Acumatica
The Cloud ERP

T240 Processing Forms

Dhiren Chhapgar

Principal Software Engineer

Timing and Agenda

Nov 18, 2020 – 10 AM-11:30 PM PST

Day 1

- **Creating a Simple Processing Form (**Company Story + Lesson 1.1**)**

Nov 19, 2020 – 10 AM-11:30 PM PST

Day 2

- **Adding Filtering Parameters to the Form and Implementing a Custom PXAccumulator Attribute (**Lesson 1.2 - Lesson 2.1**)**

Nov 20, 2020 – 10 AM-11:30 PM PST

Day 3

- **Updating Data with a Custom Accumulator Attribute and Redirecting to a Report at the End of Processing (**Lesson 2.2 - Lesson 3.1**)**

Initial Configuration before we start with T240

Previous training courses

New Forms

- ✓ RS201000 : Repair Services - maintenance form to manage the lists of repair services
- ✓ RS202000 : Serviced Devices - maintenance form to manage the lists of devices that can be serviced
- ✓ RS203000 : Services and Prices - maintenance form to define and maintain the price for each provided repair service
- ✓ RS301000 : Repair Work Orders - data entry form to create and manage work orders for repairs
- ✓ RS101000 : Repair Work Order Preferences - setup form to specify the company's preferences for the repair work orders

Existing Forms

- ✓ IN202500 : Stock Items - Modified to mark a particular stock items as repair item
- ✓ AR302000 : Payments and Applications - modify the status of a repair work order once the payment has been released

T240: Processing Forms

Processing Pages

What are processing pages ?

- ✓ To process multiple records at once
- ✓ Processing can be manual or scheduled without user involvement
- ✓ Option to specify filter
- ✓ Name of ASPX pages for processing start with the two-letter module abbreviation followed by 50.

Examples –

- ❑ AR501000 : Release AR Documents
- ❑ SO501000 : Process Orders

T240: Assign Work Orders

T240 – Processing Form

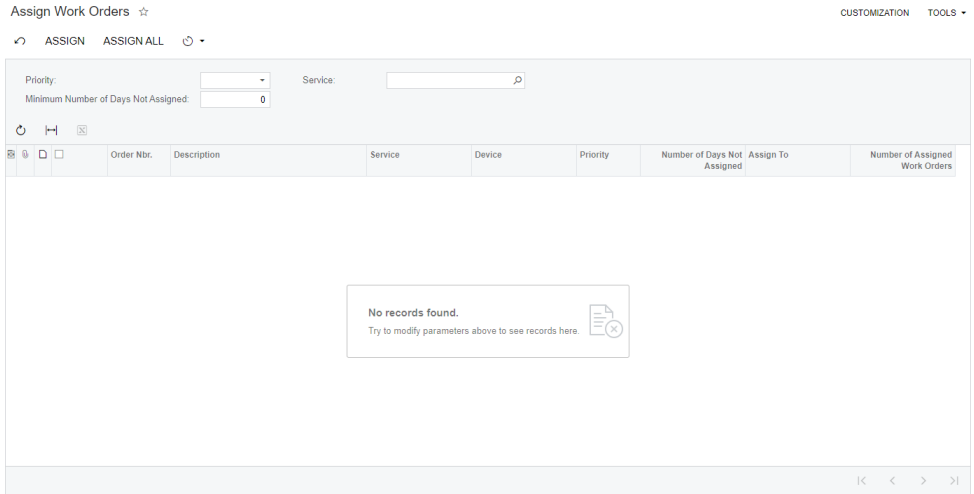
New Processing Page - Assign Work Orders (RS501000)

- ✓ To assign multiple repair work orders at the same time.
- ✓ Simple processing form without any filtering parameters for user selection.
- ✓ Add a filter to the form so that only the records that satisfy the filtering parameters are displayed in the table.
- ✓ Implement the selection of the default assignee
- ✓ Use of PXAccumulator attribute to update the number of assigned orders for each employee. I
- ✓ Implement redirection to a report at the end of the processing.

Actions on the Repair Work Orders Form

The form will contain the following actions:

- **Assign & Assign All** – mass-processing action.
- **Filter** – dynamic filtering
- **Table** – list of the orders that are ready for assignement.



1.1: Creating a Simple Processing Form

Objectives:

- Learn how to create a simple processing form

Processing Graphs

In graphs for processing pages you can use data views of the `PXProcessing` type.

You can set the processing routine for such a view with

```
pxProcessingView.SetProcessDelegate(StaticProcessingRoutine);
```

When processing, you can use following static methods of `PXProcessing`:

- `SetInfo(index, messageOrException)`
- `SetWarning(index, messageOrException)`
- `SetError(index, messageOrException)`

PXProcessing

There are variations:

- `PXProcessing` – provides data records for processing together with the related convenience routines. Allows `Where` and `OrderBy` BQL clauses
- `PXProcessingJoin` – all the same but permits `Join` clause
- `PXFilteredProcessing` – the same as `PXProcessing`, but takes a filter DAC argument to enable filtering records properly
- `PXFilteredProcessingJoin` – combines the benefits of the above two

Selecting Records

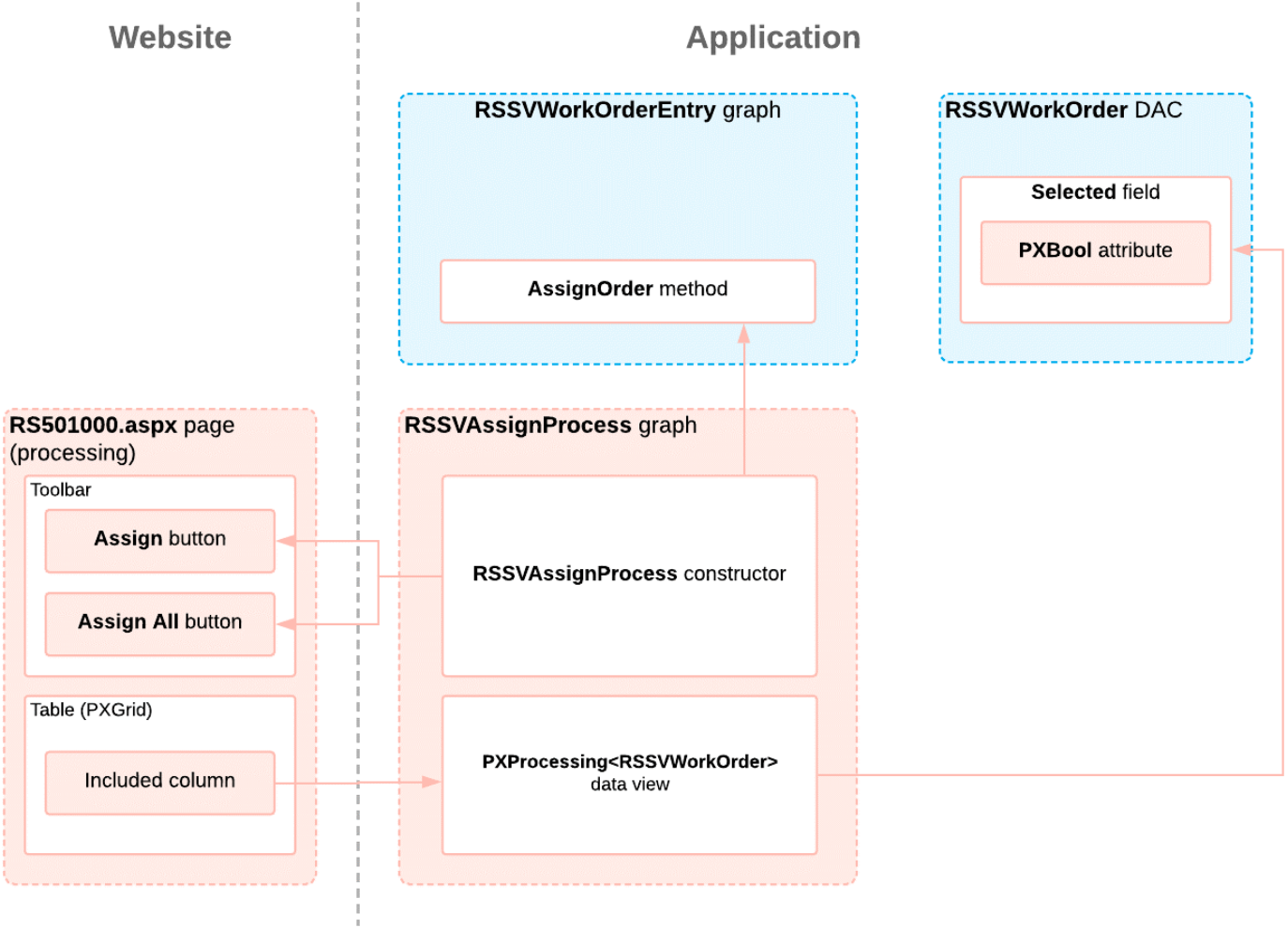
To pick records for processing the system uses the `Selected` field. You can declare it on your DAC like this:

```
public abstract class selected : IBqlField { }

[PXBool]
[PXUIField(DisplayName = "Selected")]
public virtual bool? Selected { get; set; }
```

It is possible to use other field for this purpose – tell the system of it with `PXProcessing.SetSelected<DAC.field>()` method.

Summary


Website
Application
RSSVWorkOrderEntry graph
AssignOrder method
RSSVWorkOrder DAC
Selected field
PXBool attribute
RS501000.aspx page (processing)
Toolbar
Assign button
Assign All button
Table (PXGrid)
Included column
RSSVAssignProcess graph
RSSVAssignProcess constructor
PXProcessing<RSSVWorkOrder> data view

1.2: Adding Filtering Parameters to the Processing Form

Objectives:

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

PXFilter

Special type of data view that is used to provide filtering parameters

```
public PXFilter<Product> Products;
```

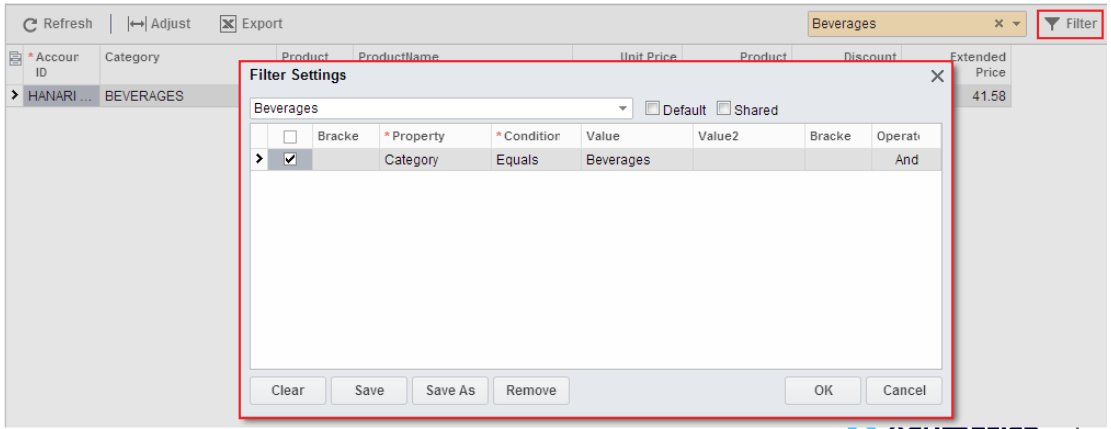
- Always creates a single data record
- Never retrieves or saves this data record to the database.
- The PXFilter data view is used to specify values used by the application logic or other data views and never supposed to be stored anywhere besides the current user session.
- Can be used with virtual DACs

PXFilterable

Added to a details data view to enable custom reusable filters

```
[PXFilterable]
public PXSelectReadonly<Product> Products;
```

Adds a Filter Settings dialog to the grid thus allowing users to create, save and reuse their own filters



IsDirty

To avoid the confirmation dialog when leaving processing pages:

```
//in the graph
public override bool IsDirty
{
	get
	{
		return false;
	}
}
```

Tells the system that there are no unsaved changes in the graph

2.1, 2.2: PXAccumulator Attribute

Objectives:

- Implement a custom attribute derived from the PXAccumulator attribute
- Implement a processing operation by using a static method
- Specify the values of the fields updated by a PXAccumulator attribute
- Use the PXDBScalar attribute
- Define the external presentation of field values

PXDBScalar

Allows to define a sub-query to select a value from a field of any DAC

```
//In the ProductReorder class
[PXDecimal(2)]
[PXDBScalar(typeof(Search<ProductQty.availQty,
       Where<ProductQty.productID, Equal<ProductReorder.productID>>>))]
public virtual decimal? AvailQty { get; set; }
```

```
SELECT ..., (SELECT TOP (1) ( productqty.availqty )
FROM productqty ProductQty
WHERE ( productqty.productid = ProductReorder.productid )
ORDER BY productqty.availqty), ...
FROM ...
```

PXDBCalced

Defines an expression to calculate a value from fields of the same DAC

```
[PXDecimal(2)]
[PXDBCalced(typeof(Minus<
                  Sub<IsNull<ProductReorder.availQty, decimal_0>,
                      ProductReorder.minAvailQty>>),
            typeof(decimal))]
public virtual decimal? Discrepancy { get; set; }
```

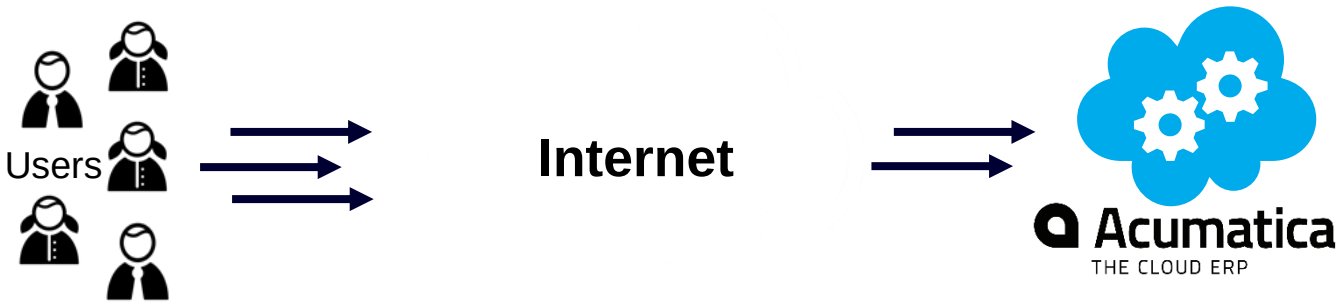
`PXDBCalced` can be used only on unbound fields!

Accumulator Attribute

Concurrency Control

You can’t lock record from reading while someone edits it:

- You don’t know who and when will save it
- You don’t know if user still online



Concurrency Control

2 Scenarios:

Editing of Single record

- Use of “Optimistic Concurrency Control” technics

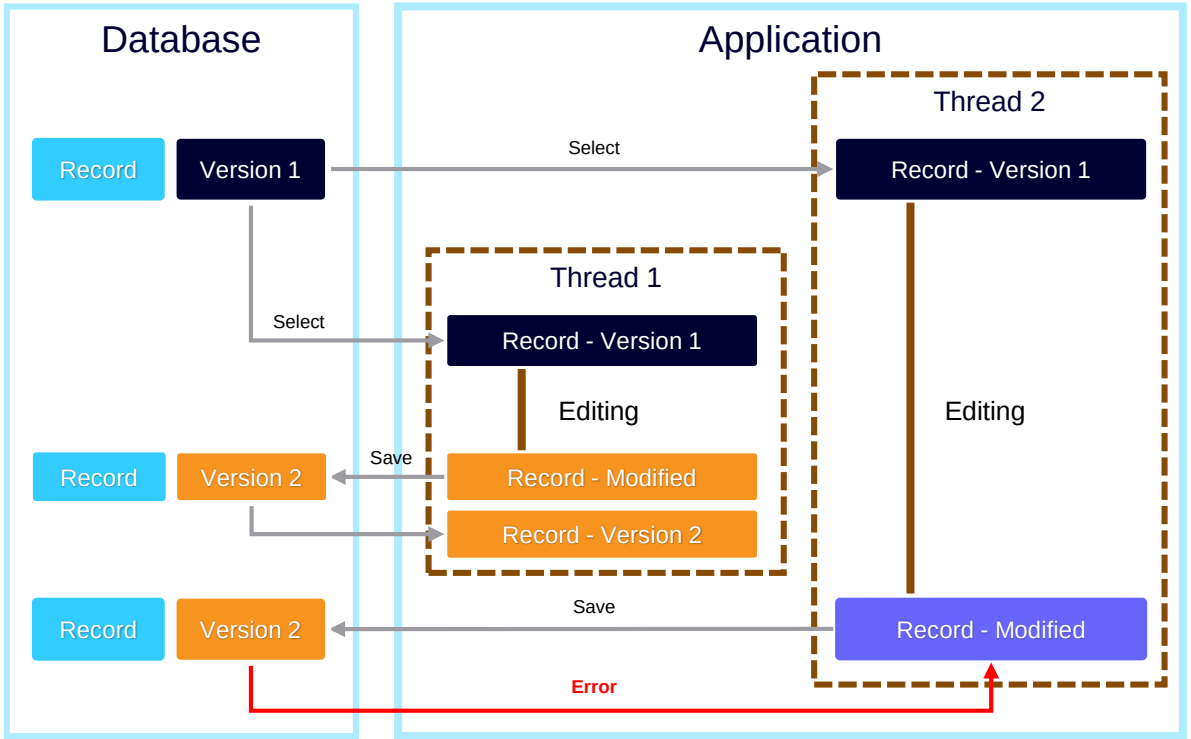
Concurrent Update of Shared record

- Use of calculation rules instead of value replacement

Optimistic Concurrency Control

Transactions does not lock resources assuming that they can frequently complete without interfering with each other.

Before committing, each transaction verifies record version.



Day 3

Agenda:

- Updating Data with a Custom Accumulator Attribute (Lesson 2.2)
- Redirecting to a Report at the End of Processing (Lesson 3.1)

PXAccumulator

Accumulators are used to update fields, which are changed frequently and concurrently by many users. They affect SQL executed to update data in DB.

You might want to use `PXAccumulator`-based attribute to:

- Update a field without checking for the version of the data record stored in the DB (ignoring the optimistic blocking mechanism)
- Define specific update policy – possibly with certain restrictions

PXAccumulator – Calculation Rules

CompanyID	Product	Quantity	Timestamp
1	TV	10	3/26/2015 5:21 PM

User A – 4 Sell TV = 6 and User B – Buy 12 TV = 22

Where the correct result? Do we need to recalculate values each time?

Instead of replacing values we add the calculation rules.

Lets evaluate it by difference: 10 - 4 + 12 = 18

PXAccumulator

To create your own accumulator you inherit from `PXAccumulatorAttribute` and typically have to implement:

- constructor
- `PrepareInsert()` method
  - Call the `base.PrepareInsert()` to initialize the collection of columns. (it will returns `true` if initialization succeeds, your implementation should also return `true` if everything goes well)
  - Set restrictions and update policies for specific columns
- `PersistInserted()` method
  - Overriding the PersistInserted() method is needed to tweak the persist operation—for example, to catch and process errors.
  - Can be replaced with columns.AppendException which adds Restriction as well as configures exception when restriction is violated

PXAccumulator – Example

```
1 reference | 0 changes | 0 authors, 0 changes
public class ARBalAccumAttribute : PXAccumulatorAttribute
{
    1 reference | 0 changes | 0 authors, 0 changes
    public ARBalAccumAttribute()
    {
        base._SingleRecord = true;
    }
    63 references | 0 changes | 0 authors, 0 changes
    protected override bool PrepareInsert(PXCache sender, object row, PXAccumulatorCollection columns)
    {
        if (!base.PrepareInsert(sender, row, columns))
        {
            return false;
        }

        ARBalances bal = (ARBalances)row;

        columns.Update<ARBalances.lastInvoiceDate>(bal.LastInvoiceDate, PXDataFieldAssign.AssignBehavior.Maximize);

        return true;
    }
}
```

```
[Serializable]
[ARBalAccum]
[PXCacheName(Messages.ARBalances)]
99+ references | 0 changes | 0 authors, 0 changes
public partial class ARBalances : PX.Data.IBqlTable
{ ... }
```

In this example, the class derived from PXAccumulatorAttribute overrides the PrepareInsert() method and specifies the assignment behavior for several fields.

PXAccumulator – Example

```
[PXAccumulator(
    new Type[]
    {
            typeof(APHistory.finYtdBalance),
            typeof(APHistory.tranYtdBalance),
            typeof(APHistory.finYtdBalance)
    },
    new Type[]
    {
            typeof(APHistory.finBegBalance),
            typeof(APHistory.tranBegBalance),
            typeof(APHistory.finYtdBalance),
    })]
[Serializable]
[PXHidden]
20 references | 0 changes | 0 authors, 0 changes
public partial class APHist : APHistory, IBaseAPHist
{ ... }
```

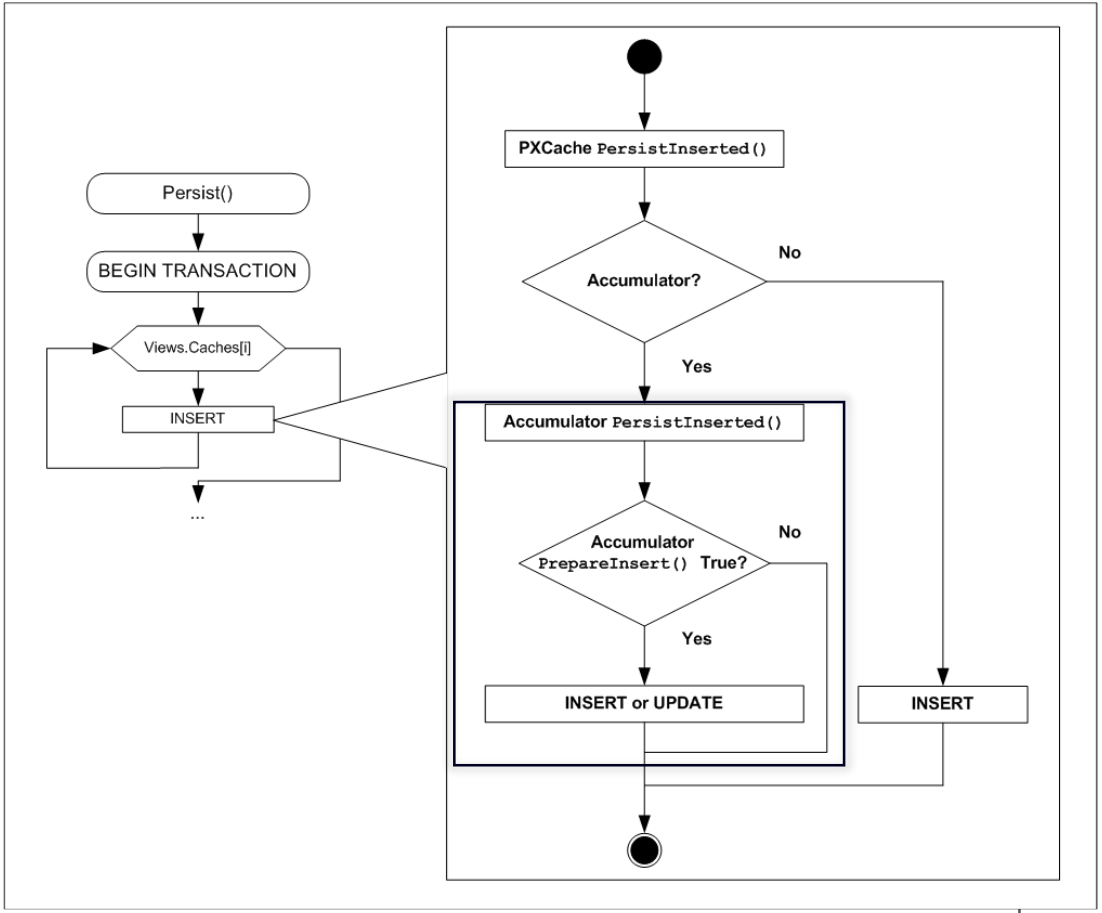
Code shows how the attribute can be used directly.

When a data record is saved, value of every field from the first array will be added to the previously saved value of the corresponding field from the second array.

That is, **FinYtdBalance** values will be accumulated in the **FinBegBalance** value, TranYtdBalance values in the TranBegBalance value, and so on

PXAccumulator

How can this possibly work?



PXAccumulator

You set update policies with `columns.Update<Field>(value, policy)`

Policies are available through `PXDataFieldAssign.AssignBehavior`:

1. **`Replace`**: `The new value is inserted into the data field, and the previous value is overwritten.`
2. **`Summarize:`** `The new value is added to the value stored in the database.`
3. **`Maximize:`** `The maximum of the new value and the value from the database is saved in the database.`
4. **`Minimize:`** `The minimum of the new value and the value from the database is saved in the database.`
5. **`Initialize`**: `The new value is saved in the database as the value if the field does not have a value in the database. If the data field is not null, the new value is discarded.`

Note: `Minimize` and `Maximize` do not require setting new `value`. Other – do

PXAccumulator Restriction

1) `columns.Restrict<Field>(comparison,value)`

- Violated restriction raises `PXLockViolationException` – you *must* handle it in `PersistInserted()`
- Restriction violation is *not the only* reason for `PXLockViolationException`

2) `columns.AppendException` in `PrepareInsert()`:

- Sets restriction and handles violation – `PersistInserted()` not needed
- Usage `columns.AppendException(`

```
"Update will lead to negative quantity in stock!",
new PXAccumulatorRestriction<ProductQty.availQty>(PXComp.GE, 0m));
```

PXAccumulator - Usage

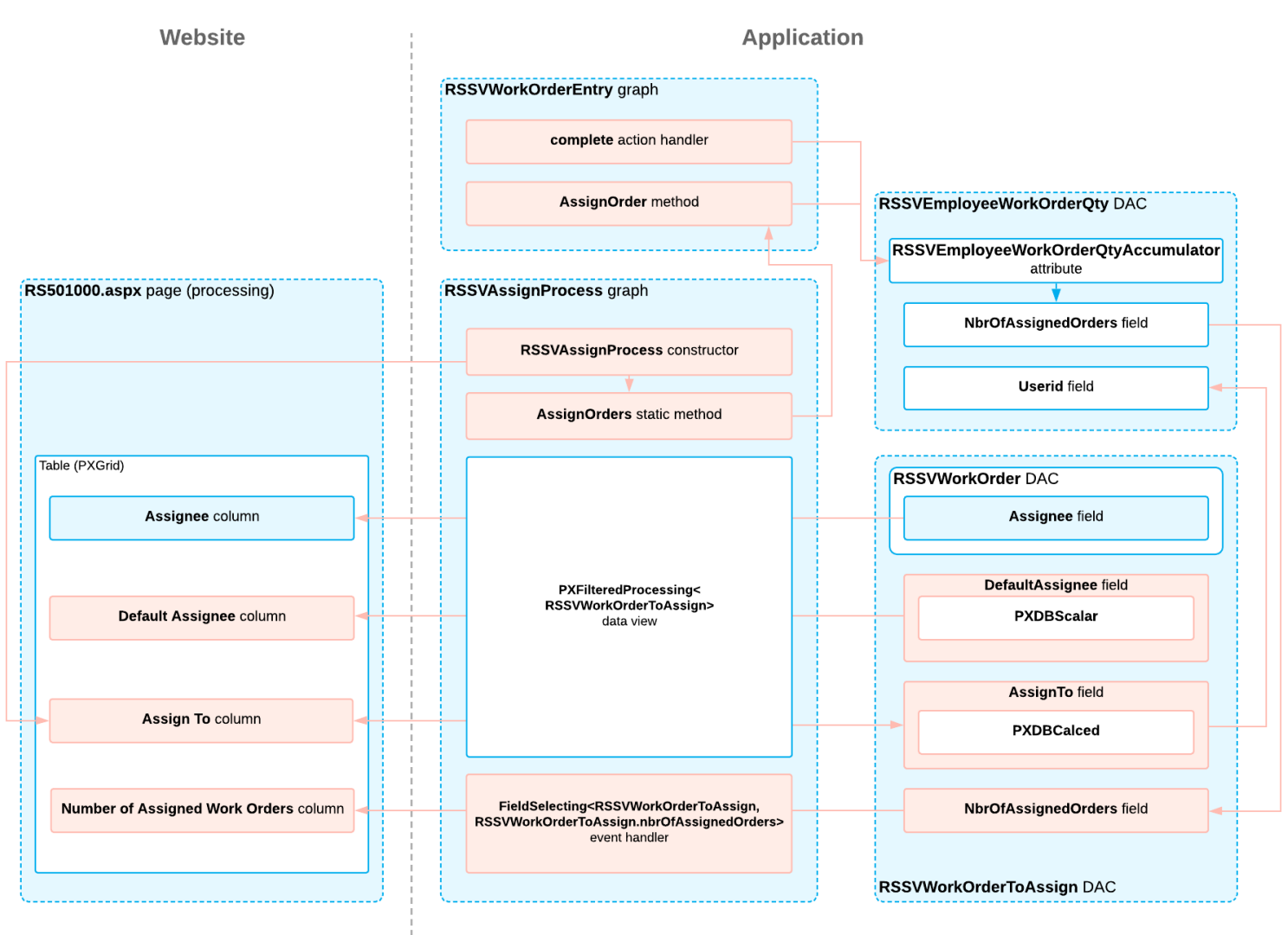
Accumulator will trigger update logic only for records with `Inserted` status

You should use `cache.Insert(…)` if you want records to be handled by it

Framework will know whether to update an existing or to insert a new record

In `PersistInserted()` set `columns.InsertOnly` or `columns.UpdateOnly` to permit only insert or update database operations

Summary


Website
Application
RSSVWorkOrderEntry graph
complete action handler
AssignOrder method
RSSVEmployeeWorkOrderQty DAC
RSSVEmployeeWorkOrderQtyAccumulator attribute
NbrOfAssignedOrders field
Userid field
RS501000.aspx page (processing)
RSSVAssignProcess graph
RSSVAssignProcess constructor
AssignOrders static method
Table (PXGrid)
Assignee column
Default Assignee column
Assign To column
Number of Assigned Work Orders column
PXFilteredProcessing< RSSVWorkOrderToAssign> data view
FieldSelecting<RSSVWorkOrderToAssign, RSSVWorkOrderToAssign.nbrOfAssignedOrders> event handler
RSSVWorkOrder DAC
Assignee field
DefaultAssignee field
PXDBScalar
AssignTo field
PXDBCalced
NbrOfAssignedOrders field
RSSVWorkOrderToAssign DAC

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

Objectives:

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

Redirecting

We use exceptions to redirect:

- `PXRedirectRequiredException`
- `PXPopupRedirectException`
- `PXRedirectWithReportException`
- `PXRedirectToUrlException`

```
DocumentEntry graph = PXGraph.CreateInstance<DocumentEntry>();
graph.Document.Current = someDoc;
throw new PXRedirectRequiredException(graph, "Document");
```

What next?

Join Development Community, participate, learn from others

Acumatica Development Network (ADN)

- http://adn.acumatica.com/

Acumatica Community

- https://community.acumatica.com/

Stack Overflow Community:

- http://stackoverflow.com/questions/tagged/acumatica

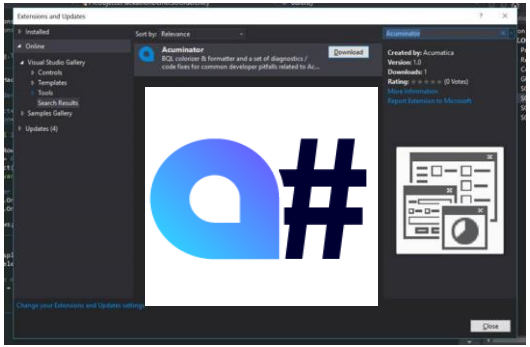
Git Hub opensource Projects

- https://github.com/Acumatica/

Visual Studio Extensions – Acuminator

Blogs

- http://asiablog.acumatica.com/
- http://www.timrodman.com/tag/acumatica/



Acumatica Developer Network

GitHub

Blogger™

Acumatica
The Cloud ERP

Thank You

Dhiren Chhapgar

No Reliance
This document is subject to change without notice. Acumatica cannot guarantee completion of any future products or program features/enhancements described in this document, and no reliance should be placed on their availability.

Confidentiality: This document, including any files contained herein, is confidential information of Acumatica and should not be disclosed to third parties.