Acumatica
The Cloud ERP

T220 Data Entry and Setup Forms

Sergey Marenich

Commerce Team Lead & Architect

Marenich Sergey

Blog: http://asiablog.acumatica.com

Experience: 13 years of experience at Acumatica as a developer, solution architect, commerce team lead.

Specialization: As a developer, he specializes in C# with deep expertise in Microsoft technologies. Apart from development work, Sergey is also experienced in the field of training, implementations consulting as well as team and customer relationship management.

Agenda

Day 1 – September 8

- Acumatica Architecture
- Training Initial Steps
- Configuring Drop-Downs
- Formulas
- Master-Detail
- ASPX Markup
- PXSelector Attribute
- Configuring a Complex based on Repair Work Order Form

Day 2 – September 9

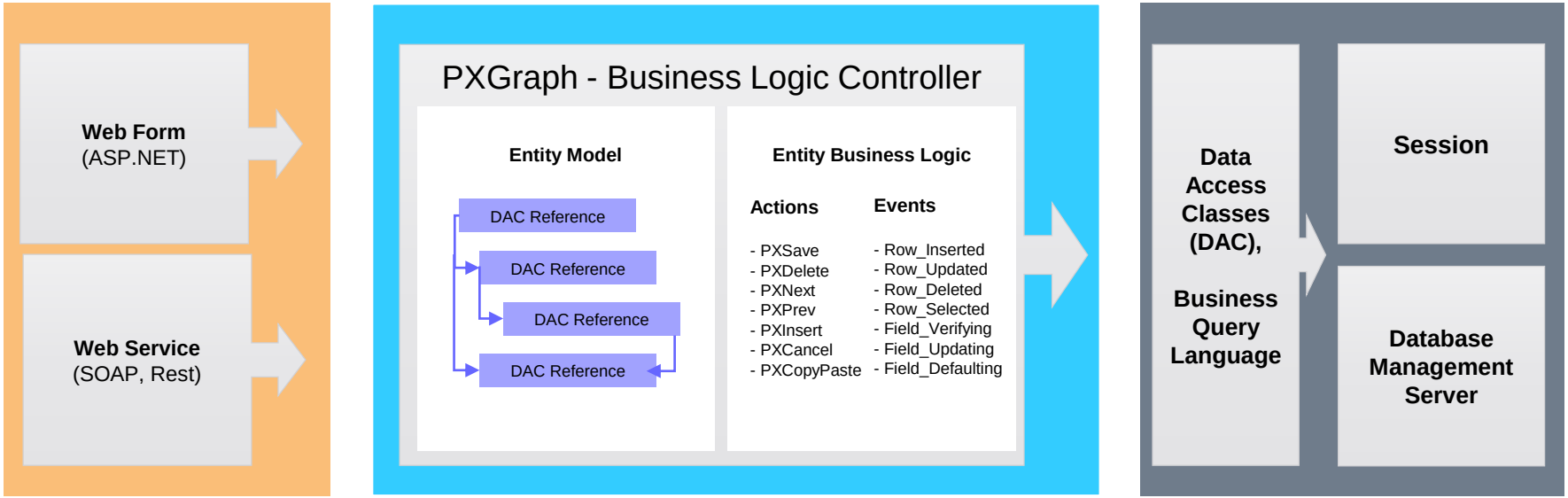
- Form View Mode
- RowTemplate
- Substitute Form with a Shared Filter
- Event Model
- Working with Cache
- PXLayoutRule with Complex Layout
- Declarative Style Programming
- Status Workflow

Day 3 – September 10

- Fields Validation
- Setup Forms
- PXSetup Data View
- PXPrimaryGraph Attribute
- System Configurations validations
- Auto-Numbering with CS.AutoNumberAttribute
- Use Dialogs

Acumatica Architecture

Acumatica Framework Architecture



Presentation Layer

Business Logic Layer

Data Access Layer

Acumatica Platform Essentials

- Data Access Classes (DAC)
- Business Logic Containers (Graph)
- Data Views
- Business Query Language (BQL)
- User Interface (ASP.NET)
- Events
- Customizations (Extensions)

Before we Start

Useful Development Environment Optimization

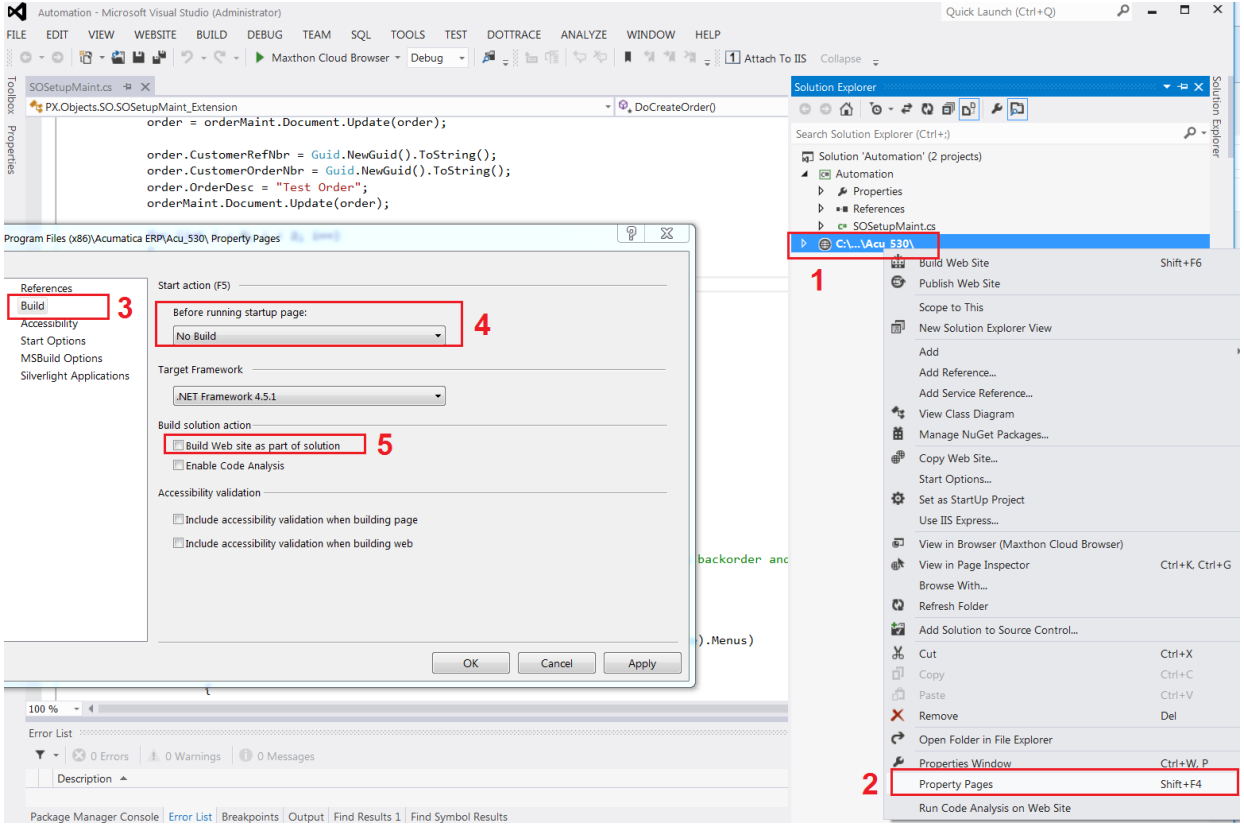
Web.config:

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

Useful Development Environment Optimization

Web Site compilation is slow

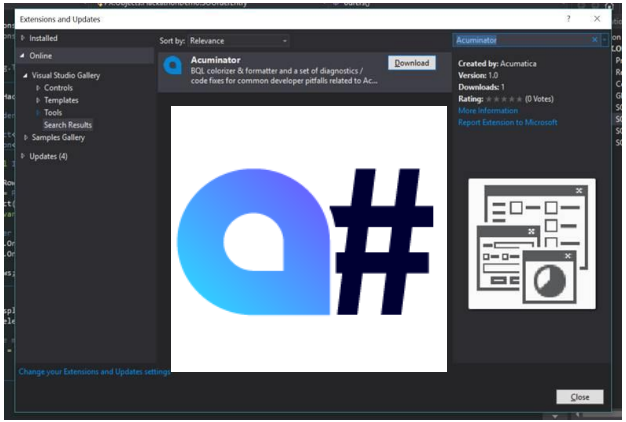
…and isn't necessary



Useful Development Environment Optimization

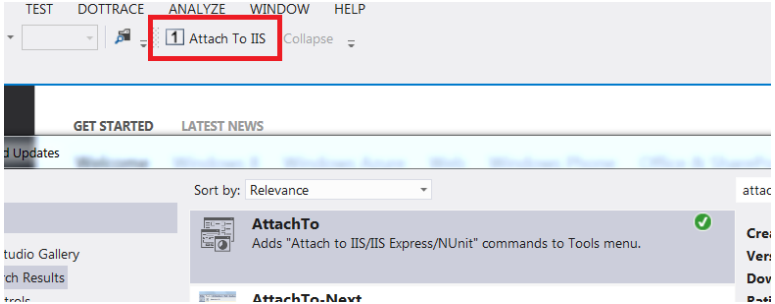
“Acuminator” Extension

Static code analysis, colorizer and suggestions tool for Acumatica Framework



“Attach To” Extension

- Attach Debugger to Acumatica with 1-click



Cheats Store

https://github.com/Acumatica/Help-and-Training-Examples

The Smart Fix company

Story

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

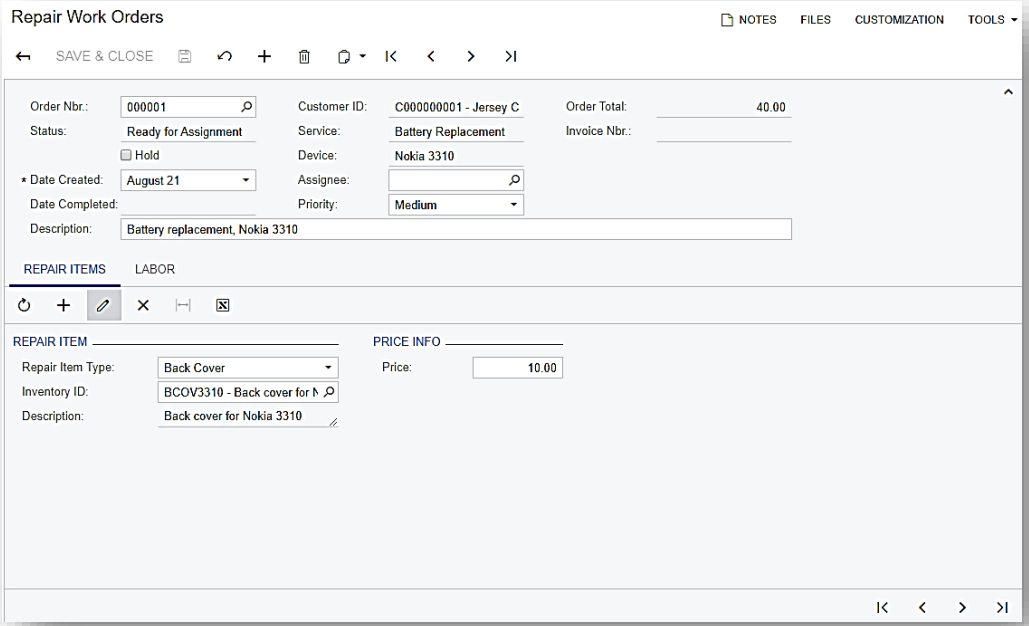
- Battery replacement: This service is provided on customer request and does not require any preliminary diagnostic checks.
- Repair of liquid damage: This service requires a preliminary diagnostic check and a prepayment.
- Screen repair: This service is provided on customer request and does not require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

T220: The Repair Work Orders Form

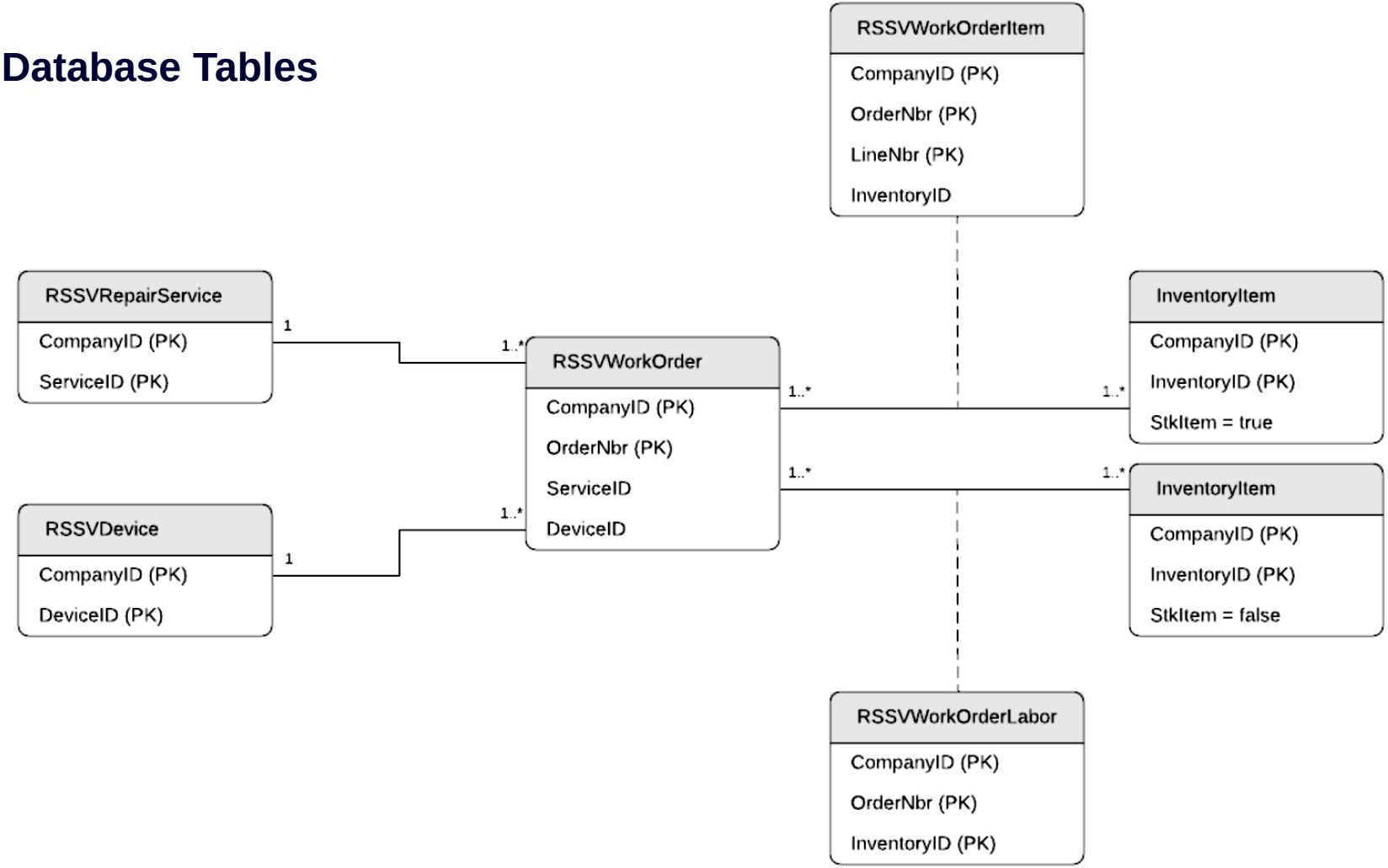
The Repair Work Orders Form

The form will contain the following tabs:

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



T220 Database Tables



Lesson 1.1: Configuring a Complex Form Layout

Objectives:

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

Step 1.1.1: Creating the Form

Attributes

Configuring Drop-Downs

What can you do to the drop-downs?

- Chose between `int` and `string` values
- Set the list of values that the *system* will see
- Set the list of values that a *user* will see

You will likely use these values elsewhere, so store them in some constants!

```
[PXStringList(
    new string[]
    {
        Status.OnHold,      // "H"
        Status.Shipping,    // "S"
        Status.Delivered    // "D"
    },
    new string[]
    {
        "Hold",
        "In progress",
        "Done"
    })]
public string Status { get; set; }
```

Configuring Drop-Downs

You can dynamically set the list of drop-down control.

```
PXStringListAttribute.SetList<Shipment.status>(sender, row,
      new string[] // List of values
      {
            ShipmentStatus.OnHold,
            ShipmentStatus.Shipping,
      },
      new string[] // List of labels displayed in the UI
      {
            "On Hold",
            "Shipping",
      });
```

PXSelector Attribute

What can you do to them?

- Tell what you want to select
- Restrict what you want to select
- Chose the fields to show in the selector table (even joine
- Replace the displayed key with a `SubstituteKey`
- Show a `DescriptionField`

And a bit more on the page side

Note: selectors perform validation!

```
[PXSelector(
  typeof(Search<Product.productID,
         Where<Product.productType,
                  Equal<coolProduct>>>),
  typeof(Product.productCD),
  typeof(Product.description),
  typeof(Product.productType)
  SubstituteKey =
         typeof(Product.productCD),
  DescriptionField =
         typeof(Product.description))]
public virtual Int32? ProductID
{
         get;
         set;
}
```

PXSelector - Accessing Related Records

`PXSelector` attribute is most commonly used to link entities together

Thus it can help at getting the related object:

```
var customer =
      (Customer)PXSelectorAttribute
            .Select<Invoice.customerID>(cache, invoice);
```

Handy, isn’t it?

PXRestiction Attribute

Restrictor search the Selector and inject extra conditions and error messages.

DAC:

```
[PXSelector(typeof(Search<Product.productID>)),
public virtual Int32? ProductID { get; set;}
```

Cache Extension:

```
[PXMergeAttribute(Method = MergeMethod.Append)
[PXRestrictorAttribute(
      typeof(Where<Product.active, Equal<True>>),
      Message.ViolationMessage
      )]
public virtual Int32? ProductID { get; set;}
```

Formulas

Formulas

```
[PXDBDecimal(2)]
[PXUIField(DisplayName = “Price")]
[PXDefault(TypeCode.Decimal, "0.0")]
[PXFormula(
        typeof(Mult<Line.lineQty, Line.unitPrice>),
        typeof(SumCalc<Document.totalPrice>))]
public virtual decimal? LinePrice { get; set; }
```

How to **calculate** the field

How to **aggregate** the field

Formulas

Calculate the value of its field:

- `Add<,>`
- `Sub<,>`
- `Mult<,>`
- `Div<,>`
- `Minus<>`
- `Switch<Case<>,>`

This one enables complex conditional calculations

Aggregate the resulting values into a *parent's* field with:

`SumCalc<>`

`CountCalc<>`

`MinCalc<>`

`MaxCalc<>`

Formulas

There is also `PXUnboundFormula` – unlike `PXFormula` it doesn't assign the calculated value to the field, which it is attached to

It calculates what it is told to and aggregates it in the parent, without changing anything in the detail record.

```
[PXDBDecimal(2)]
[PXDefault(TypeCode.Decimal, "0.0")]
[PXUnboundFormula(
      typeof(Switch<Case<Where<Current<ShipmentLine.cancelled>, Equal<False>>,
                        ShipmentLine.lineQty>,
                    decimal_0>),
      typeof(SumCalc<Shipment.totalQty>))]
public virtual decimal? LineQty { get; set; }
```

Note: unbound formula supports only `SumCalc` and `MaxCalc` aggregates.

Formulas

```
[PXFormula(typeof(Validate<…>))]
```

- formula will raise dependentField's FieldVerifying event each time the RelatedField is updated.

```
[PXFormula(typeof(Current<…>))]
```

- formula fetches the field value from the record stored in the Current property of the DAC’s cache.

```
[PXFormula(typeof(Parent<…>))]
```

- formula fetches the field value from the parent data record as defined by PXParentAttribute.

```
[PXFormula(typeof(IsTableEmpty<…>))]
```

- formula returns true if the DB table corresponding to the specified DAC contains no records.

```
[PXFormula(typeof(Selector<…>))]
```

- formula can evaluate and update value from foreign record referenced by selector.

```
[PXFormula(typeof(Default<…>))]
```

- Raises the FieldDefaulting for the field to which the formula is attached once the specified field changes.

Step 1.1.2: Configuring the Controls of the Summary Area

ASPX

Configuring ASP.NET pages

You set:

1. `TypeName` (fully qualified name of the graph) and `PrimaryView` (the name of the primary view) properties of the *data source control*
2. `DataSourceID` and `DataMember` (name of the view) properties of each data-bound container (`PXFormView`, `PXGridLevel` or `PXTab`):
3. `DataField` (name of the field) property of each and every data-bound control (e.g. `PXTextEdit`, `PXSelector`)

User Interface - ASPX

User Interface in Acumatica controls by ASPX files and LayoutRules

```
<px:PXFormView ID="form" DataMember="Products">
<Template>
      <px:PXLayoutRule StartRow="True"/>
      <px:PXSelector ID="edProductCD" DataField="ProductCD" />
      <px:PXTextEdit ID="edProductName" DataField="ProductName" />

      <px:PXLayoutRule StartColumn="True"/>
      <px:PXCheckBox ID="edActive" DataField="Active"/>
      <px:PXNumberEdit ID="edUnitPrice" DataField="UnitPrice"/>
</Template>
</px:PXFormView>
```

Data View

Relative Positioning Rules

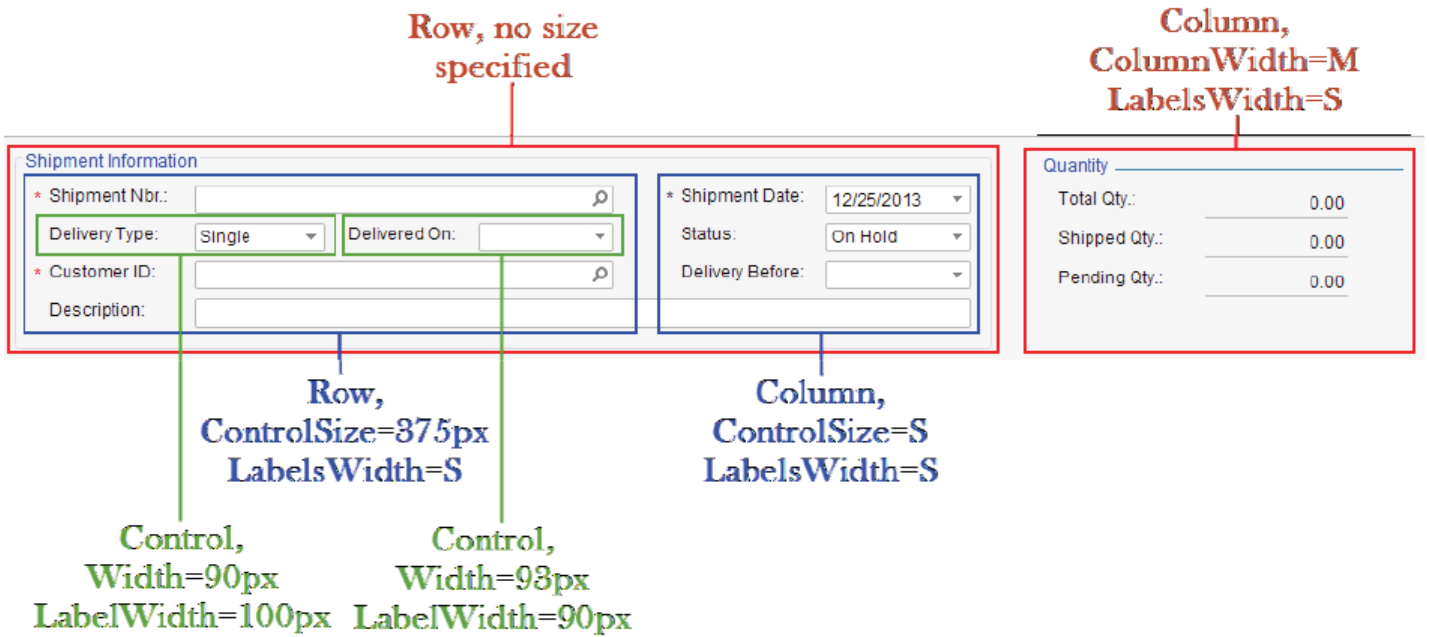
DAC Field

Attributes defines UI Controls

Control	Attributes on the DAC Field	ASPX Definition
Text box	[PX(DB)String]	<px:PXTextEdit ID= ... />
Number edit box	[PX(DB)Int] or [PX(DB)Decimal]	<px:PXNumberEdit ID=... />
Mask edit box	[PX(DB)String(InputMask =...)]	<px:PXMaskEdit ID=... />
Drop-down list	[PXStringList] or [PXIntList]	<px:PXDropDown ID=... / >
Selector	[PXSelector]	<px:PXSelector ID=... />
Check box	[PX(DB)Bool]	<px:PXCheckBox ID=... />
Date-time picker	[PX(DB)Date]	<px:PXDateTimeEdit ID=... />
Time span edit box	[PXDBTimeSpan]	<px:PXDateTimeEdit ID=... TimeMode="True" />

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

PXLayoutRule + PXPanel



Layout

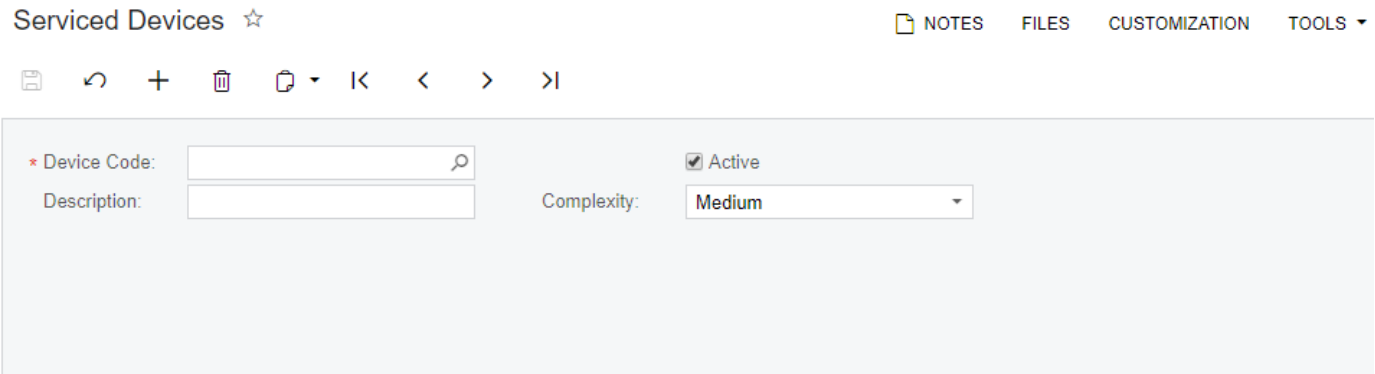
PXLayoutRule

```
<px:PXLayoutRule ID="PXLayoutRule1" runat="server"></px:PXLayoutRule>
```

The look of a form is determined by the contents of its `<Template>` element.

Any form template *must* start with a *Row* layout rule

Add *Column* rules to arrange the subsequent controls in a separate column



PXLayoutRule

You can position controls on the form using rules

- COLUMN (StartColumn=“true”) – all further controls will be positioned to start with a new column
- ROW (StartRow=“true”) – all further controls will be positioned to start with a new row
- GROUP – all further controls till next layout rule will be grouped together with header
- MERGE – all further controls till next layout rule will be merged together with single cell
- EMPTY – Ends Group or Merge Rules
- SPAN (ColumnSpan=“#") – Stretch field in multiple columns

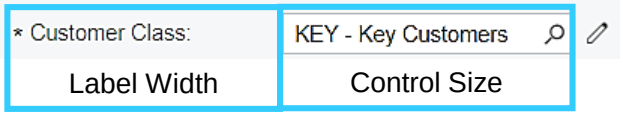
Layout Editing – Control Sizes

You can change size of the control with 2 properties:

- “ControlSize” (or Size) – Length of the editor
- “LabelsWidth” – Length of the label

T-Shirt Sizes:

- *XXS* (40px)
- *XS* (70px),
- ***S*** (100px),
- ***SM*** (150px),
- ***M*** (200px),
- ***XM*** (250px),
- *L* (300px),
- *XL* (350px),
- *XXL* (400px)



Override	Property	Value
☐	**Base Properties**	
☐	ColumnSpan	
☐	ColumnWidth	
☐	ControlSize	M
☐	EndGroup	
☐	GroupCaption	
☐	LabelsWidth	SM
☐	Merge	
☐	StartColumn	True
☐	StartGroup	

Step 1.1.4: Configuring Form View Mode for the Grid

RowTemplate

<RowTemplate>

RowTemplate allows you to have more settings than simple GridColumn.

```
<px:PXGridLevel DataMember="Locations">
	<RowTemplate>
		<px:PXSelector AutoRefresh="True" DataField="LocationID" />
	</RowTemplate>
	<Columns>
		<px:PXGridColumn CommitChanges="True" DataField="LocationID" />
	</Columns>
</px:PXGridLevel>
```

One case when you need `RowTemplate` is when you enable *form edit mode* for the grid.

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

Lesson 1.2: Copying Field Values from One Record to Another

Objectives:

- Copy the field values from one record to another record by using event handlers

Events Model

Events Declaration - Graphs

Classic:

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

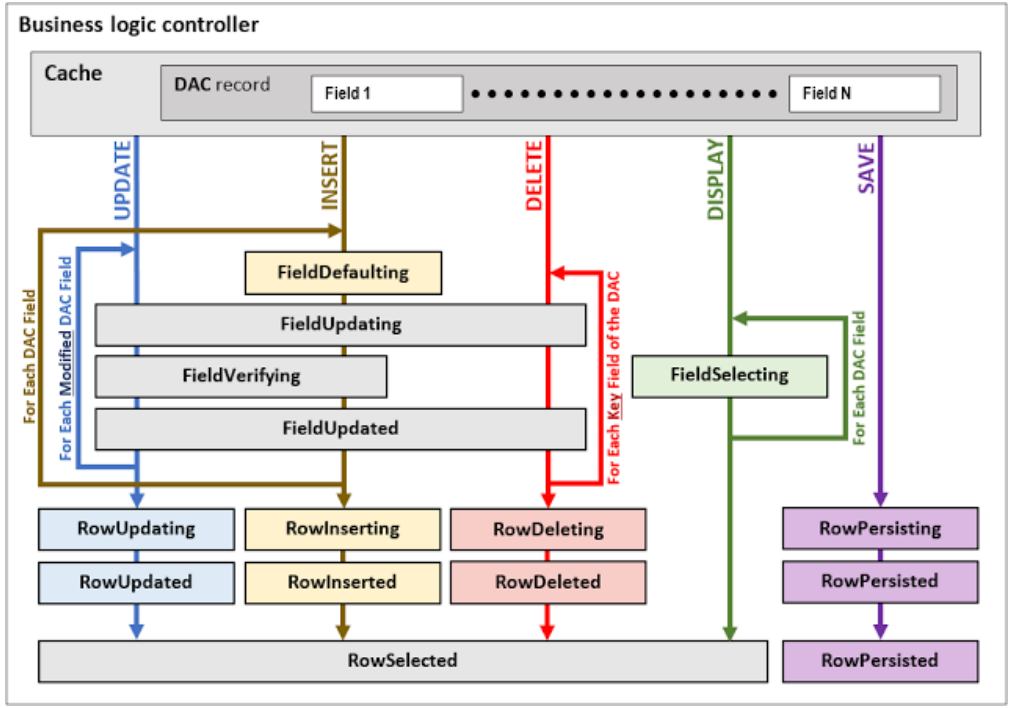
Generic:

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

Understanding The Event Model

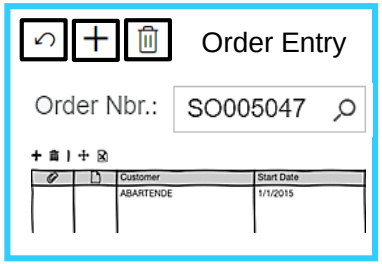
Acumatica events:

- 12 Row-level events
- 5 Field-level events



Where do events come from? - Example

Web Browser



ASP.NET Page

```
<px:PXDataSource
 TypeName="OrderEntry" />

<px:PXFormView
 DataMember="Orders">
 <Template>
  <px:PXSelector
   DataField="OrderNbr"/>
 </Template>
</px:PXFormView>
```

PXGraph

```
public class OrderEntry
 : PXGraph<OrderEntry, Order>
{

  public PXSelect<Order>
   Orders;

 public void
  _(Events.RowSelected<Order>
e)
 {
 }

}
```

PXView<Order>:
- Selects data from Database
- Provides data for the UI

PXCache<Order>:
- Maintains changes
- Executes Events sequence

Handling Rows Events

`RowInserting` – occurs before inserting into the cache
`RowInserted` – occurs after inserting into the cache

`RowUpdating` – occurs before updating of cached record
`RowUpdated` – occurs after updating of cached record

`RowDeleting` – occurs before deleting record from the cache
`RowDeleted` – occurs after deleting record from the cache

`RowPersisting` – occurs before saving record to the database
`RowPersisted` – occurs after saving record to the database

Handling Rows Events

`RowSelecting` – occurs system reads record from database reader
`RowSelected` – occurs when system should show record to the UI

`CommandPreparing` – occurs when system generates SQL script
`ExceptionHandling` – occurs when system handles the exception

Handling Fields Events

`FieldDefaulting` – occurs when system calc default value for the field.
`FieldVerifiyng` – occurs when need to validate input value.
`FieldUpdated` – occurs when the field value is changed.

`FieldSelecting` – occurs when need to show the field to the UI.
`FieldUpdating` – occurs when field is coming from the UI.

Do not forget to take care of `CommitChanges`

Modifying Data in Cache

Modifying Data in Cache

`var row = cache.Insert()` – simply inserts a new record into cache.

`var row = cache.Insert(object)` – does nothing and returns `null` if a record with the same keys already exists in the cache. Inserts the record and returns the *inserted* one otherwise.

`var row = cache.Update(object)` – updates the record if it is already in the cache. If there is no such record in the cache, gets it from the DB, puts into the cache and updates. If there is no suitable record in the DB, it is inserted with `Insert()`.

`var row = cache.Delete(object)` – sets `Deleted` status for the record. Similarly to the `Update(..)` will go to DB if fails to find the record in the cache. No, the record is not removed from the cache or the database. Feels crazy?

Modifying Data in Cache

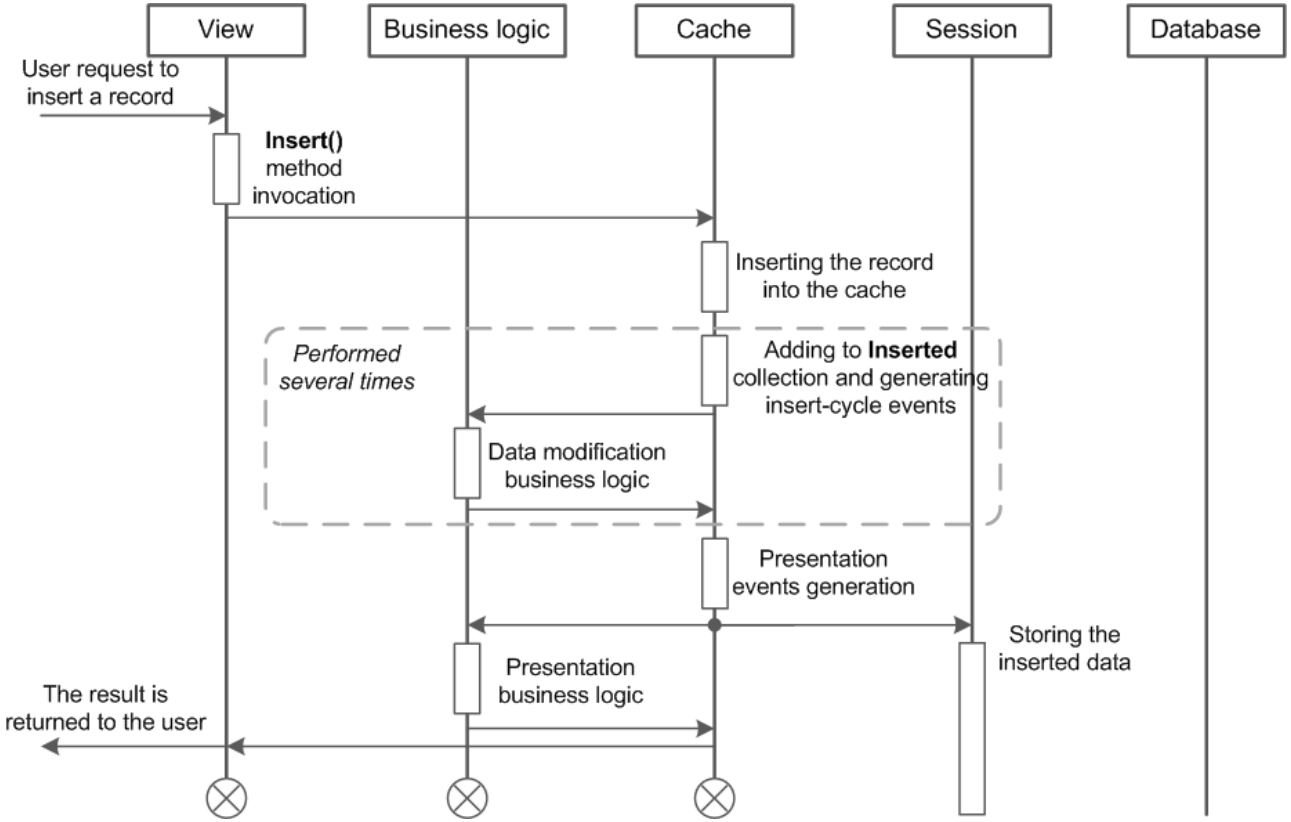
`cache.SetValueExt<DAC.field>(row, newValue)` – sets the field and raises `FieldUpdated` event. *Note*: record status is not changed, `RowUpdated` is not fired.

`cache.SetDefaultExt<DAC.field>()` – sets the default value of the field and raises all the required

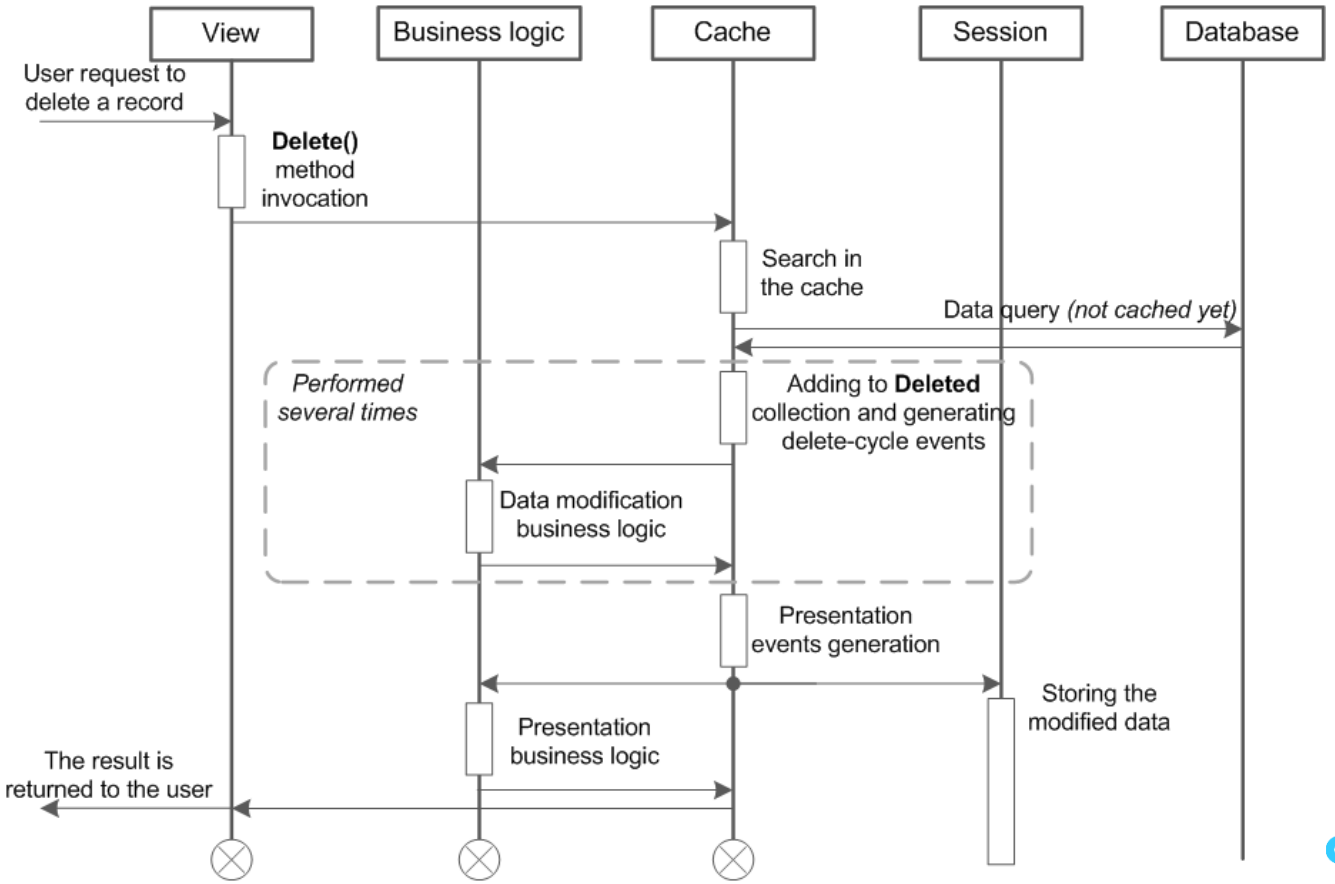
`cache.Locate(row)` (e.g. `var acct = Accounts.Cache.Locate(row)`) – looks up a record with matching keys in the cache. Doesn't go to the DB.

Can be done either on the cache object or through a data view instance.

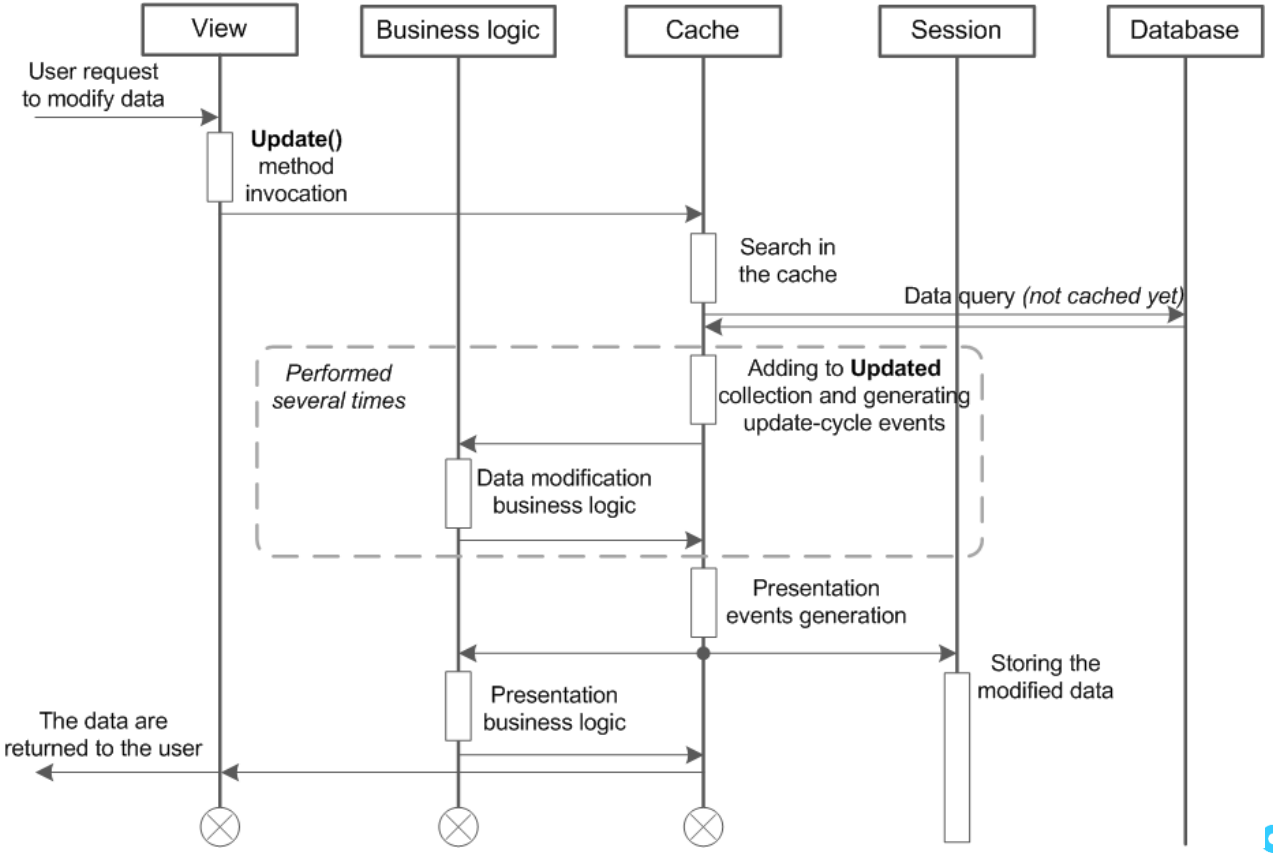
Inserting a New Entity Record



Deleting an Existing Entity Record



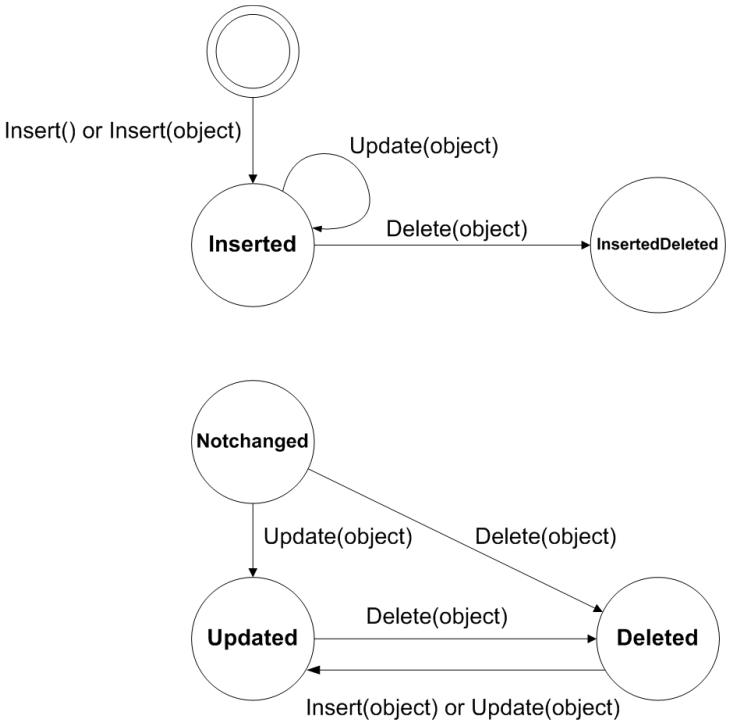
Updating an Existing Entity Record



Record Statuses

It’s pretty sane actually. Other statuses are:

- `Notchanged`
- `Updated`
- `Inserted`
- `Deleted`
- `InsertedDeleted`



Lesson 1.3: Implementing the Status Logic

Objectives:

- Implement the logic related to the statuses selected when the Hold check box is cleared
- Implement the run time UI logic by using the attributes

BQL - Constants

A BQL Constants is derived from the generic Constant<T>

```
public class decimal_100 : PX.Data.BQL.BqlString.Constant<Decimal>
{
        public decimal_100() : base(100m) { }
}
```

Usage:

```
public PXSelect<Product,
        Where<Div<Product.minAvailQty, decimal_100>,
        GreaterEqual<decimal_1>>> Records;

public SelectFrom<Contract>.
        Where<Contract.contractCD.IsEqual<defaultWarranty>>.
        View DefaultWarranty;
```

Adjusting UI Dynamically

Instead of PXUIEnabledAttribute.SetEnabled(…)

Declarative Style:

- Set Enabled

```
[PXUIEnabledAttribute(
        typeof(Where<InventoryItemExt.usrRepairItem, Equal<True>>))]
```

- Set Visible

```
[PXUIVisibleAttribute(
        typeof(Where<InventoryItemExt.usrRepairItem, Equal<True>>))]
```

Lesson 1.4: Validating the Field Values

Objectives:

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

Validating Data – Single Field

The most natural place to implement validation is the `FieldVerifying` event handler

However, `FieldVerifying` doesn't suit the case when validation logic depends on several fields – use `RowUpdating` under such circumstances

Anyway, if something is wrong you can simply:

```
throw new PXSetPropertyException("Danger!!!");
```

In addition to setting an error or warning sign on a particular field:

```
cache.RaiseExceptionHandling<ShipmentLine.lineQty>(
        line, e.NewValue,
        new PXSetPropertyException("Danger!!!", PXErrorLevel.Warning));
```

* Don't forget `CommitChanges` if you want the warnings right away!

Validating Data – Single Field

Instead of throwing errors at user you could also adjust the value yourself:

```
e.NewValue = product.MinAvailQty;
```

But be sure to let user know about the fact that something has changed with a warning icon and message on a field in UI:

```
cache.RaiseExceptionHandling<ShipmentLine.lineQty>(
      line,
      e.NewValue,
      new PXSetPropertyException("Changed!!!", PXErrorLevel.Warning));
```

Validating Data - Dependent Fields

When validation logic changes based on the values of other fields of the same record, which are declared earlier, we don't use `FieldVerifying`

We do use `RowUpdating`, whose handler has access to:

- `e.NewRow` – modified record, and
- `e.Row` – original version of the record

Given these, we can check whether any of the fields affecting validation have changed with `cache.ObjectsEqual` and validate if that's the case:

```
if(cache.ObjectsEqual<DAC.field1, DAC.field2>(e.NewRow, e.Row) ==
false)

{ /* Validation for the case field1 or field2 has changed */ }
```

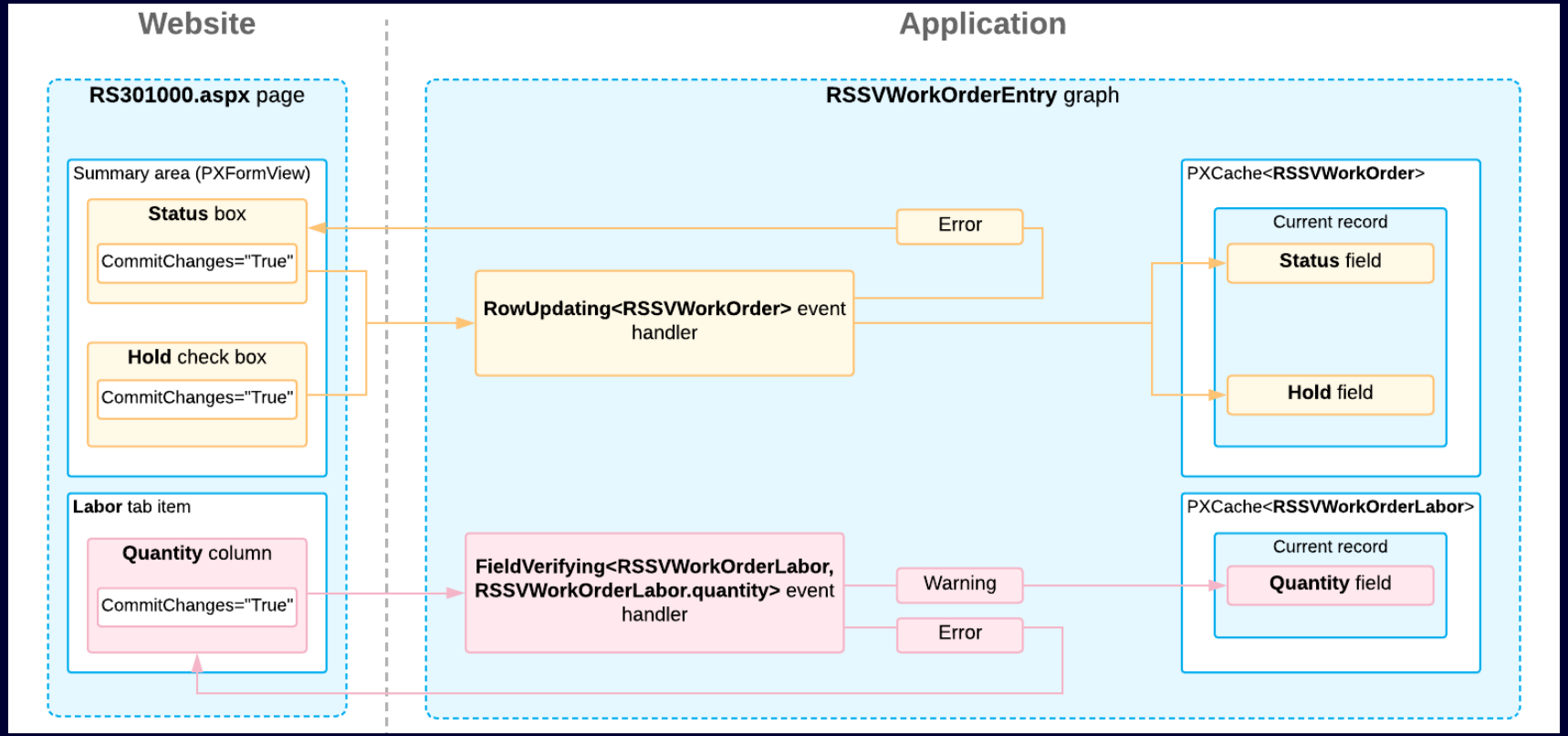
Validating Data - Dependent Fields

Instead of RowUpdating or FieldVerifying

Declarative Style:

```
[PXUIVerifyAttribute(
        typeof(Where<InventoryItemExt.ustMinQty, GreaterThan<Zero>>))]
```

Summary

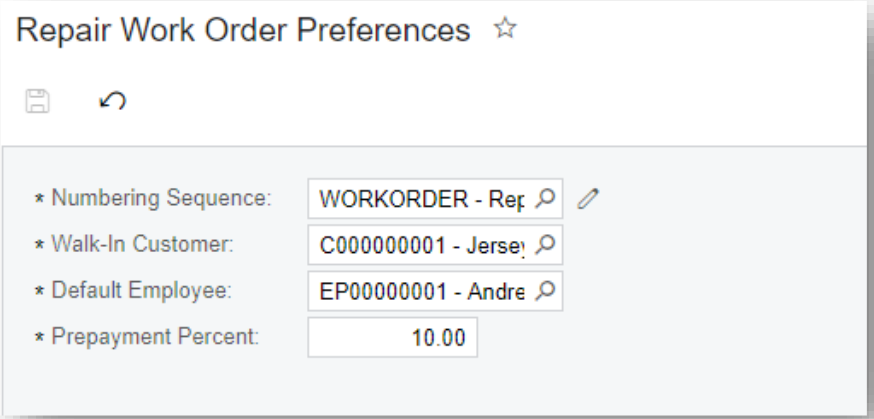

Website
Application
RS301000.aspx page
RSSVWorkOrderEntry graph
Summary area (PXFormView)
Status box
CommitChanges="True"
Hold check box
CommitChanges="True"
Labor tab item
Quantity column
CommitChanges="True"
RowUpdating<RSSVWorkOrder> event handler
FieldVerifying<RSSVWorkOrderLabor, RSSVWorkOrderLabor.quantity> event handler
Error
Warning
Error
PXCache<RSSVWorkOrder>
Current record
Status field
Hold field
PXCache<RSSVWorkOrderLabor>
Current record
Quantity field

T220: The Repair Work Order Preferences Form

The Repair Work Order Preferences Form

The form will contain the following:

- The **Numbering Sequence** box will hold the numbering sequence that is used to auto-number repair work orders.
- The **Walk-In Customer** box will contain the identifier of the customer record for walk-in orders.
- The **Default Employee** box will specify the default assignee for repair work orders.
- The **Prepayment Percent** box will contain the percent of prepayment that a customer should pay for a service that requires prepayment.



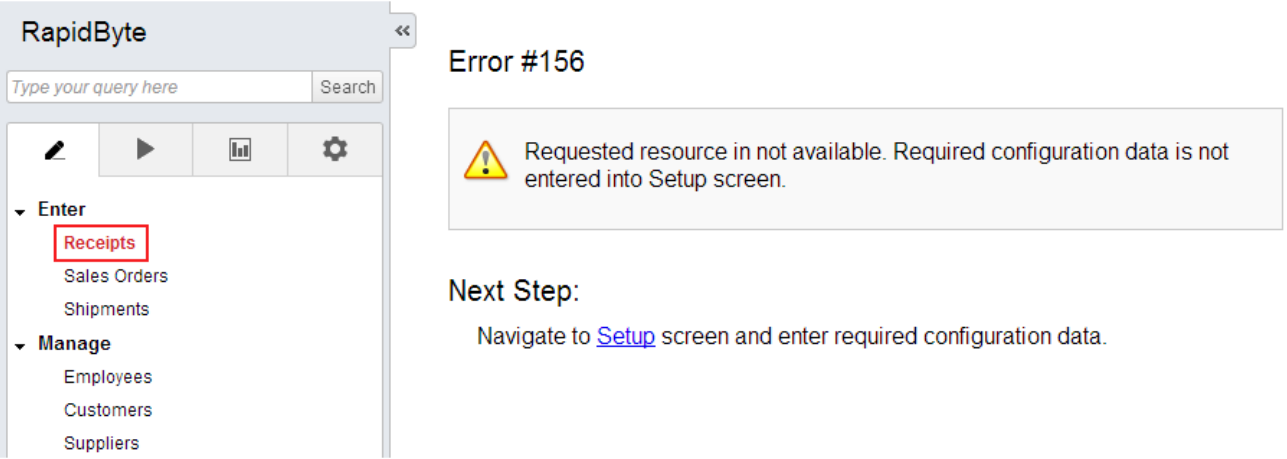
Lesson 2.1: Configuring the Auto-Numbering of a Field Value

Objectives:

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

PXSetup

Until a proper configuration entry is created user will see something like this on the dependent page:



PXSetup

To prompt user to configure a module before entering any documents:

`PXSetup` is a special kind of `PXSelect`, which makes this task easy. To tell that a graph needs certain piece of configuration you:

1. Declare a setup view within the graph

```
public PXSetup<Setup> setup;
```

2. Ask it for the `Current` member on graph construction

```
public ReceiptEntry()
{
        Setup setup = setup.Current;
}
```

This will throw a special exception upon failure to get a `Setup` record

PXPrimaryGraph

Noticed the Setup link below the warning?

`PXPrimaryGraphAttribute` makes navigating to a record possible

The attribute is attached to a DAC and tells the system which graph should it use to display a record of this DAC:

```
[Serializable]
[PXPrimaryGraph(typeof(SetupMaint)]
public partial class Setup : PX.Data.IBqlTable
{
      ...
}
```

PXPrimaryGraph

Works fine with any records – not only configuration ones

Can chose where to go based on the rules that you provide:

```
[PXPrimaryGraph(new Type[] {
        typeof(InvoiceEntry),
        typeof(PaymentEntry) },
      new Type[] {
        typeof(Select<Invoice,
              Where<Invoice.nbr, Equal<Current<Document.nbr>>>>),
        typeof(Select<Payment,
              Where<Payment.nbr, Equal<Current<Document.nbr>>>>) })]
public partial class Document : PX.Data.IBqlTable
```

destination graph

destination graph

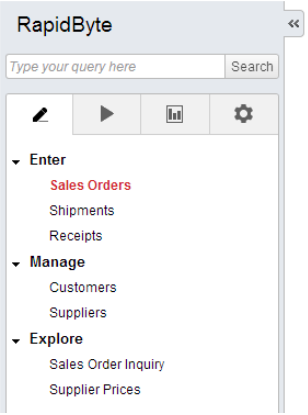
condition

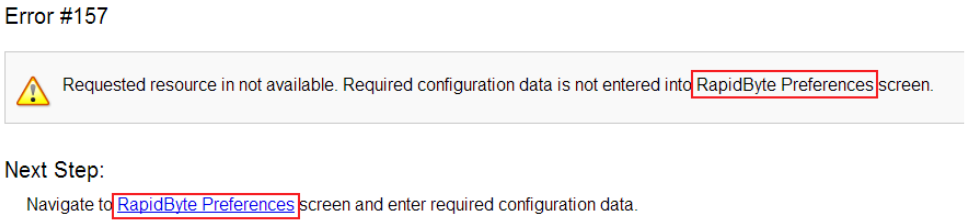
condition

PXCacheName

It is possible to set a user-friendly name for a DAC:

```
[System.SerializableAttribute()]
[PXPrimaryGraph(typeof(SetupMaint))]
[PXCacheName("RapidByte Preferences")]
public partial class Setup : PX.Data.IBqlTable
{
        //...
}
```





User Dialogs

Requesting Confirmations

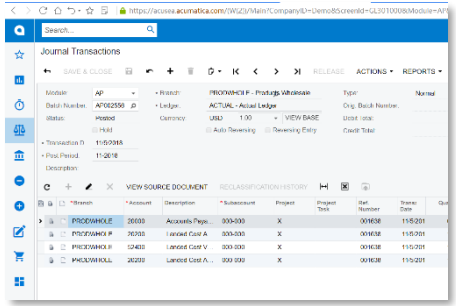
Throwing errors is a bit too one-sided way to communicate with users – what if we want to ask them something?

```
WebDialogResult res = this.PaymentMethod.Ask(
            "Confirmation",
            "Are you totally sure? Really?",
            MessageButtons.YesNo);
      if (res == WebDialogResult.Yes)
      {
            //Do things if user says 'Yes'.
      }
```

Also use the `e.ExternalCall` to check whether an event was initiated from UI (`true`) or from code (`false`)

User Dialogs

Web application is mostly one-way communication – Request -> Response



Get

HTML

Post

XML

Acumatica
THE CLOUD ERP

How to:

- Return control to user?
- Prevent operation modify data?

User Dialogs

Exceptions that returns control to base code:

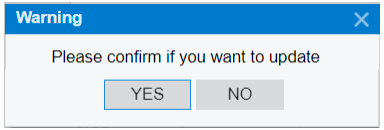
Browser | **Server**

Post -> Save

```
throw new PXDialogRequiredException("Key",
          null, "Messages"
          MessageButtons.YesNo,
          MessageIcon.Warning);
```

Response

Dialog Request



Post -> Answer

```
DialogAnswer answer
          = graph.Caches[typeof(DialogAnswer)]
          .Current.Answer.Value;
```

Overriding Attributes on Graph Level

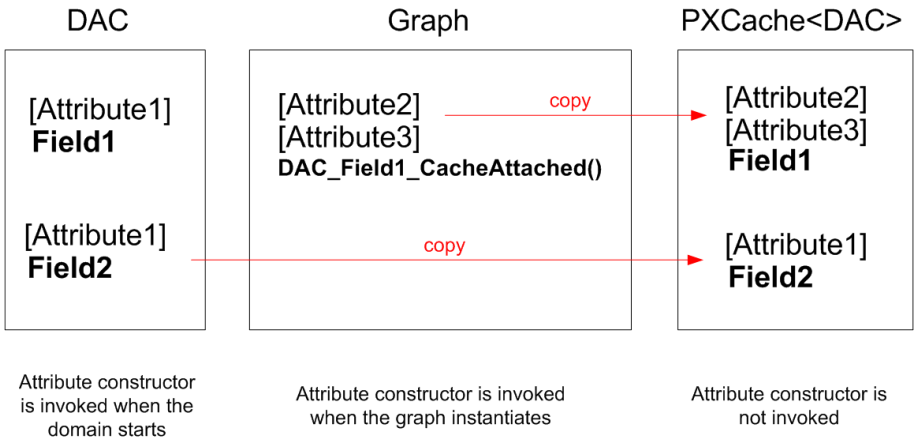
Overriding Attributes

Sometimes you want to make special configuration of a field in the scope of some particular graph – use `CacheAttached` event pseudo-handler

```
[PXDBString(15, IsUnicode = true, IsKey = true)]
[PXSelector(typeof(Product.productCD),
            typeof(Product.productCD),
            typeof(Product.productName),
            typeof(Product.unitPrice),
            typeof(Product.stockUnit))]
[PXUIField(DisplayName = "Product ID")]
protected virtual void Product_ProductCD_CacheAttached(PXCache sender)
{
}
```

No implementation – only the attributes – but *all* of them

Level of Attributes



1)Type – on domain initialization

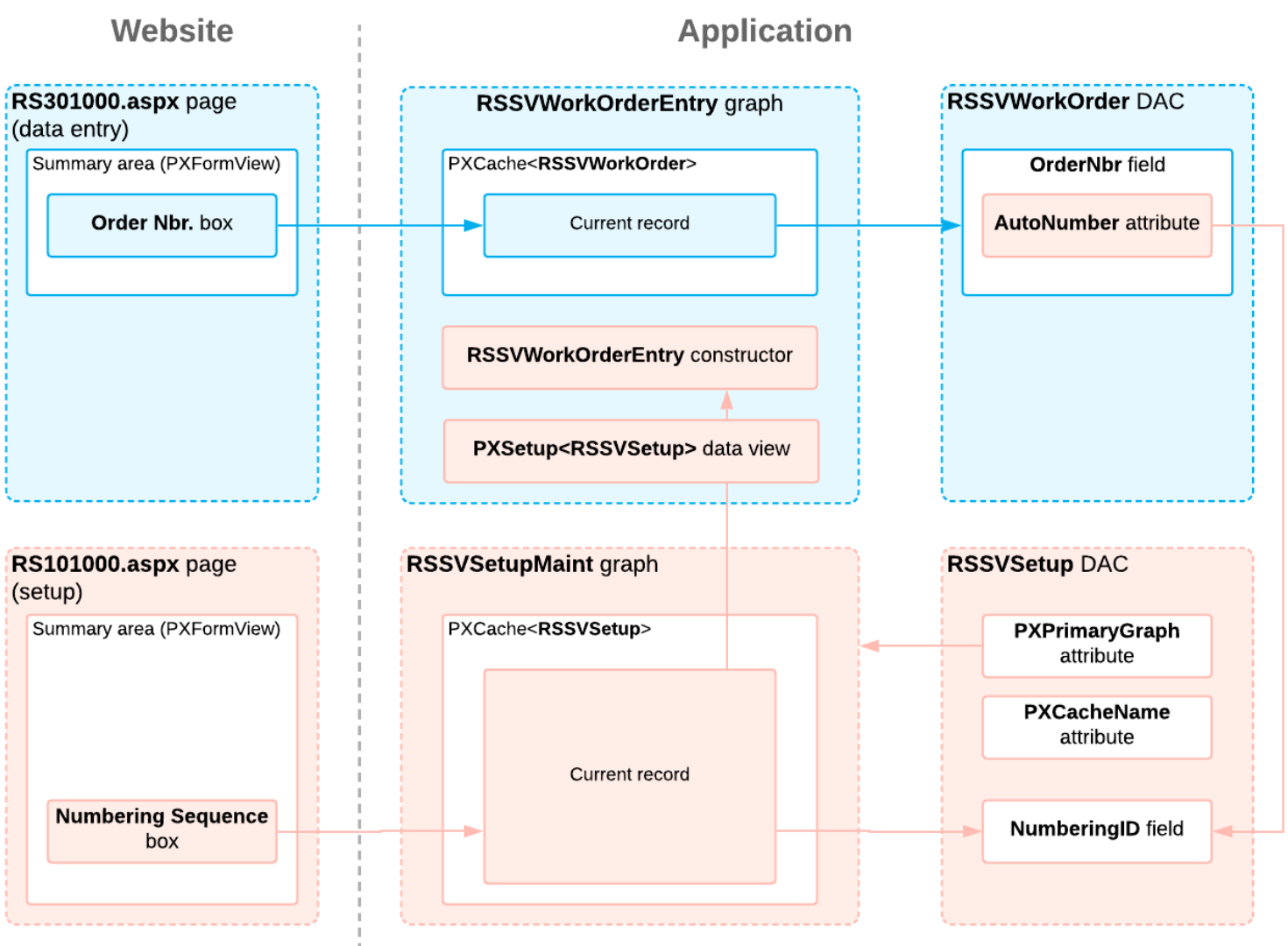
2)Cache – on cache initialization

```
PXUIFieldAttribute.SetEnabled<DAC>(cache, null, true);
```

3)Record – on record operation

```
PXUIFieldAttribute.SetEnabled<DAC>(cache, row, true);
```

Summary


Website
Application
RS301000.aspx page (data entry)
Summary area (PXFormView)
Order Nbr. box
RSSVWorkOrderEntry graph
PXCache<RSSVWorkOrder>
Current record
RSSVWorkOrderEntry constructor
PXSetup<RSSVSetup> data view
RSSVWorkOrder DAC
OrderNbr field
AutoNumber attribute
RS101000.aspx page (setup)
Summary area (PXFormView)
Numbering Sequence box
RSSVSetupMaint graph
PXCache<RSSVSetup>
Current record
RSSVSetup DAC
PXPrimaryGraph attribute
PXCacheName attribute
NumberingID field

Summary

Join Development Community

Acumatica Development Network (ADN)

- http://adn.acumatica.com/

Stack Overflow Community:

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

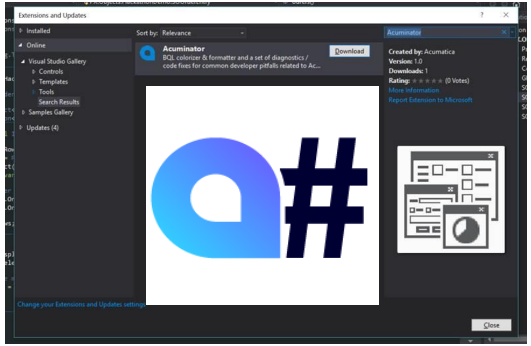
Git Hub opensource Projects

- https://github.com/Acumatica/

Visual Studio Extensions – Acuminator

Tons of Blogs

- http://asiablog.acumatica.com/
- http://blog.zaletskyy.com/Tags/Acumatica
- http://www.timrodman.com/tag/acumatica/



Acumatica Developer Network

GitHub

Blogger™

Acumatica
The Cloud ERP

Thank You

Sergey Marenich

http://asiablog.acumatica.com

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.