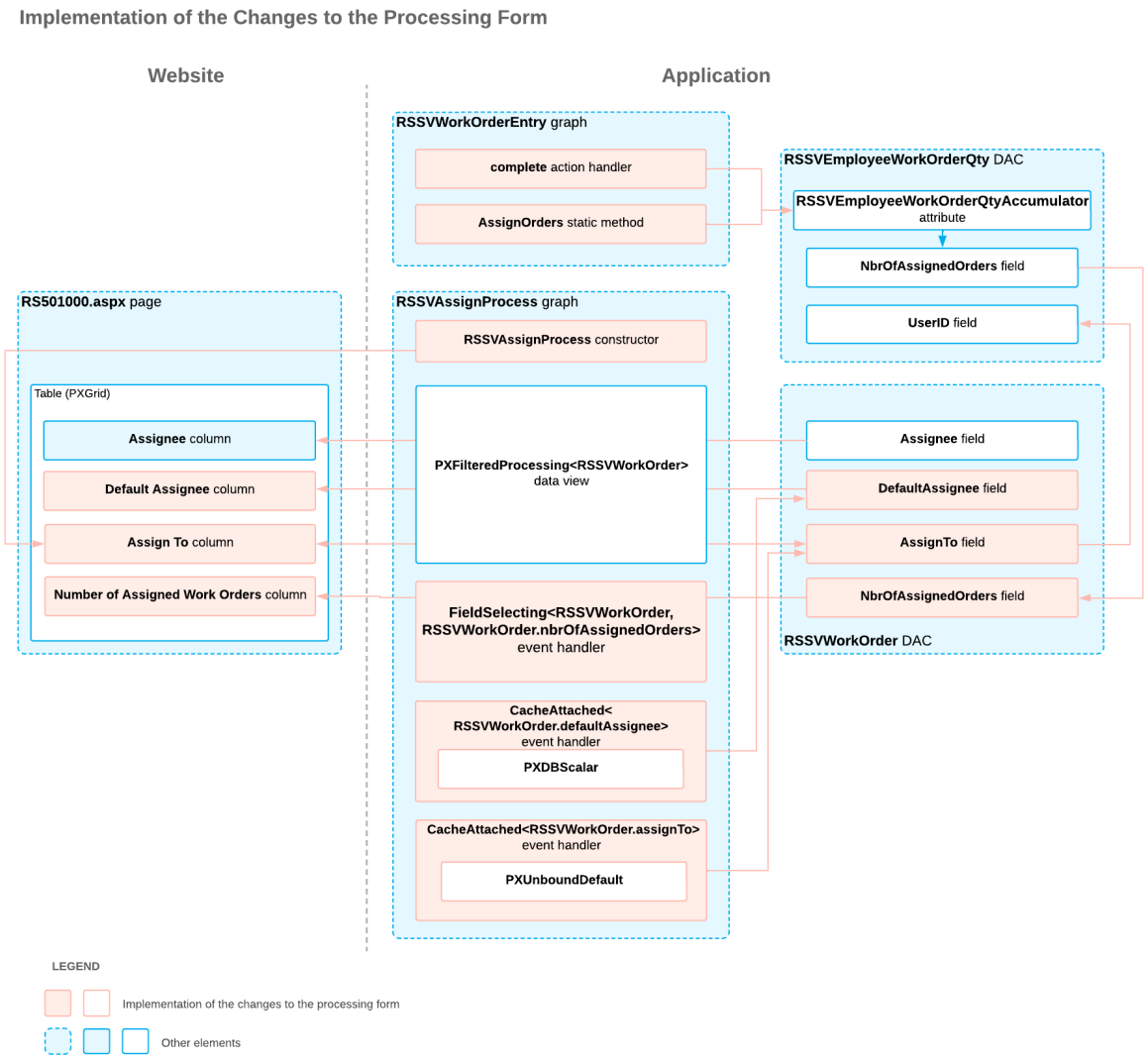
In this lesson, you have learned how to change the values of the fields that are updated by a `PXAccumulator` attribute.

You have modified the implementation of the `AssignOrders()` method and the `complete()` action handler of the `RSSVWorkOrderEntry` graph so that `1` is added to or subtracted from the number of assigned work orders. The value that is specified for the number of assigned work orders in the `AssignOrders()` method and the `complete()` action handler is added to the value stored in the database by the custom `PXAccumulator` attribute.

You have learned how to use the `PXDBScalar` and `PXUnboundDefault` attributes and how to define the external presentation of a field value.

You have also learned how to replace attributes of a DAC field using the `CacheAttached` event handler.

The following diagram shows the changes that you have performed in this lesson.



Review Questions

1. Which attribute would you use to define the field value that should be the smallest value in the column of the corresponding database table?
   a. `PXAccumulator`
   b. `PXDBCalced`
   c. `PXDBScalar`
2. Suppose that you need to define the external presentation of a field value in run time. How would you specify the external presentation of the value?
   a. In `e.NewValue` of the `FieldUpdating` event handler
   b. In `e.ReturnValue` of the `FieldSelecting` event handler
   c. In `e.NewValue` of the `FieldSelecting` event handler
   d. In `e.ReturnValue` of the `FieldUpdating` event handler

Answer Key

1. **c**
2. **b**

Part 3: Redirection to a Report at the End of Processing

For better usability of the Assign Work Orders (RS501000) form, the managers of the Smart Fix company have requested that the form be modified so that at the end of the processing, the system displays a report that shows which repair work orders have been assigned to which employees during the processing.

In this part of the course, you will modify the processing operation of the Assign Work Orders form so that it displays this report at the end of the operation. You will use the `RS601000.rpx` report file, which is provided with this training course, as the report to be displayed.

> ⚠ Creation of reports with Acumatica Report Designer is outside of the scope of this training course. To learn more about creation of reports, see the *S130 Data Retrieval and Analysis* training course.

As part of completing the lesson of this part, you will test the updated functionality of the form.

Lesson 3.1: Adding Redirection to a Report at the End of Processing

In this lesson, you will modify the Assign Work Orders (RS501000) form so that it displays a report at the end of processing. The report will list the repair work orders that have been assigned during the assignment operation and the assignees to which they are assigned.

An example of the report about an assignment operation is shown in the following screenshot.

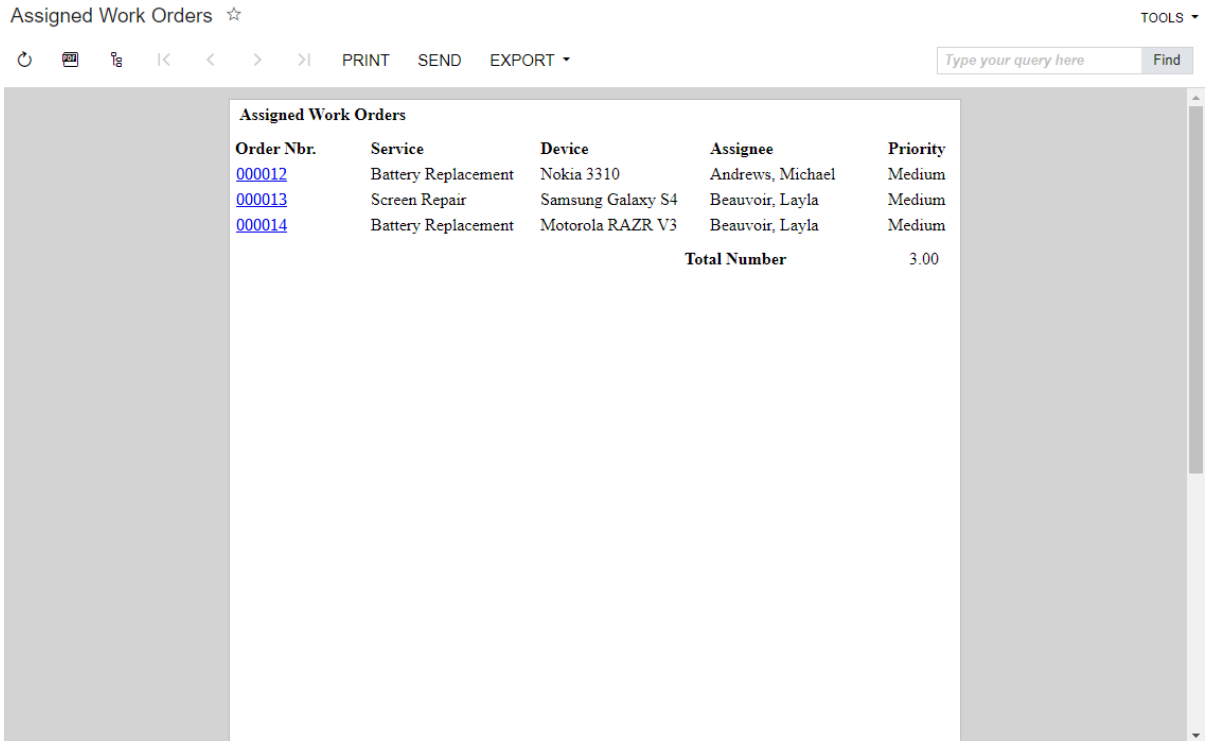


Figure: The report

You will also add the report file to the *PhoneRepairShop* customization project.

Lesson Objectives

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

- Redirect to a report at the end of the processing operation
- Include a report in a customization project

Step 3.1.1: Including a Report in the Customization Project

In this step, you will add the `RS601000.rpx` report file, which is provided with this training course, to the customization project. You must include the report file in the customization project so that the report is available in each Acumatica ERP instance to which you publish the *PhoneRepairShop* customization project.

The report is not supposed to be used directly from the UI of Acumatica ERP; therefore, you will not include it in any workspace.

Including RS601000.rpx in the Customization Project

To include the report file in the customization project, do the following:

1. Copy the `RS601000.rpx` file to the `ReportsCustomized` folder of your Acumatica ERP instance for the training course. The system uses this folder to search for custom and customized Acumatica ERP reports.
2. In the Customization Project Editor, open the *PhoneRepairShop* customization project.
3. On the Custom Files page, add the `ReportsCustomized\RS601000.rpx` file, and save your changes.

   For details on adding files to the customization project, see *To Add a Custom File to a Project* in the documentation.
4. Publish the customization project.
5. On the *Site Map* (SM200520) form of Acumatica ERP, add a new row with the following settings, and save your changes:
   - **Screen ID**: `RS.60.10.00`
   - **Title**: `Assigned Work Orders`
   - **URL**: `~/frames/reportlauncher.aspx?id=RS601000.rpx`
   - **Graph Type**: Empty
   - **Workspaces**: Empty
   - **Category**: Empty
6. In the Customization Project Editor (with it opened for the *PhoneRepairShop* customization project), on the Site Map page, add the site map item for the Assigned Work Orders report.

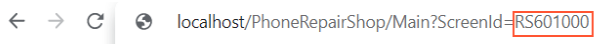
   

   For details about addition of a site map item to the customization project, see *To Add a Site Map Node to a Project* in the documentation.
7. Publish the customization project.

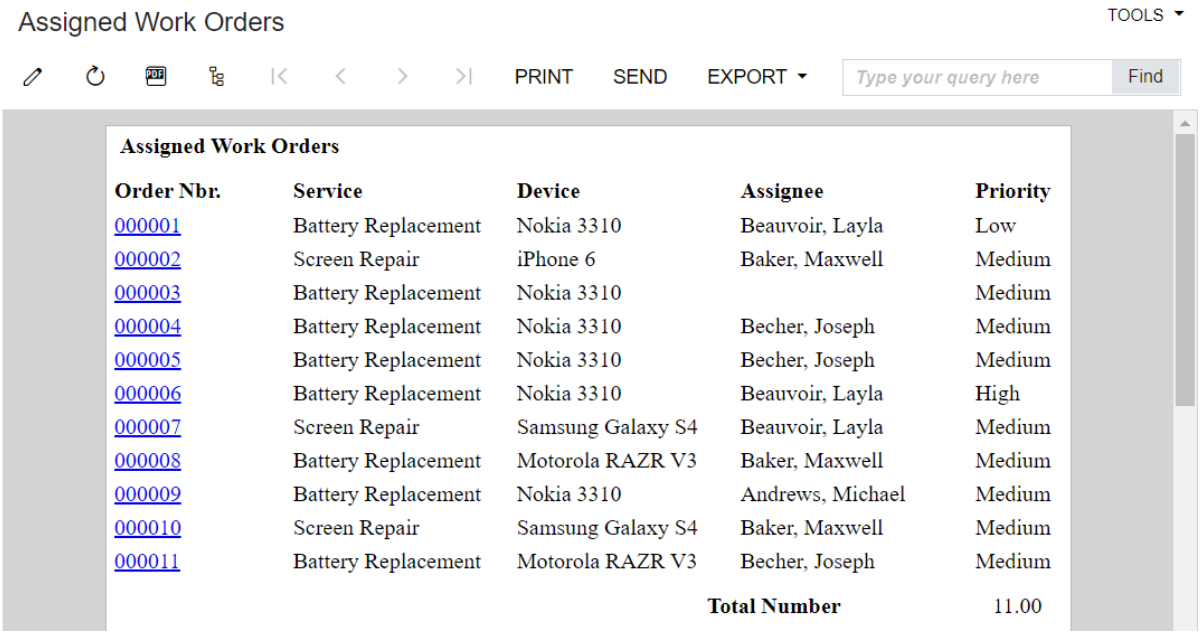
Testing the Report

In Acumatica ERP, make sure the report is displayed correctly as follows:

1. Open the Assigned Work Orders (RS601000) report form.

> Because the report has no workspace specified, you cannot find it in the UI by typing its name in the **Search** box or browsing the main menu. The only way to open the report is to type the ID of the report in the address line of the browser, as shown in the following screenshot.
>
> 
>
>
> *Figure: Report ID*

2. On the report form toolbar, click **Run Report**. The report is displayed as shown in the following screenshot. Because no filtering is specified in the report settings, the report displays all the repair work orders that exist in the application database. When the system redirects to this report from the Assign Work Orders (RS501000) form, filtering will be specified.



Assigned Work Orders

Order Nbr.	Service	Device	Assignee	Priority
000001	Battery Replacement	Nokia 3310	Beauvoir, Layla	Low
000002	Screen Repair	iPhone 6	Baker, Maxwell	Medium
000003	Battery Replacement	Nokia 3310		Medium
000004	Battery Replacement	Nokia 3310	Becher, Joseph	Medium
000005	Battery Replacement	Nokia 3310	Becher, Joseph	Medium
000006	Battery Replacement	Nokia 3310	Beauvoir, Layla	High
000007	Screen Repair	Samsung Galaxy S4	Beauvoir, Layla	Medium
000008	Battery Replacement	Motorola RAZR V3	Baker, Maxwell	Medium
000009	Battery Replacement	Nokia 3310	Andrews, Michael	Medium
000010	Screen Repair	Samsung Galaxy S4	Baker, Maxwell	Medium
000011	Battery Replacement	Motorola RAZR V3	Becher, Joseph	Medium
			Total Number	11.00

Figure: Assigned Work Orders report

Related Links

- *To Add a Custom File to a Project*
- *To Add a Site Map Node to a Project*

Step 3.1.2: Adding Redirection to a Report

In this step, you will implement redirection to the Assigned Work Orders (RS601000) report at the end of the `AssignOrders()` method. The report will display the repair work orders that have been assigned during the processing operation the user invoked on the form.

To redirect to the report, you will throw the `PXReportRequiredException` exception. Once an exception is thrown, it interrupts the current context and propagates up the call stack until it is handled by Acumatica Framework, which performs the redirection. You don't need to implement the handling of the exceptions that are used for redirection.

The Assigned Work Orders report has no filtering parameters. You will pass the data to be displayed in the report (that is, the repair work orders that have been assigned) in the parameters of the `PXReportRequiredException` constructor.

For details about the implementation of redirection to webpages, see *Redirection to Webpages* in the documentation.

Implementing Redirection to a Report

To implement the redirection, do the following:

1. In the `Messages.cs` file, add the following constant, which specifies the name of the webpage that will display the report.

```
public const string ReportRS601000Title = "Assigned Work Orders";
```

2. In the `RSSVWorkOrderEntry.cs` file, modify the `AssignOrders()` method, as follows:
   a. In the beginning of the method, add the following lines.

```
// The result set to run the report on.
PXReportResultset assignedOrders =
    new PXReportResultset(typeof(RSSVWorkOrder));
```

   b. In the end of the `try` block, add the following code.

```
// Add to the result set the order
// that has been successfully assigned.
if (workOrder.Status == WorkOrderStatusConstants.Assigned)
{
    assignedOrders.Add(workOrder);
}
```

   c. In the end of the method, add the following code.

```
if (assignedOrders.GetRowCount() > 0 && isMassProcess)
{
    throw new PXReportRequiredException(assignedOrders, "RS601000",
                                        Messages.ReportRS601000Title);
}
```

3. Build the project.
4. Publish the customization project to include the latest version of the extension library in the customization project.

Related Links

- *Redirection to Webpages*

Step 3.1.3: Testing the Redirection to the Report

In this step, you will test the redirection to the Assigned Work Orders (RS601000) report, which should occur at the end of the assignment operation on the Assign Work Orders (RS501000) form.

Testing the Redirection to the Report

To test the redirection to the report, do the following:

1. On the Repair Work Orders (RS301000) form, create three repair work orders with the settings specified in the following table. Save each of them and click **Remove Hold**.

	Work Order *000012*	**Work Order *000013***	**Work Order *000014***
Customer ID	*C000000001*	*C000000002*	*C000000001*
Service	*Battery Replacement*	*Screen Repair*	*Battery Replacement*
Device	*Nokia 3310*	*Samsung Galaxy S4*	*Motorola RAZR V3*
Assignee	*Andrews, Michael*	Empty	*Beauvoir, Layla*
Description	`Test order`	`Test order`	`Test order`

   The created work orders have the *Ready for Assignment* status and have been assigned the *000012*, *000013*, and *000014* order numbers (if you have created work orders only by following the instructions in the training guides of the *T* courses).
2. On the Assign Work Orders (RS501000) form, make sure that three repair work orders are displayed.
3. On the form toolbar, click **Assign All**. At the end of the processing, the Assigned Work Orders (RS601000) report is displayed for the three assigned work orders, as shown in the following screenshot.

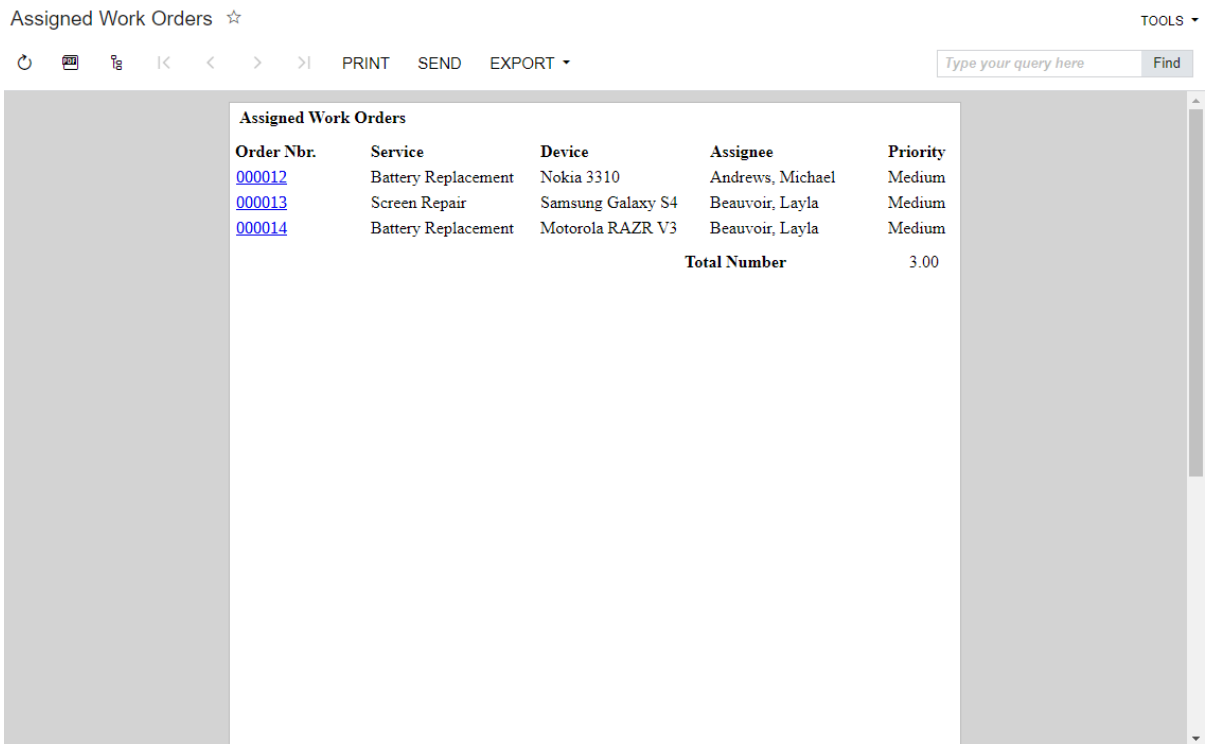
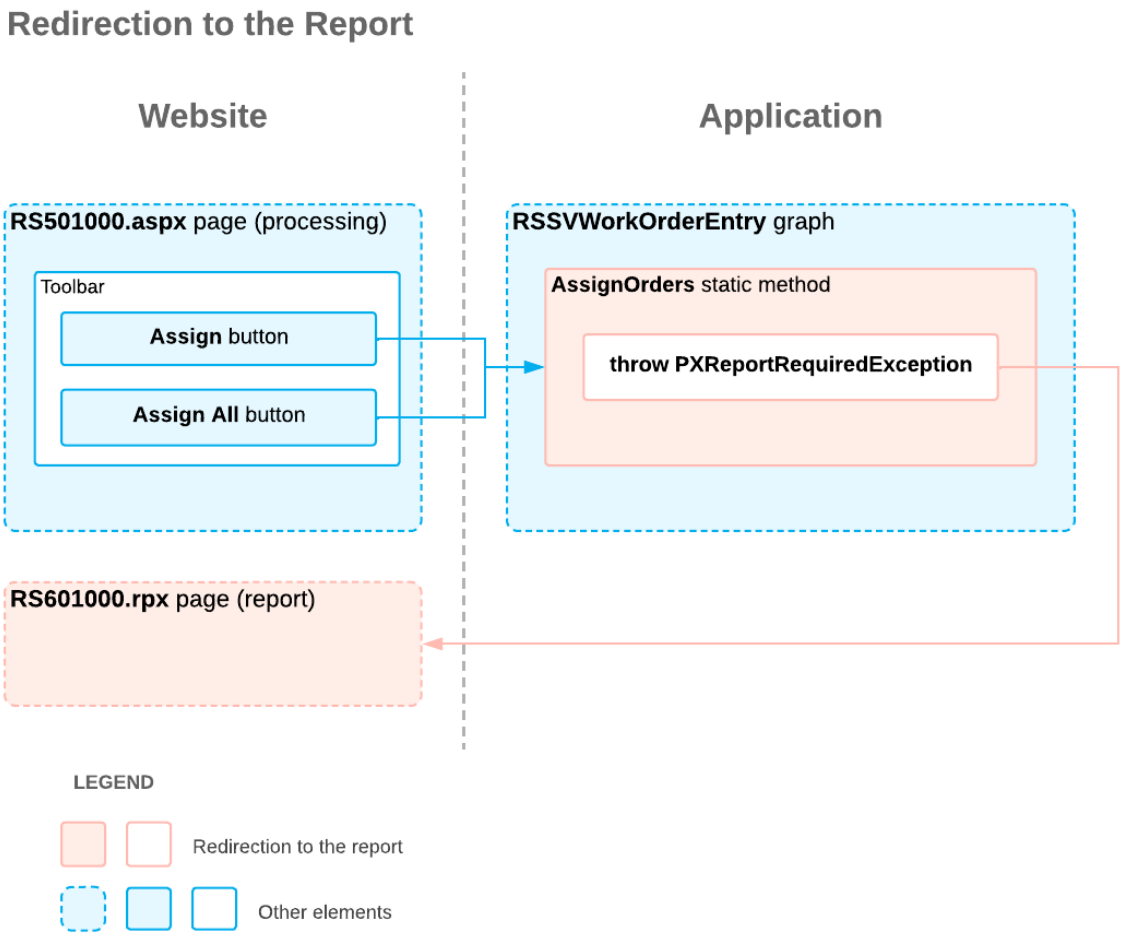


Figure: Assigned Work Orders report

Lesson Summary

In this lesson, you have learned how to implement the redirection to a report at the end of the processing delegate. You have used the `PXReportRequiredException` exception to perform the redirection. You have passed the result set with the data of the repair work orders that have been assigned to the `PXReportRequiredException` constructor.

The following diagram shows the summary of the implementation.



Review Questions

1. Which approach can you use to redirect to a report at the end of the processing?
   a. Throw the `PXReportRequiredException` exception in the processing method
   b. Throw the `PXReportRequiredException` exception and handle it in the processing method
   c. Implement an action handler and call it in the processing method
2. Which information do you need to include in the customization project so that the customized application performs redirection to a custom report?

a. Only the report file

b. The report file and its position in the UI

c. The report file, its position in the UI, and the implementation of the redirection to this report (in an extension library or a *Code* item of the customization project)

Answer Key

1. **a**
2. **c**

Appendix: Use of Event Handlers

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

Event	Scenario	Examples in the Guide
`CacheAttached`	Replacing the attributes of a DAC field	*Step 2.2.2: Replacing Field Attributes (with PXDBScalar and PXUnboundDefault in CacheAttached)*
`FieldSelecting`	Defining the external presentation of a field value (that is, the value that is displayed in the UI)	*Step 2.2.4: Defining the External Presentation of Field Values (in FieldSelecting)*
`RowSelected`	Specifying the workflow action to be used for the processing	• *Step 1.1.3: Configuring the Processing Graph and Data View (with ProcessingView and RowSelected)* • *Step 1.2.3: Defining the Data Views (with PXFilter and ProcessingView.FilteredBy)*

Appendix: Reference Implementation

You can find the reference implementation of the customization described in this course in the `Customization\T240` 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 *T100* data inserted by specifying the following settings:
      - **Login Tenant Name**: `MyTenant`
      - **New**: Selected
      - **Insert Data**: *T100*
      - **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)`, `C:\Program Files`, and `C:\Users` 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.

You load the customization project with the results of the *T230 Actions* 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\T230\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\T230\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)`, `C:\Program Files`, and `C:\Users` folders, we recommend that you use the `App_Data\Projects` folder for the project of the extension library.
      >
      > If the website folder is in the `C:\Program Files (x86)`, `C:\Program Files`, or `C:\Users` folder, we recommend that you store the project outside of these folders to avoid an issue with permission to work in these folders. In this case, you need to update the links to the website and library references in the project.

   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. On 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 previous training courses of the *T* series. This project also contains the customization plug-in, which fills in the tables created in these 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*. (The creation of customization plug-ins is outside of the scope of this course.)