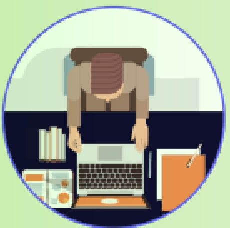
Roadmap

Mike Chtchelkonogov
Founder & Chief Technology Officer
Acumatica
mik@acumatica.com

Andrew Boulanov
Head of Platform Development
Acumatica
aboulanov@acumatica.com



Acumatica xRP Priorities

Platform Development & Modernization

Performance & Scalability Improvements

Low Code | No Code Customization

Improving Reporting Capabilities

Acumatica xRP Priorities

Platform Development and Modernization

- UI 2.0 Web.Forms elimination
- BQL 2.0 with LINQ (demo)
- Extensions 2.0 with unit tests support (demo)

Acumatica xRP Priorities

Performance and Scalability Improvements

- Constant performance assessment and monitoring
- Runtime and static code validations of potential performance issues
- Inquiry delegates and query cache optimization
- Lazy graph initialization

Acumatica xRP Priorities

Low Code / No Code Customization

- New workflow engine
  - State controller
  - Customizable actions
- Customizing entry form with attributes
- Per tenant customization
- Scripting customization layer

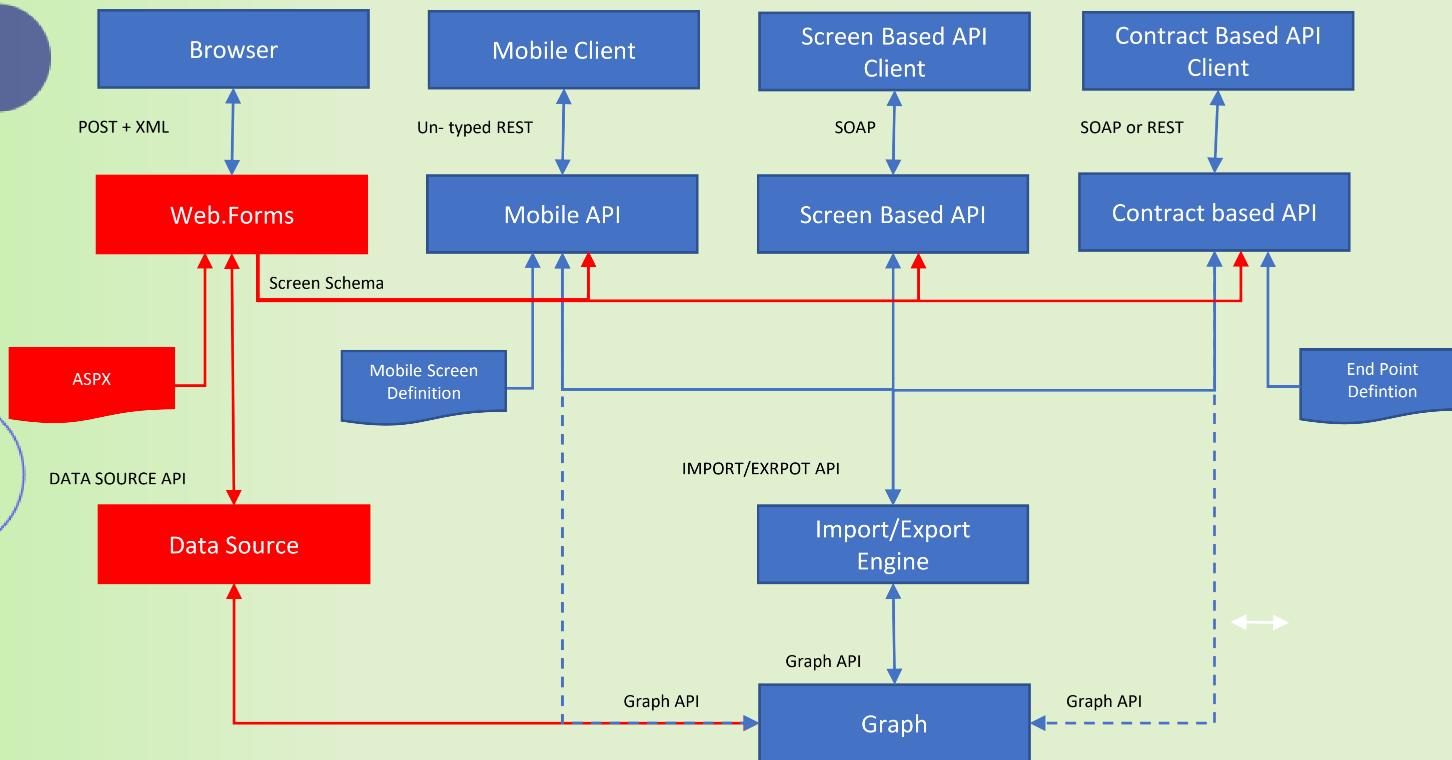
Acumatica xRP Priorities

Improving Reporting Capabilities

- New Pivots design (demo)
- Preview in Report Designer (demo)
- Improved ARM designer
- Relation suggestions in GI and improved GI designer
- Layout editor on top of GI

UI 2.0 What to expect and how will it work

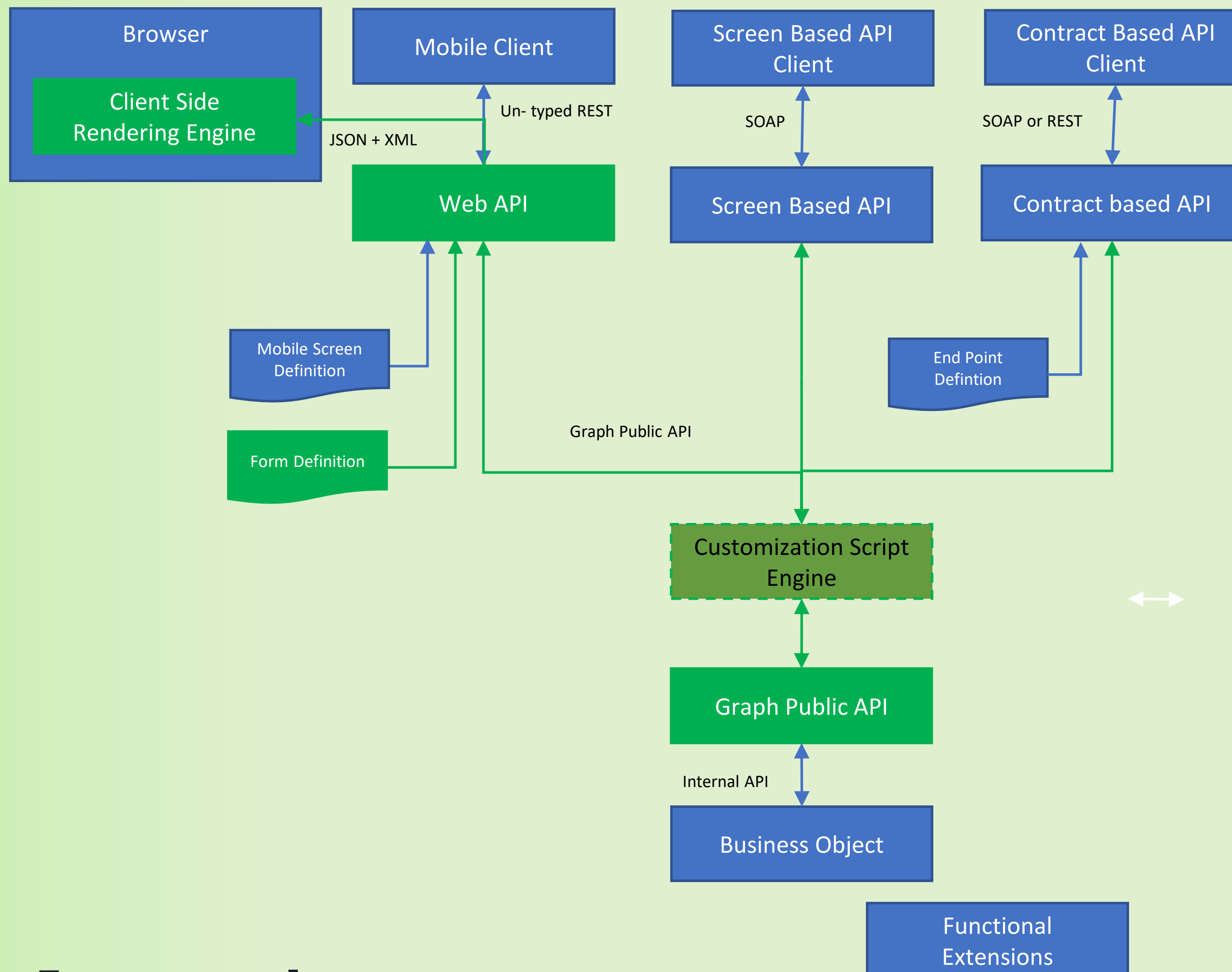
UI 2.0 - Current Frontend Architecture



- Web.Forms technology is legacy and going to be obsolete in 2-3 years
- Web.Forms technology is heavy, it consume up to 30% of CPU time and request execution time
- Datasource and Import/Export Engine provide two alternative options to access API
- Meta data generated out of ASPX form is used for other API's
- No public API is available for implementing scripting customization engine and tests

Acumatica
The Cloud ERP

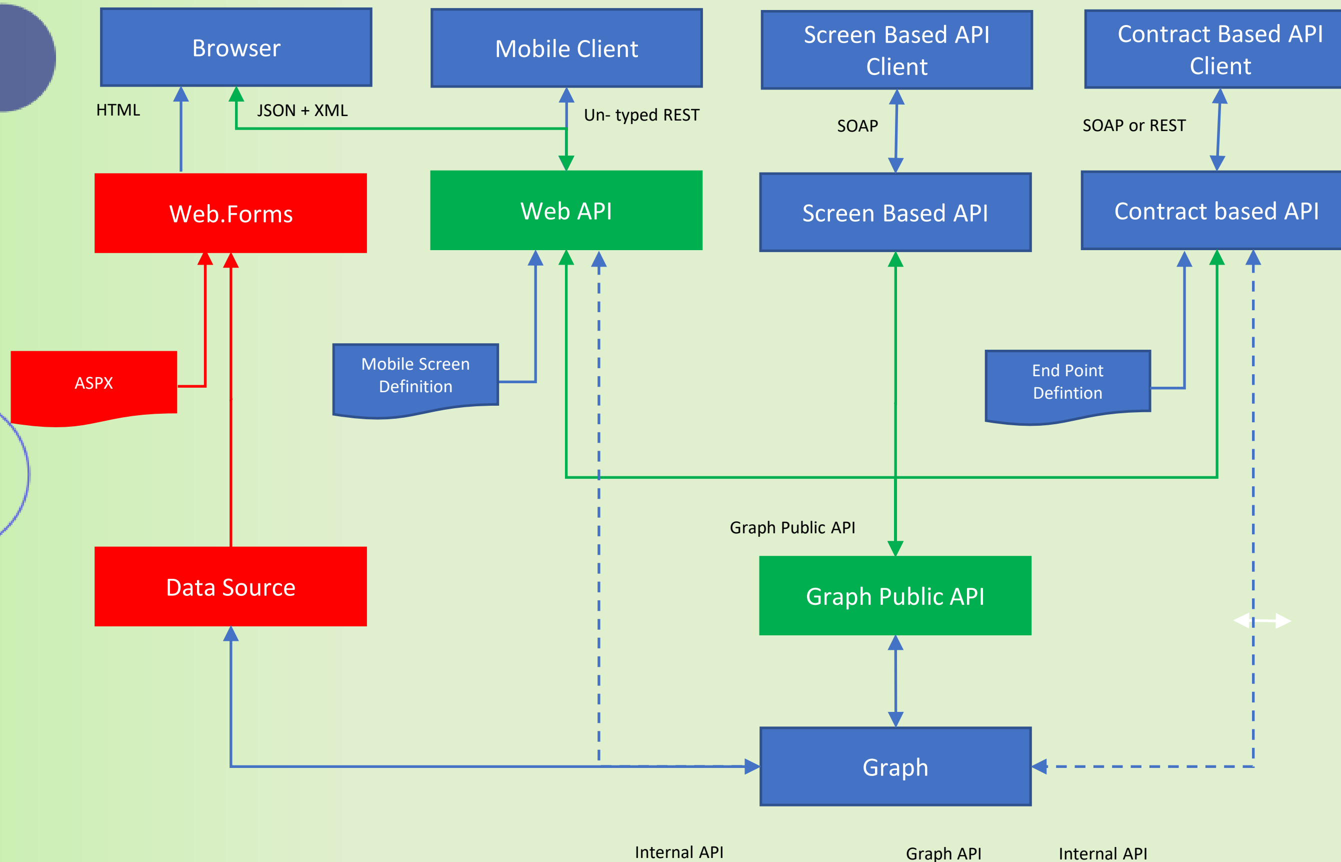
UI 2.0 - Future Frontend Architecture



- Web.Forms technology replaced with modern web api technology
- Mobile and Web API are uniform and can be unified as a single component
- Public graph API does not depend on UI definition and uniform for all external access introducing a public contract
- Public graph API creates a point for implementing customization script Engine
- Public graph API creates a point for implementing simplified integration tests against the business object

Acumatica
The Cloud ERP

UI 2.0 - Frontend Architecture – Migration Path



- Step 1 – Modify Mobile API to handle HTTP requests from browser and modify JS to bypass web forms and work though Web API.
- Step 1 – Work on Import/Export Engine to expose public graph API that will be uniform for all frontend engines.
- Step 2 – Replace Web Forms with new rendering engine.
- Step 3 – Inject scripting customization layer in front of Graph Public API engine.

Acumatica
The Cloud ERP

Programming API Enhancements

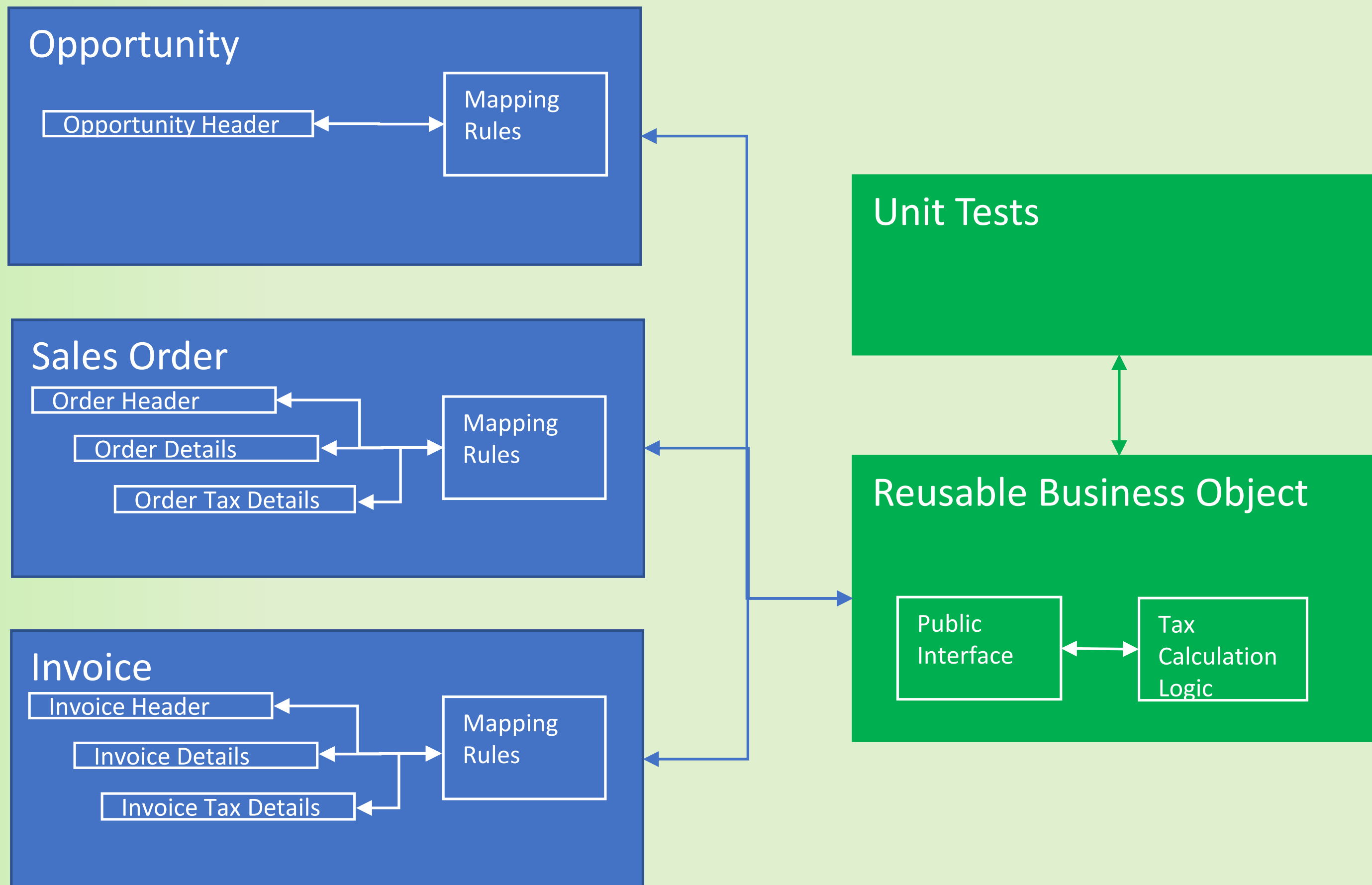
Functional Extensions

Unit Testing Framework

Object Layer in BQL

LINQ Support

Reusable Business Objects



- Eliminates duplicated code from the business objects.
- Isolates reusable pieces of business logic through the public API
- Supports use of the same logic on heterogeneous data structures through the mapping
- Regression testing can be automated through the unit tests
- Can be customized or substituted through the standard extensions customization mechanism

Multicurrency Extension

```
PX.Objects.Extensions.MultiCurrency.MultiCurrencyGraph<TGr    GetCuryID()

/// <summary>The generic graph extension that defines the multi-currency functio ...
public abstract class MultiCurrencyGraph<TGraph, TPrimary> : PXGraphExtension<TGraph>, IPXCurrencyHelper
        where TGraph : PXGraph
        where TPrimary : class, IBqlTable, new()
{
    IPXCurrencyHelper implementation

    Mappings

    #region Selects and Actions
    /// <summary>The current <see cref="CurrencyInfo" /> object of the document.</summary>
    public PXSelect<CurrencyInfo>
            currencyinfo;
    protected IEnumerable currencyInfo()...
    public PXSelect<CurrencyInfo, Where<CurrencyInfo.curyInfoID, Equal<Required<CurrencyInfo.curyInfoID>>>>
            currencyinfobykey;

    /// <summary>The <strong>Currency Toggle</strong> action.</summary>
    public PXAction<TPrimary> currencyView;
    [PXUIField(DisplayName = "Toggle Currency", MapEnableRights = PXCacheRights.Select, MapViewRights = PXCacheRights.Select)]
    [PXButton(ImageKey = PX.Web.UI.Sprite.Main.Money, Tooltip = CM.Messages.ToggleCurrencyViewTooltip)]
    protected virtual IEnumerable CurrencyView(PXAdapter adapter)...
    #endregion

    Initialization

    Currency Fields Processing
```

```
/// <summary>A mapped cache extension that represents
/// a document that supports multiple currencies.</summary>
public class Document : PXMappedCacheExtension
{
    #region BAccountID
    /// <exclude />
    public abstract class bAccountID : IBqlField
    {
    }
    /// <summary>The identifier of the business account of the document.
    public virtual Int32? BAccountID
    {
        get;
        set;
    }
    #endregion
    CuryID
    CuryViewState
    CurrencyRate
    CuryInfoID
    DocumentDate
}
```

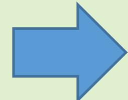
```
public interface IPXCurrencyService
{
    int BaseDecimalPlaces();
    int CuryDecimalPlaces(string curyID);
    int PriceCostDecimalPlaces();
    int QuantityDecimalPlaces();
    string DefaultRateTypeID(string moduleCode);
    IPXCurrencyRate GetRate(string fromCuryID, string toCuryID,
    int GetRateEffDays(string rateTypeID);
    string BaseCuryID();
    IEnumerable<IPXCurrency> Currencies();
    IEnumerable<IPXCurrencyRateType> CurrencyRateTypes();
}

public class DatabaseCurrencyService : IPXCurrencyService
{
```

- Multi-Currency Extension encapsulates everything related to the multi-currency feature in maintenance forms – from retrieving of currency list to recalculating base amounts in response to document modifications
- It connects to documents, details, vendor/customer records via some kind of interfaces called ‘mappings’, specifying only fields required for the multi-currency feature
- A special server is designed to provide Currencies, Rate Types, Rates and other info, making it easier to replace database storage with an external source
- Also Sales Tax, Sales Price, Discount, Contract Address extensions have been developed already, but not completely reworked with the new approach
- CRM Opportunities & Quotes, FS Appointments & Service Order entry forms already benefit from the functional extensions

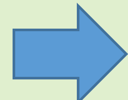
Advantages

```
if (!CompareIgnoreCase.IsInList(sender.Fields, _CuryRateField))
{
    sender.Fields.Add(_CuryRateField);
}
if (!CompareIgnoreCase.IsInList(sender.Fields, _CuryViewField))
{
    sender.Fields.Add(_CuryViewField);
}
```



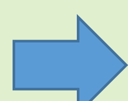
```
#region CurrencyRate
public abstract class curyRate : IBqlField { }
[PXDecimal]
public virtual decimal? CuryRate
{
    get;
    set;
}
#endregion
```

```
sender.Graph.FieldSelecting.AddHandler(_ChildType,
    _CuryRateField, curyRateFieldSelecting);
sender.Graph.FieldSelecting.AddHandler(_ChildType,
    _CuryIDField, curyIdFieldSelecting);
sender.Graph.FieldVerifying.AddHandler(_ChildType,
    _CuryIDField, curyIdFieldVerifying);
sender.Graph.FieldSelecting.AddHandler(_ChildType,
    _CuryViewField, curyViewFieldSelecting);
```



```
protected virtual void _(Events.FieldSelecting<Document,
{
    bool curyviewstate = Base.Accessinfo.CuryViewState;
    CurrencyInfo info = GetCurrencyInfo(e.Row);
    if (info != null) {
        if (!curyviewstate) {
            e.ReturnValue = info.SampleCuryRate;
        } else {
            e.ReturnValue = 1m;
        }
    }
}
```

```
private sealed class CurrencyInfoView : PXView {
    private CurrencyInfoAttribute _Owner;
    private PXView _innerView;
    public CurrencyInfoView(PXGraph graph, CurrencyInfoAttribute owner)
        : base(graph, false, new Select<CurrencyInfo, Where<CurrencyInfo.curyInfoID
        _Owner = owner;
        _innerView = new PXView(graph, false, new Select<CurrencyInfo, Where<Curren
    }
    public override List<object> Select(object[] currents, object[] parameters, obj
        searches = null;
        PXCache cache = _Graph.Caches[_Owner._ChildType];
        if (parameters == null || parameters.Length == 0 || parameters[0] == null)
        {
```

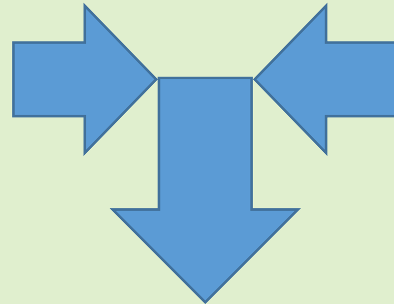


```
public PXSelect<CurrencyInfo>
        currencyinfo;
protected IEnumerable currencyInfo()...
public PXSelect<CurrencyInfo, Where<CurrencyInfo.curyInfoID,
    Equal<Required<CurrencyInfo.curyInfoID>>>>
        currencyinfobykey;
public PXAction<TPrimary> currencyView;
[PXUIField(DisplayName = "Toggle Currency", MapEnableRights =
[PXButton(ImageKey = PX.Web.UI.Sprite.Main.Money, Tooltip = C
protected virtual IEnumerable CurrencyView(PXAdapter adapter)
#endregion
```

- If you need a virtual field, you just declare a virtual field - no need to inject it dynamically
- If you need an event handler, you may just declare it like in a regular graph
- If you need a view or a button, you declare it within your functional extension – no need to create it on the fly
- It is possible to apply a customization on top of functional extensions overriding virtual methods, overriding of event handlers will be supported as well in the near future
- Even if your extension applies to a single screen, consider using this new approach which allows you to benefit from the unit testing framework

Unit Testing Framework

```
public class DocumentMock : PXCacheExtension<DocumentMockBase>
{
    public abstract class curyInfoID : PX.Data.IBqlField { }
    [PXDBLong]
    [CurrencyInfo(typeof(CurrencyInfo.curyInfoID))]
    public virtual Int64? CuryInfoID { get; set; }
    public abstract class curyID : PX.Data.IBqlField { }
    [PXDBString(5, IsUnicode = true, InputMask = ">LLLLL")]
    [PXSelector(typeof(Currency.curyID))]
    [PXUIField(DisplayName = "Currency")]
    public virtual String CuryID { get; set; }
    public abstract class baseField1 : IBqlField { }
    [PXDBDecimal(4)]
    [PXDefault(TypeCode.Decimal, "0.0")]
    public decimal? BaseField1 { get; set; }
    public abstract class curyField1 : IBqlField { }
    [PXDBCurrency(typeof(curyInfoID), typeof(baseField1))]
    [PXDefault(TypeCode.Decimal, "0.0")]
    public decimal? CuryField1 { get; set; }
```



```
public class BAccountMock : PXCacheExtension<BAccountMockBase>
{
    public abstract class curyID : IBqlField { }
    [PXDBString(5, IsUnicode = true)]
    public virtual String CuryID { get; set; }
    public abstract class curyRateTypeID : IBqlField { }
    [PXDBString(6, IsUnicode = true)]
    public virtual String CuryRateTypeID { get; set; }
    public abstract class allowOverrideCury : IBqlField { }
    [PXDBBool]
    [PXDefault(true)]
    public virtual Boolean? AllowOverrideCury { get; set; }
    public abstract class allowOverrideRate : IBqlField { }
    [PXDBBool]
    [PXDefault(true)]
    public virtual Boolean? AllowOverrideRate { get; set; }
}
```

ticurrency

Run All | Run... | Playlist : All

Pure (10+ tests)

Test	Duration
Passed Tests (10)	6 sec
PX.Objects.Extensi...	5 sec
PX.Objects.Extensi...	4 ms
PX.Objects.... (2)	104 ms
PX.Objects.Ex...	99 ms
PX.Objects.Ext...	5 ms
PX.Objects.Extensi...	9 ms
PX.Objects.Extensi...	7 ms
PX.Objects.E... (2)	27 ms
PX.Objects.Ex...	25 ms
PX.Objects.Ext...	2 ms
PX.Objects.Extensi...	4 ms
PX.Objects.Exten...	10 ms

Summary

Last Test Run Passed (Total Run Ti

PX.Objects.Tests | PX.Objects.ExtensionTests.MultiCurrency.Test | CuryDefaultRate(decimal doccury1, decimal docb

```
248    [Theory]
249    [InlineData("EUR", "BANK", false, false)]
250    [InlineData("JPY", "SPOT", true, true)]
251    public void DefaultFromSource(string curyID, string curyRateTypeID, bool curyEnabled, bool rateEnabled)
252    {
253        var graph = PXGraph.CreateInstance<GraphMock>();
254        var extension = graph.GetExtension<BindingMock>();
255
256        var source = graph.Source
257            .Insert().GetExtension<BAccountMock>();
258        source.CuryID = curyID;
259        source.CuryRateTypeID = curyRateTypeID;
260        source.AllowOverrideCury = curyEnabled;
261        source.AllowOverrideRate = rateEnabled;
262        graph.Source
263            .Update(graph.Source.Current);
264
265        var doc = graph.Documents
266            .Insert().GetExtension<DocumentMock>();
267
268        CurrencyInfo info = extension.currencyinfo.Current;
269        PXCache cache = extension.currencyinfo.Cache;
270
271        Assert.Equal(info.CuryID, curyID);
272        Assert.Equal(info.CuryRateTypeID, curyRateTypeID);
273        Assert.Equal(((PXFieldState)graph.Documents.Cache.GetStateExt<DocumentMock.curyID>(graph.Documents.Current)
274        Assert.Equal(((PXFieldState)cache.GetStateExt<CurrencyInfo.sampleCuryRate>(info)).Enabled, rateEnabled);
275    }
```

- Multi-Currency Extension connects to sample document, detail, and business account classes
- A graph mock combines them all together
- The test inserts desired objects into the graph, which completely emulates database content
- A currency service is emulated as well
- After execution of the business logic, the test verifies that the Multi-Currency Extension properly sets currency and rate types at the document level, and also properly either enables or disables appropriate fields from the test method parameters

Object Layer in BQL

```
int idx;
if (!String.IsNullOrWhiteSpace(e.FieldName)
    && e.FieldName.StartsWith(BqlCommand.SubSelect + nameof(Creator.displayName), StringComparison.OrdinalIgnoreCase)
    && (idx = e.FieldName.IndexOf(" FROM ", 7, StringComparison.OrdinalIgnoreCase)) != -1)
{
    ISqlDialect dialect = sender.Graph.SqlDialect;
    e.FieldName = BqlCommand.SubSelect +
        dialect.functions2sql("switch",
            new string[] {
                dialect.quoteTableAndColumn(_Type.Name + "Ext", "FirstName") + " IS NULL",
                dialect.functions2sql("switch",
                    new string[] {
                        dialect.quoteTableAndColumn(_Type.Name + "Ext", "LastName") + " IS NULL",
                        dialect.quoteTableAndColumn(_Type.Name + "Ext", "UserName"),
                        dialect.quoteTableAndColumn(_Type.Name + "Ext", "LastName") }),
                dialect.functions2sql("switch",
                    new string[] {
                        dialect.quoteTableAndColumn(_Type.Name + "Ext", "LastName") + " IS NULL",
                        dialect.quoteTableAndColumn(_Type.Name + "Ext", "FirstName"),
                        dialect.functions2sql("concat",
                            new string[] {
                                dialect.quoteTableAndColumn(_Type.Name + "Ext", "FirstName"),
                                "' '",
                                dialect.quoteTableAndColumn(_Type.Name + "Ext", "LastName") }) }) })
        + e.FieldName.Substring(idx);
Query q = (e.Expr as SubQuery)?.Query();
if (q!=null && q.GetSelection().Count>0 && (q.GetSelection()[0] as Column)?.Name.Equals(nameof(Creator.displayName), StringComparison.OrdinalIgnoreCase)==true) {
    SimpleTable tbl = new SimpleTable(_Type.Name + "Ext");
    SQLSwitch extcase = new SQLSwitch().Case(
        new Column("FirstName", tbl).IsNull(),
        new SQLSwitch().Case(
            new Column("LastName", tbl).IsNull(),
            new Column("UserName", tbl)
        ).Default(new Column("LastName", tbl))
    ).Default(
        new SQLSwitch().Case(
            new Column("LastName", tbl).IsNull(),
            new Column("FirstName", tbl)
        ).Default(
            new Column("FirstName", tbl)
                .Concat(new SQLTree.Constant(" "))
                .Concat(new Column("LastName", tbl))
        )
    );
```

- BQL classes are immutable and produce SQL text directly in Parse methods
- Particular fields get their SQL representation in the CommandPreparing event handlers returning text as a field name
- Will remove potential security holes when CommandPreparing event returns SQL text
- Will help to get rid of SQL text post-processing; like adding company id and company mask restrictions, top counts, etc.
- Also this new functionality will replace the flattening procedure

LINQ Support

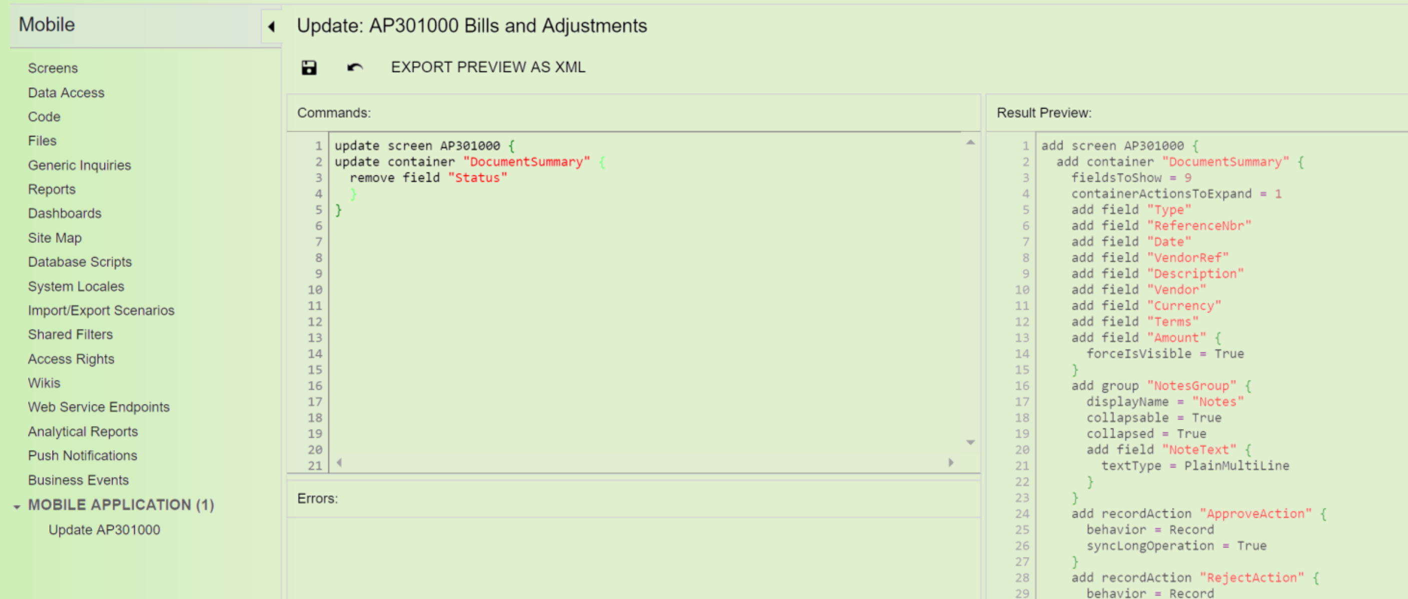
```
var record =
PXSelectJoin<INItemSiteSettings,
        LeftJoin<POVendorInventory,
            On<POVendorInventory.inventoryID, Equal<INItemSiteSettings.inventoryID>,
            And<POVendorInventory.active, Equal<boolTrue>,
            And<POVendorInventory.vendorID, Equal<Required<Vendor.bAccountID>>,
            And<Where<POVendorInventory.subItemID, Equal<Required<POVendorInventory.subItemID>>,
                    Or<POVendorInventory.subItemID, Equal<INItemSiteSettings.defaultSubItemID>,
                    Or<POVendorInventory.subItemID, IsNull,
                    Or<Where<Required<POVendorInventory.subItemID>, IsNull,
                         And<POVendorInventory.subItemID, Equal<True>>>>>>>>>>>>,
    Where<INItemSiteSettings.inventoryID, Equal<Required<INItemSiteSettings.inventoryID>>,
        And<INItemSiteSettings.siteID, Equal<Required<INItemSiteSettings.siteID>>>>>
        .Select(graph, vendorID, subItemID, subItemID, itemID, siteID)
        .Select(r => new { Item = r.GetItem<POVendorInventory>(), Site = r.GetItem<INItemSiteSettings>() })
        .Where(r => r.Item != null && r.Site != null)
        .OrderBy(r => r.Item.LastPrice)
        .ThenByDescending(r => r.Item.SubItemID == r.Site.
        .ThenByDescending(r => r.Item.VendorLocationID !=
        .ThenByDescending(r => r.Item.IsDefault == true)
        .ThenByDescending(r => r.Item.VendorLocationID ==
        .FirstOrDefault();
```

```
Where<INItemSiteSettings.inventoryID, Equal<Required<INItemSiteSettings.inventoryID>>,
    And<INItemSiteSettings.siteID, Equal<Required<INItemSiteSettings.siteID>>>>>
    .Select(graph, vendorID, subItemID, subItemID, itemID, siteID)
    .Select(r => new { Item = r.GetItem<POVendorInventory>(), Site = r.GetItem<INItemSi
    .Where(r => r.Item != null && r.Site != null)
    .OrderBy(r => r.GetItem<POVendorInventory>().LastPrice)
    .ThenByDescending(r => r.GetItem<POVendorInventory>()
        .SubItemID == r.Record.DefaultSubItemID)
    .ThenByDescending(r => r.GetItem<POVendorInventory>()
        .VendorLocationID != null)
    .ThenByDescending(r => r.GetItem<POVendorInventory>()
        .IsDefault == true)
    .ThenByDescending(r => r.GetItem<POVendorInventory>()
        .VendorLocationID == vendor?.DefLocationID)
    .FirstOrDefault();
```

```
var record =
(from site from PXSelect<INItemSiteSettings>.Select(graph)
           join poitem in
                     (from filtered PXSelect<POVendorInventory>.Select(graph)
     where filtered.Record.Active == true
                && filtered.Record.Active == vendorID
                && filtered.Record.SubItemID == subItemID
     select filtered)
                on new { InventoryID = site.Record.InventoryID,
                         SubItemID = site.Record.DefaultSubItemID }
                equals new { poitem.Record.InventoryID,
                         poitem.Record.SubItemID }
                into j1
           from item in j1.DefaultIfEmpty()
           where item.Record.Active == true
                     && item.Record.VendorID == vendorID
                     && item.Record.SubItemID == subItemID
           orderby item.Record.LastPrice descending,
                     item.Record.SubItemID == site.Record.DefaultSubItemID,
                     item.Record.VendorLocationID != null,
                     item.Record.IsDefault == true,
                     item.Record.VendorLocationID == vendor.DefLocationID
           select new { item.Record.VendorLocationID,
               site.Record.PreferredVendorID,
               site.Record.PreferredVendorLocationID
           }).FirstOrDefault();
```

- Select method will not go to SQL server immediately anymore
- PXSelectResult will implement IQueryable interface
- It will give us the ability to utilize LINQ, and also to make performance optimizations, adding top 1 expression when result is converted into a single DAC instance
- Where a query can be difficult to convert into LINQ, it will be possible to make adjustments in BQL using expressions

Mobile Site Map



- MSDL only based, conversion tool from XML is available
- New maintenance screen makes it much easer to create and update customizations to mobile site map
- The screen validates MSDL on every save with error reporting and live resulting site map preview
- Customizations may be different for different tenants

Thank You!

https://adn.Acumatica.com