A walkthrough of a Business Intelligent Reporting Framework

Using:

.NET, ADO.NET Datasets and XML

When is a reporting framework Business Intelligent?

- When it enables creation of reports which are not simple reports but detailed business analysis perspectives/insights into valuable information.
- When it can do so intuitively even with the most complex of requirements and yet allowing the user to extrapolate by maneuvering various variables and inputs on the fly.
- When it does so without requiring code-recompilation at any stage
- When it is ubiquitous and agnostic to underlying database allowing for flexibility of choice to inputs and variables
- When it allows for optimum resource utilization minimizing the need to open database connections and run SQL Queries over and again for same set of data using intelligent caching.
- When is it futuristic and allows scalability and expansion of new features/platforms/domains with minimum coding [The “Don’t Repeat Yourself” paradigm]
- When it integrates well and seamlessly with other components of the bigger software.

Using ADO.NET Datasets with XML

- Let’s now look at how .NET Datasets when used craftily with XML, lead us to the inception of an extremely powerful yet flexible framework for intelligent business reporting.

ADO.NET Datasets- Benefits

- Binds seamlessly with UI controls as well as most of databases allowing for automatic population of UI controls with data
- Can work in offline/disconnected mode allowing for powerful filter queries without multiple roundtrips to the source database
- Can be persisted to local hard-disk as an XML document with Schema information as XSD and can be loaded offline again.
- Inherently XML enabled [Gets converted to XML and back with one .NET method call]
- Syncs automatically with databases on connection restoration
- Can integrate with legacy VBA code as ADO Recordset

XML- Benefits

- De-facto standard for ubiquitous and technology agnostic information exchange
- Can be easily casted into valid HTML as XHTML and XSLT for displaying powerful reports and printing them off-the-shelf.
- Can be used to intake values fed to UI controls from any desktop/web application front-end and insert them as input variables to SQL Queries **which are also** embedded within XML tags making the Reports run completely outside the compiled code
- Can work well with Legacy Code and Access applications

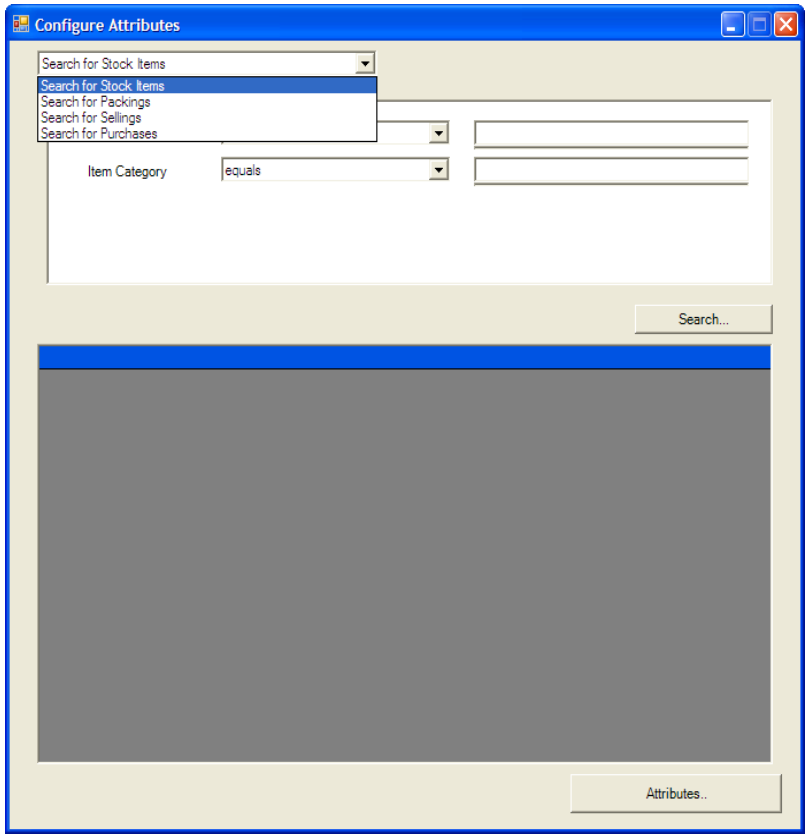
Result

- ADO.NET Datasets + XML= A powerful Business Intelligent Framework for complex reporting needs of today.

- Framework Insights

User selects type of report

UI

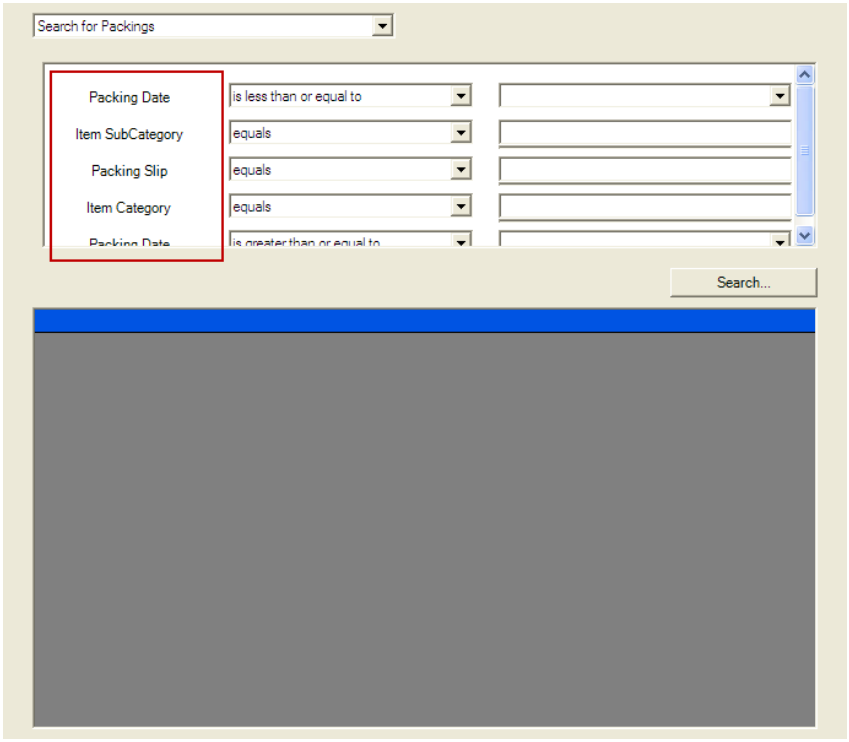


XML

```
- <Schema>
  + <SearchCriteria Value="Search for Stock Items">
  + <SearchCriteria Value="Search for Packings">
  + <SearchCriteria Value="Search for Sellings">
  + <SearchCriteria Value="Search for Purchases">
  </Schema>
```

User selects input variables

UI

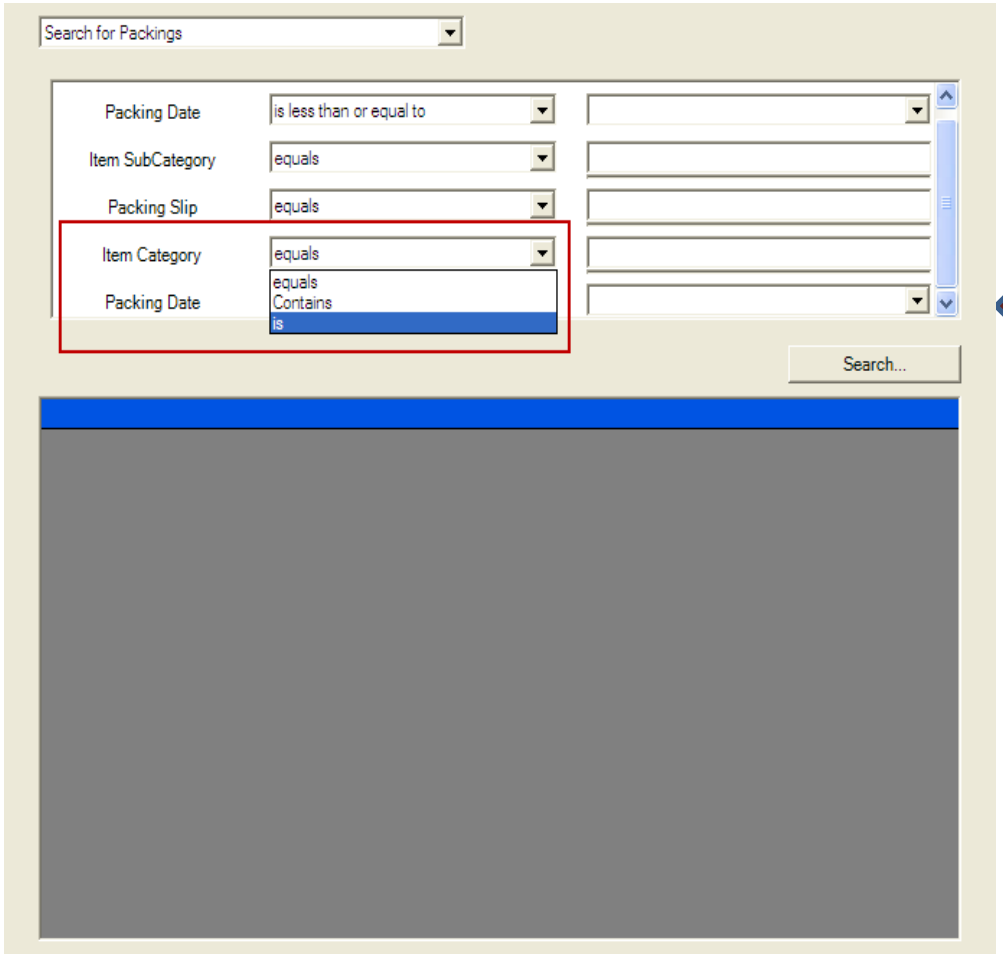


XML

```
- <Schema>
  + <SearchCriteria Value="Search for Stock Items">
  - <SearchCriteria Value="Search for Packings">
    + <ReportCondition ID="Item Category">
    + <ReportCondition ID="Item SubCategory">
    + <ReportCondition ID="Packing Slip">
    + <ReportCondition ID="Packing Date">
    + <ReportCondition ID="Packing Date">
    + <ResultSet BindTo="Packings">
    </SearchCriteria>
  + <SearchCriteria Value="Search for Sellings">
  + <SearchCriteria Value="Search for Purchases">
  </Schema>
```

User selects filter criteria

UI

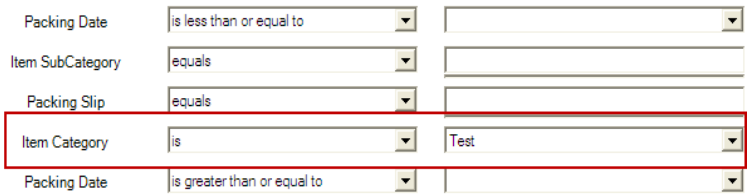


XML

```
- <Schema>
  + <SearchCriteria Value="Search for Stock Items">
  - <SearchCriteria Value="Search for Packings">
    - <ReportCondition ID="Item Category">
      - <Entity Name="Item Category" Mapsto="Item
        - <Conditions>
          + <Condition Value="=" Show="equals">
          + <Condition Value="Like" Show="Contains">
          + <Condition Value="=" Show="is">
          </Conditions>
        </Entity>
      </ReportCondition>
    + <ReportCondition ID="Item SubCategory">
    + <ReportCondition ID="Packing Slip">
    + <ReportCondition ID="Packing Date">
    + <ReportCondition ID="Packing Date">
    + <ResultSet BindTo="Packings">
    </SearchCriteria>
  + <SearchCriteria Value="Search for Sellings">
  + <SearchCriteria Value="Search for Purchases">
  </Schema>
```

Sub-Query runs and populates the options

UI

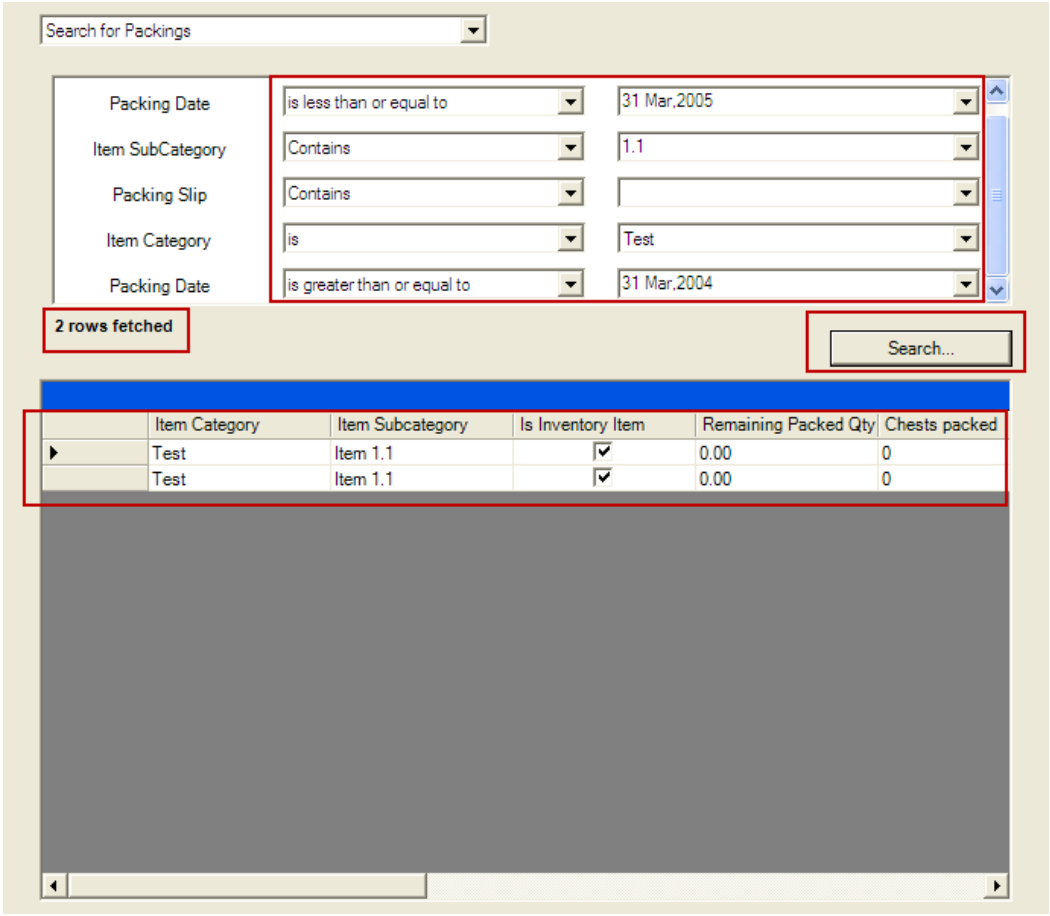


XML

```
- <Schema>
  + <SearchCriteria Value="Search for Stock Items">
  - <SearchCriteria Value="Search for Packings">
    - <ReportCondition ID="Item Category">
      - <Entity Name="Item Category" Mapsto="ItemCategories.ItemCategoryName" DataType="String">
        - <Conditions>
          + <Condition Value="=" Show="equals">
          + <Condition Value="Like" Show="Contains">
          - <Condition Value="=" Show="is">
            - <Domain Type="ListBox">
                <Query>Select distinct ItemCategoryName from ItemCategories where Len(ItemCategoryName) > 0</Query>
              </Domain>
            </Condition>
          </Conditions>
        </Entity>
      </ReportCondition>
    + <ReportCondition ID="Item SubCategory">
    + <ReportCondition ID="Packing Slip">
    + <ReportCondition ID="Packing Date">
    + <ReportCondition ID="Packing Date">
    + <ResultSet BindTo="Packings">
    </SearchCriteria>
  + <SearchCriteria Value="Search for Sellings">
  + <SearchCriteria Value="Search for Purchases">
  </Schema>
```

User selects more criteria and performs search

UI



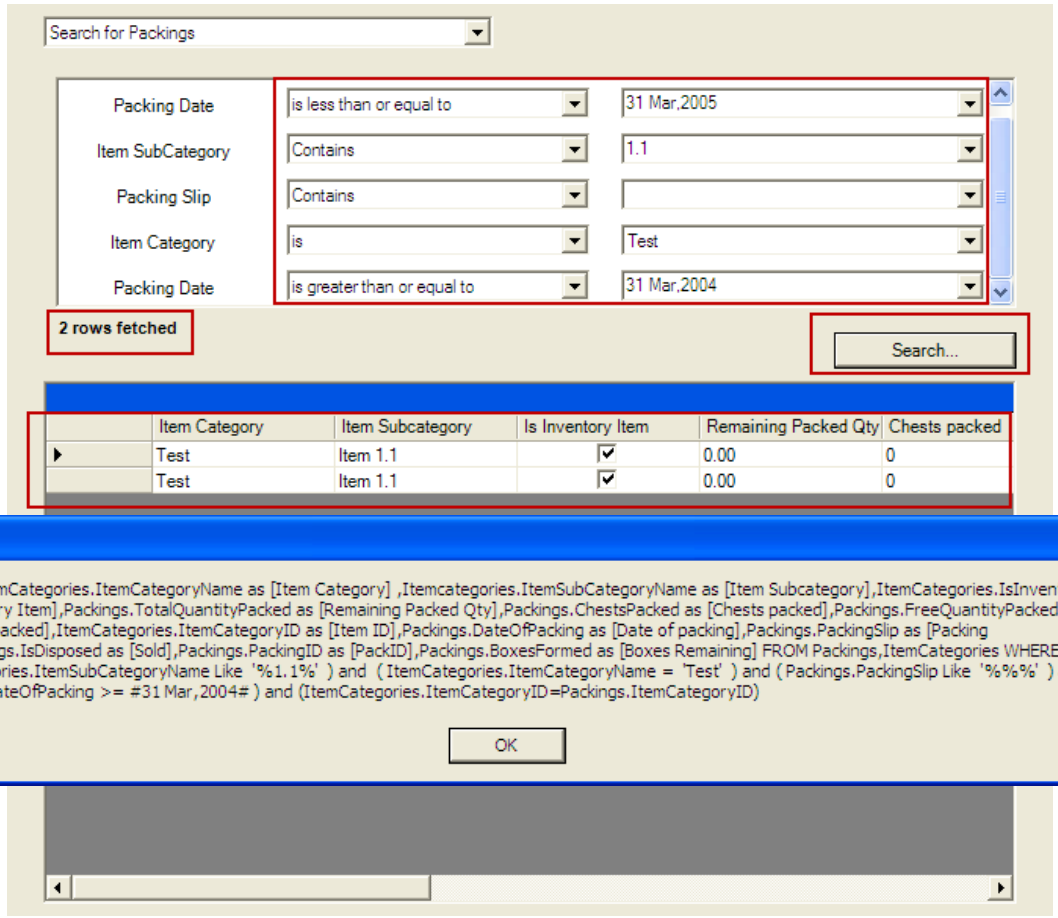
Framework replaces {placeholders} with SQL expressions, builds SQL and runs the report

XML

```
- <Conditions>
  + <Condition Value="=" Show="equals">
  + <Condition Value="Like" Show="Contains">
  - <Condition Value="=" Show="is">
    - <Domain Type="ListBox">
        <Query>Select distinct ItemCategoryName from ItemCategories where Len(ItemCategoryName) > 0</Query>
      </Domain>
    </Condition>
  </Conditions>
</Entity>
</ReportCondition>
+ <ReportCondition ID="Item SubCategory">
+ <ReportCondition ID="Packing Slip">
+ <ReportCondition ID="Packing Date">
+ <ReportCondition ID="Packing Date">
- <ResultSet BindTo="Packings">
    <MasterSet TableName="Packings">SELECT ItemCategories.ItemCategoryName as [Item Category] ,Itemcategories.ItemSubCategoryName as [Item
      Subcategory],ItemCategories.IsInventory as [Is Inventory Item],Packings.TotalQuantityPacked as [Remaining Packed Qty],Packings.ChestsPacked as
      [Chests packed],Packings.FreeQuantityPacked as [Free Qty packed],ItemCategories.ItemCategoryID as [Item ID],Packings.DateOfPacking as [Date of
      packing],Packings.PackingSlip as [Packing Slip],Packings.IsDisposed as [Sold],Packings.PackingID as [PackID],Packings.BoxesFormed as [Boxes
      Remaining] FROM Packings,ItemCategories WHERE ( {Item SubCategory} ) and ( {Item Category} ) and ( {Packing Slip} ) and ( {Packing Date} ) and
      (ItemCategories.ItemCategoryID=Packings.ItemCategoryID)</MasterSet>
  </ResultSet>
</SearchCriteria>
+ <SearchCriteria Value="Search for Sellings">
+ <SearchCriteria Value="Search for Purchases">
</Schema>
```

Built SQL can be seen in Debug mode

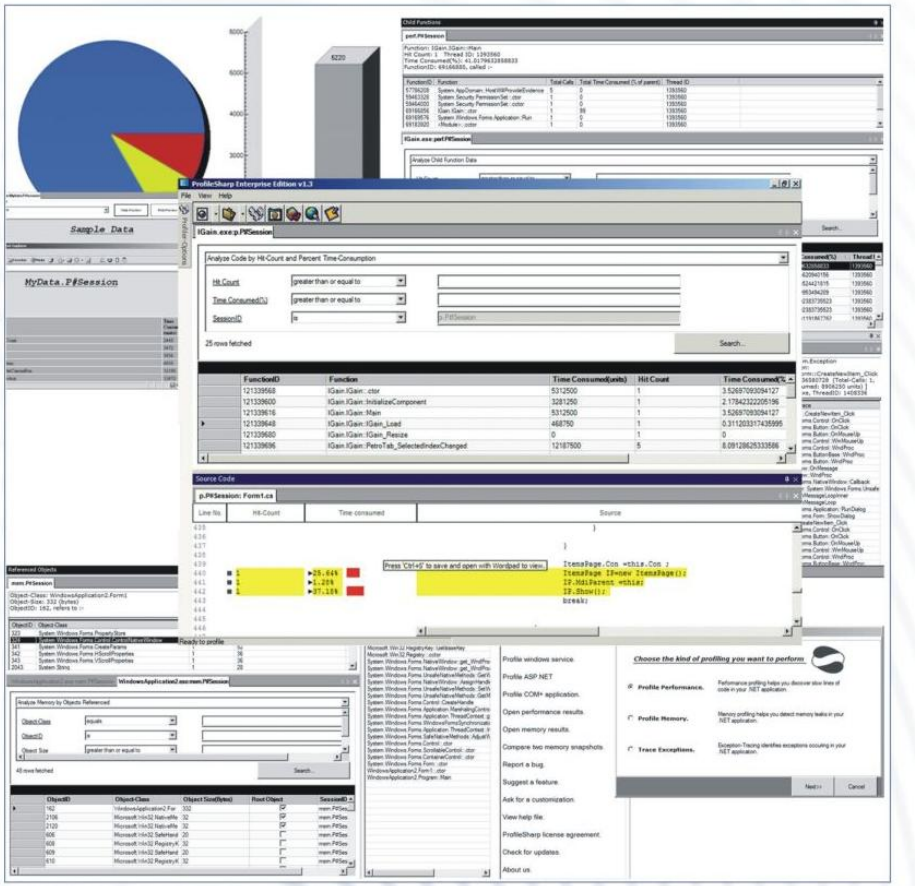
UI



Benefits

- Runs with any database and any development platform
- Can pre-fetch records (using SQL) before applying filters to the Master SQL Query [just as ***FilterCustList*** precedes **SearchCustResult**]
- Integrates well with 3rd party tools and components
- Seamlessly adheres to any table. Decreases Reports development from days to hours of work (mostly XML writing required)
- Works with minimum database round-trips. Caches data into .NET Datasets
- Can be easily customized without much coding of the framework. Mostly requires XML editing.

One of the many completely different software that also uses this framework



- Thanks!
  - We look forward to your queries.