# Microsoft Dynamics SL SDK to Acumatica Framework Reference Guide

Joe Jacob
Crestwood Associates

Version 1.0

Last updated: April 23, 2019

# Contents

# Copyright



11235 SE 6th Street,
Suite 140
Bellevue, WA 98004

**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.

# Introduction

The purpose of this guide offers a quick reference for those already familiar with the Dynamics SL SDK and are new to the Acumatica platform. This guide will help direct you to the proper concepts and API's as they relate to the expansive world of Acumatica development. It's not meant to be a replacement for training courses such as the T100, 200, and 300 series. However, it will provide a nice jump start to the training material as you move forward.

For those of you coming from the Dynamics world, you will quickly see as you learn that the Acumatica design patterns are a natural progression to the evolution of ERP technology. Additionally, and probably the most important difference between development environments, is that Dynamics SL is mostly a VB.NET language base whereas Acumatica is built on C#. If you are rusty in C#, don't be discouraged in the slightest. Immersing yourself in C# will just add fuel to your motivation.

The guide in its present form is the first iteration of effort and will be updated in future installments. Topics we are considering adding in subsequent updates include, but not limited to the following:

- A comparison of the SL menu system to Acumatica Site Map
- How reporting works in SL (Crystal Reports/FRx) vs. Acumatica
- Quick Queries –vs- Generic Inquires
- Transaction Import –vs- Import Scenario
- ERP Upgrades / standard practices for upgrading source code

We expect that there will be other topics added as well. It is our sincere hope that you find this guide of great utility and your foray into the Acumatica world of development is productive and fruitful!

## Cross Reference Topics (Concepts / API Function Calls / Properties )

<table>
<tr><th>Dynamics SL Concept</th><th>Dynamics SL</th><th>Acumatica</th></tr>
<tr><td>Core Programming Language and Tools</td><td>VB.NET using Windows Form Classes and the VB Tool Kit SDK.</td><td>C#, and ASP.NET<br>Both SL and Acumatica use various .NET frameworks depending on product release versions.</td></tr>
<tr><td>CallApplic, CallApplicWait</td><td>Runs another application from an existing screen</td><td>A graph instance is created, the graph is filled with the found record and PXRedirectRequiredException() is used to call the ASPX page.<br><br><pre>public virtual void GotoPOOrder()
{
    var poOrdEntryGraph = PXGraph.CreateInstance<POOrderEntry>();
    var currentPoPorder = VendorOrders.Current;
    poOrdEntryGraph.Document.Current = poOrdEntryGraph.Document.Search<POOrder.orderNbr>(currentPoPorder.OrderNbr);
    if (poOrdEntryGraph.Document.Current != null)
    {
        throw new PXRedirectRequiredException(poOrdEntryGraph, true, "Purchase order Details");
    }
}</pre></td></tr>
<tr><td>First Created and Last Updated Fields</td><td>Common data fields to represent created and last updated information at the record level<br><br>Fields: Crtd_DateTime, Crtd_Prog, Crtd_User, Lupd_DateTime, Lupd_Prog, Lupd_User</td><td>The following fields are used in many Acumatica tables:<br><br><table><tr><th>Field</th><th>Attribute</th></tr><tr><td>CreatedByID</td><td>[PXDBCreatedByID]</td></tr><tr><td>CreatedByScreenID</td><td>[PXDBCreatedByScreenID]</td></tr><tr><td>CreatedByDateTime</td><td>[PXDBCreatedDateTime]</td></tr><tr><td>LastModifiedByID</td><td>[PXDBLastModifiedByID]</td></tr><tr><td>LastModifiedByScreenID</td><td>[PXDBLastModifiedByScreenID]</td></tr><tr><td>LastModifiedDateTime</td><td>[PXDBLastModifiedDateTime]</td></tr><tr><td></td><td></td></tr></table></td></tr>
</table>

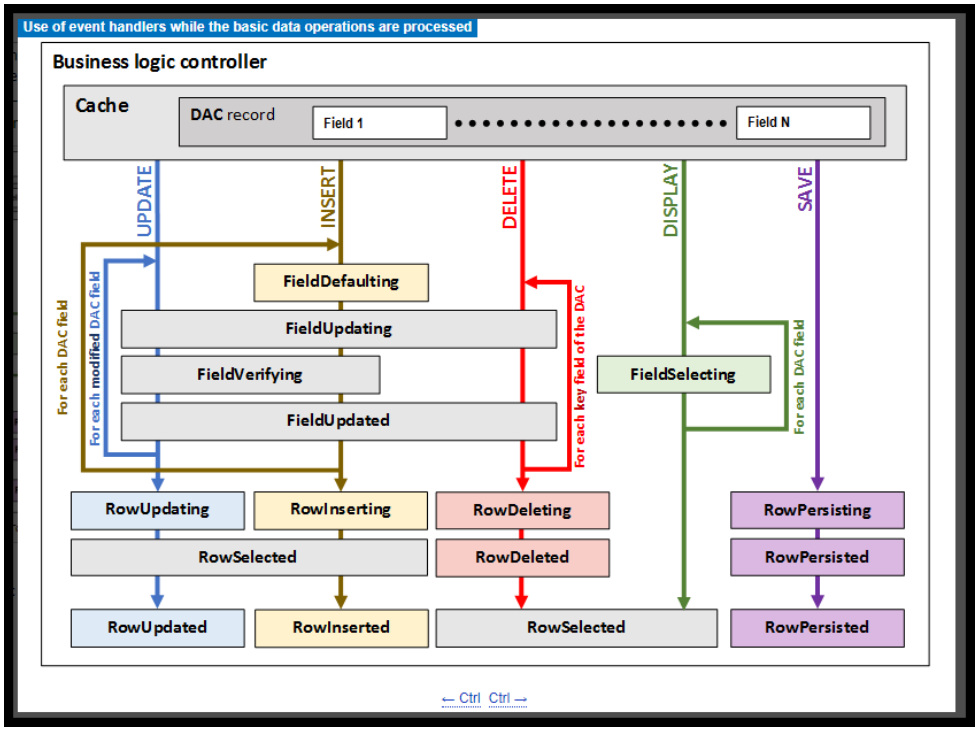
| | | |
|---|---|---|
| Typical Data Fields | Uses DataBinding attributes for standard .NET data types of String, Integer, Double, etc. | Typical Columns and Data Types |

| Value | Data Type (SQL Server) | Type Attribute on the Data Field |
|---|---|---|
| *Database identity* | *Int* | *[PXDBIdentity]* |
| *Natural key (for example, document number)* | *nvarchar (15)* | *[PXDBString(15, IsKey = true, IsUnicode = true)]* |
| *Line number* | *int* | *[PXDBInt]* |
| *Short string (for example, a name or unit of measure)* | *nvarchar (20), nvarchar (50)* | *[PXDBString(20, IsUnicode = true)]* |
| *Long string (such as a description)* | *nvarchar (255)* | *[PXDBString(255, IsUnicode = true)]* |
| *Type or status identifier (for instance, a document type)* | *int or char (1)* | *[PXDBInt] or [PXDBString(1, IsFixed = true)] respectively* |
| *Boolean flag (for example, active/inactive)* | *bit* | *[PXDBBool]* |
| *Price or cost, monetary units* | *decimal (19, 6)* | *[PXDBDecimal(6)]* |
| *Amount or total, monetary units* | *decimal (19, 4)* | *[PXDBDecimal(4)]* |
| *Quantity, pieces* | *decimal (25, 6)* | *[PXDBDecimal(6)]* |
| *Maximum, minimum, or threshold quantity, pieces* | *decimal (9, 6)* | *[PXDBDecimal(2)]* |
| *Percent, rate (for example, discount percent)* | *decimal (9, 6)* | *[PXDBDecimal(2)]* |
| *Weight or volume* | *decimal (25, 6)* | *[PXDBDecimal(6)]* |
| *Date* | *smalldatetime* | *[PXDBDate]* |
| *Time span* | *int* | *[PXDBTimeSpan(DisplayMask = "t", InputMask = "t")]* |
| *Coefficient (such as a conversion factor)* | *decimal (9, 6)* | *[PXDBDecimal(1)]* |
| | | |

<table>
<tr>
<td>Bound and Unbound Controls</td>
<td>The Solomon Data Object Class is used to define the buffer which is associated with the control object on the form.</td>
<td>DAC attributes for data types with a prefix of PXDB are bound, and a prefix <u>without</u> the DB are unbound. I.e.<br><br>Bound -> [PXDBString]<br>Unbound -> [PXString]<br><br>
Unbound Field Data Types<br>
The following attributes define a data access class field of a specific type that are not bound to any database columns.
<table>
<tr><th>Attribute</th><th>C# data type</th><th>Comment</th></tr>
<tr><td>PXBool</td><td>bool?</td><td>Boolean value</td></tr>
<tr><td>PXByte</td><td>byte?</td><td>1-byte integer value</td></tr>
<tr><td>PXDate</td><td>DateTime?</td><td>Date and time</td></tr>
<tr><td>PXDateAndTime</td><td>DateTime?</td><td>Date and time values represented by separate input controls in the user interface</td></tr>
<tr><td>PXDecimal</td><td>Decimal?</td><td>16-byte floating point numeric value with a specific precision</td></tr>
<tr><td>PXDouble</td><td>double?</td><td>8-byte floating point value</td></tr>
<tr><td>PXFloat</td><td>float?</td><td>4-byte floating point value</td></tr>
<tr><td>PXGuid</td><td>Guid?</td><td>16-byte unique value</td></tr>
<tr><td>PXShort</td><td>short?</td><td>2-byte integer value</td></tr>
<tr><td>PXInt</td><td>int?</td><td>4-byte integer value</td></tr>
<tr><td>PXLong</td><td>int64?</td><td>8-byte integer value</td></tr>
<tr><td>PXString</td><td>string</td><td>String of characters</td></tr>
<tr><td>PXTimeSpan</td><td>int?</td><td>Date and time value represented by minutes passed from 01/01/1900</td></tr>
<tr><td>PXTimeSpanLong</td><td>int?</td><td>Duration in time as the number of minutes</td></tr>
<tr><td>PXVariant</td><td>byte[]</td><td>Arbitrary array of bytes</td></tr>
</table>
<br>Reference: Unbound Field Types</td>
</tr>
</table>

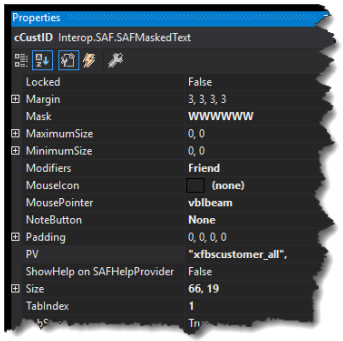
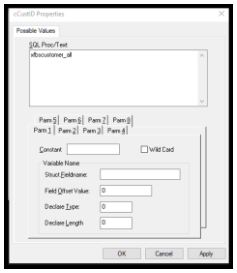
<table>
<tr>
<td>Screen Grid Control</td>
<td>Windows form control using Interop.SAF.SAFGrid with key properties of DBNav, Level, etc.</td>
<td>ASPX tag of <px:PXGrid> is used with references to the graph 'Views'<br>MasterPageFile templates are used to build standard forms.<br><br>
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>FormDetail</td><td>The master-detail editing page with FormView and Grid controls</td></tr>
<tr><td>FormTab</td><td>The record-editing page with FormView and Tab controls</td></tr>
<tr><td>FormView</td><td>The record-editing page with one FormView control</td></tr>
<tr><td>ListView</td><td>The record-editing page with one Grid control</td></tr>
<tr><td>TabDetail</td><td>The master-detail page with Tab and Grid controls</td></tr>
<tr><td>TabView</td><td>The record-editing page with one Tab control</td></tr>
</table>
</td>
</tr>
<tr>
<td>Key reference DLL files</td>
<td>Interop.SAF*<br>Microsoft.Dynamics.*<br>Solomon.Kernel</td>
<td>PX.Objects<br>PX.Data<br>PX.Common</td>
</tr>
<tr>
<td>IS_TI()</td>
<td>Returns a flag indicating whether or not the application is being automated by Transaction Import</td>
<td>[PXGraph].IsImport is a Boolean flag that will tell your solution that it is being referenced by Import Scenario.<br><br>Other useful flags would include IsExport, IsMobile, IsContractBasedAPI and ExternalCall.<br><br>Reference: Import Scenario, which is a close replacement to Transaction Import.<br>http://blog.zaletskyy.com/Tags/IsImport</td>
</tr>
<tr>
<td>MessBox()</td>
<td>Displays and message and waits for user to choose a button.</td>
<td>[PXGraph].Ask()<br><br>// Asking for confirmation on an attempt to delete<br>if (ShipmentLines.Ask("Confirm Delete","Are you sure?",MessageButtons.YesNo) != WebDialogResult.Yes)<br>{<br>  e.Cancel = true;<br>}</td>
</tr>
</table>

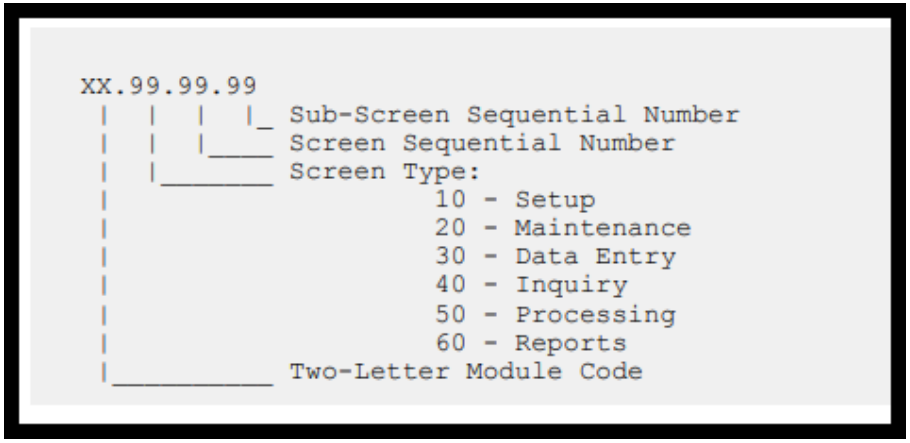
<table>
<tr>
<td>MFirst(),<br>MNext(),<br>MLast()</td>
<td><u>Memory Arrays</u><br><br>Move to the first, next, or last record in a designated memory array. Usually associated with SAFGRID control.</td>
<td>A DAC can be associated with a grid and a standard foreach can be used to iterate through the ‘view’<br><br>foreach (ShipmentLine line in ShipmentLines.Select())<br>{ … }<br><br>Reference: PXView Class, BQL</td>
</tr>
<tr>
<td>NoteColumn Properly,<br>NoteButton Properly</td>
<td>Tables with a NoteID field are used to manage text notes in SL.<br><br>Note data is stored in the SNote table.</td>
<td>Acumatica supports storing notes and attaching additional objects to data records. You can attach additional objects to a data record—for instance, add a textual note or upload a file or multiple files to a data record. You enable support for data record attachments for each particular table individually. To enable support for data record attachments, add the column that stores the global data record identifier (typically, NoteID) to the table and declare the corresponding field in the data access class.<br><br>
<table>
<tr><th>SQL Datatype</th><th>Attribute</th></tr>
<tr><td>BIGINT , null</td><td>[PXNote]</td></tr>
</table>
<br>Also worth noting is the ‘DeletedDatabaseRecord’ column with is a low-level mechanism for preserving deleted data records in the database.</td>
</tr>
</table>

| | | |
|---|---|---|
| Form Control Events Handlers | Various event handlers are used in SL such as Chk, LineChk, Default, Update, Finish, etc. | Sample Event Handler declaration<br><br>*protected virtual void DACName_FieldName_FieldDefaulting( PXCache sender, PXFieldDefaultingEventArgs e)*<br>*{*<br>*...*<br>*}*<br><br>  |

| Data Field Events | Data Record Events | Database-Related Events | Exception-Handling Event | Event for Overriding DAC Field Attributes |
|---|---|---|---|---|
| PXFieldDefaulting<br>PXFieldVerifying<br>PXFieldUpdating<br>PXFieldUpdated<br>PXFieldSelecting | PXRowSelected<br>PXRowInserting<br>PXRowInserted<br>PXRowUpdating<br>PXRowUpdated<br>PXRowDeleting<br>PXRowDeleted | PXCommandPreparing<br>PXRowSelecting<br>PXRowPersisting<br>PXRowPersisted | PXExceptionHandling | CacheAttached |
| | | | | |

Reference: [Wiki Article]

| | | |
|---|---|---|
| PV Property, PVChkFetch() | <u>Possible Value Look Ups</u><br><br>Retrieves a composite record from the database using an SQL statement from the PV property of an SAFMaskedText control.<br><br>  <br><br><br>  | <u>DAC attribute of [PXSelector]</u><br>The PV concept in Acumatica is very extensive and flexible. A DAC attribute of [PXSelector] is used and can be established at the DAC level and overridden at the Graph level.<br><br>`[PXDefault]`<br>`[PXUIField(DisplayName = "Customer ID")]`<br>`[PXSelector(`<br>`typeof(Customer.customerID),`<br>`typeof(Customer.customerCD),`<br>`typeof(Customer.companyName),`<br>`SubstituteKey = typeof(Customer.customerCD))]`<br>`public virtual int? CustomerID`<br>`{...`<br>`}`<br><br>*#region TermsID*<br>*public abstract class termsID : PX.Data.IBqlField*<br>*{*<br>*}*<br>*protected String _TermsID;*<br>*[PXDBString(10, IsUnicode = true)]*<br>*[PXSelector(typeof(Search<Terms.termsID, Where<Terms.visibleTo, Equal<TermsVisibleTo.vendor>, Or<Terms.visibleTo, Equal<TermsVisibleTo.all>>>>), DescriptionField = typeof(Terms.descr), CacheGlobal = true)]*<br>…<br><br>Reference: [PXSelectorAttribute] |

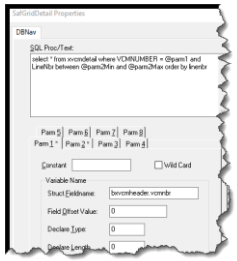
| | | |
|---|---|---|
| Screen Numbering | SL uses the (XX.999.99) format for screen titles. | Screen numbering in Acumatica is similar:<br>  <br> |
| SDK Platforms | a. New custom screens are developed using the SL VB Tool Kit SDK which produce compiled EXE's.<br><br>b. Modifications to existing screens can be done using a specialize version of VBA with its contents stored within the SL system database | Solutions are packaged into individual customization projects controlled by the Acumatica ERP. Here are a list of development components that can be stored:<br>  <br><br>A typical solution with custom screens might include one or more DLL's and ASPX files under the 'Files' section, Site Map entries, Wikis, and reports. |

| | | |
|---|---|---|
| SFetch() | Used to retrieve a composite record from the database based on some pre-defined SQL statement or stored procedure. | A deep dive into PXSelectBase class is vital. The PXSelect class is a multi-purpose class and can be used for ad-hoc SQL queries.<br><br>This class and other classes derived from PXSelectBase are used as a basis for building BQL statements. BQL is then translated into the SQL statements.<br><br>Sample<br><br>*public PXSelect<APPayment,*<br>*Where<APPayment.vendorID, Equal<Required<APPayment.vendorID>>,*<br>*And<APPayment.docType, Equal<Required<APPayment.docType>>,*<br>*And<APPayment.refNbr, Equal<Required<APPayment.refNbr>>>>>>*<br>*APPayment_VendorID_DocType_RefNbr;*<br><br>*// Retrieving data records through the data view*<br>*// The parameter values are taken from the adj APAdjust object*<br>*foreach (APPayment payment in APPayment_VendorID_DocType_RefNbr.Select(adj.VendorID, adj.AdjdDocType,adj.AdjdRefNbr))*<br>*{*<br>*//do something with each APPayment object*<br>*}*<br><br>The Search class can also be used within DAC attributes such as PXSelector, PXDbScalar and PXDefault for returning one field from the database including cached values. |

| Type of Search | Description |
|---|---|
| Search<Field> | Gets field value |
| Search<Field, Where> | Gets field value with filtering by Where condition |
| Search<Field, Where, OrderBy> | Gets field value with filtering by Where condition and ordering |
| Search2<Field, Join> | Gets field value with filtering by using Joins with other tables |
| Search2<Field, Join, Where> | Gets field value with filtering by using Joins with other tables and applying where condition |
| Search2<Field, Join, Where, OrderBy> | Gets field value with filtering by using Joins with other tables and applying where condition and ordering |
| Search3<Field, OrderBy> | Gets field with ordering application |
| Search3<Field, Join, OrderBy> | Gets field value with joins and order by application |
| Search4<Field, Aggregate> | Gets aggregated field value |
| Search4<Field, Where, Aggregate> | Gets aggregated field value with filtering by where condition |
| Search4<Field, Where, Aggregate, OrderBy> | Gets field value with filtering by where, aggregation and order by |
| Search5<Field, Join, Aggregate> | Gets field value with application of joins and aggregates |
| Search5<Field, Join, Where, Aggregate> | Gets field value with application of joins and where and aggregate |
| Search5<Field, Join, Where, Aggregate> | Gets field value based on join, where and aggregate condition |
| Search6<Field, Aggregate, OrderBy> | Gets field value based Aggregate and order by |
| Search6<Field, Join, Aggregate, OrderBy> | Gets field value based on join, aggregate and order by |
| Coalesce<Search1, Search2> | Gets field value with using Search1 or if Search1 gives null uses Search2 |

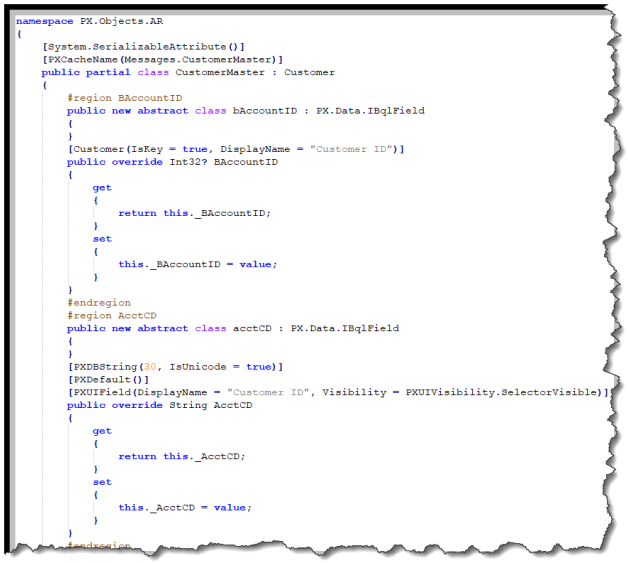
| | | |
|---|---|---|
| | | Example within a DAC attribute:<br>*[PXDefault(typeof(Search<PriceList.price, Where<PriceList.grade, Equal<Current<ParkingLot.grade>>>>))]*<br><br>Example within a method<br>*Document.Search<POOrder.orderNbr>(currentPoOrder.OrderNbr, currentPoOrder.OrderType);*<br><br>See the section on **SQL-BQL** |
| SGroupFetch() | Used to retrieve a composite record from the database based on some pre-defined SQL statement or stored procedure containing one or more group aggregate functions and/or clauses. | The PXSelectGroupBy can be used to select records from one table grouping and applying aggregations.<br><br>*foreach (PXResult<CWBleachRecL> item in PXSelectGroupBy<CWBleachRecL,*<br>*Where<CWBleachRecL.bleachRecID, Equal<Required<CWBleachRecL.bleachRecID>>>,*<br>*Aggregate<GroupBy<CWBleachRecL.inventoryID, GroupBy<CWBleachRecL.ordNbr,*<br>*Count<CWBleachRecL.lineNbr>>>>>*<br>*.Select(this, reconID))*<br>*{*<br>*string ord = ((CWBleachRecL)item).OrdNbr;*<br>*... do something*<br>*}*<br><br>See the section on **SQL-BQL** |

| | | |
|---|---|---|
| SInsert(), SUpdate() | Inserts or updates one record into each specified table within an existing database view. | When forcing a save on a PXGraph a PXAction is declared and the Actions.PressSave() function can be called.<br><br>*public PXAction<Shipment> CancelShipment;*<br>*[PXButton(CommitChanges = true)]*<br>*[PXUIField(DisplayName = "Cancel Shipment")]*<br>*protected virtual void cancelShipment()*<br>*{*<br>*  Shipment row = Shipments.Current;*<br>*  row.Status = ShipmentStatus.Cancelled;*<br>*  // Update the data record in the cache of Shipment data records*<br>*  Shipments.Update(row);*<br>*  // Triggering the Save action to save changes in the database*<br>*  Actions.PressSave();*<br>*}*<br><br>Another option is the use of PXDatabase.Update<br><br>*PXDatabase.Update<SOOrder>(*<br>*  new  PXDataFieldAssign<SOOrderExt.usrRetDelStatus>(PXDbType.VarChar, retDelFlag),*<br>*  PXDataFieldRestrict<SOOrder.orderNbr>(PXDbType.NVarChar,  orderNumber),*<br>*  new PXDataFieldRestrict<SOOrder.orderType>(PXDbType.Char,  Messages.DefaultOrderType)*<br>*);* |
| | | |

| | | |
|---|---|---|
| DBNAV() | Used to facilitate navigation through all database records in the result set of an SQL statement.<br><br>Typically used in a GRID definition.<br><br>  | Data views are graph members that are used to retrieve and modify data records of a particular data access class (DAC). You use data views:<br><br>• To provide data retrieval and manipulation functions for the UI<br>• To retrieve and manipulate data from code<br><br>*public class SalesOrderEntry : PXGraph<SalesOrderEntry, SalesOrder>*<br>*{*<br>*// Provides an interface for manipulation of sales orders*<br>*public PXSelect<SalesOrder> Orders;*<br><br>*// Provides an interface for manipulation of detail lines of the specified order*<br>*public PXSelect<OrderLine, Where<OrderLine.orderNbr, Equal<Current<SalesOrder.orderNbr>>>> OrderDetails;*<br>*…*<br>*}*<br><br>See the section on **SQL-BQL** |
| Status() | Report process status information to either the Process Status Window or the Event Log or both. | The PXLongOperation is a static class that is used to execute a long-running operation, such as processing data or releasing a document, asynchronously in a separate thread. This class manages the threads created on the Acumatica ERP server to process long-running operations.<br><br>This class also animates a spinning wheel icon and timer showing the user 'status' of a process.<br><br>*PXLongOperation.StartOperation(this, delegate()*<br>*{*<br>*ReleaseDocs(list);*<br>*});*<br><br>Typically placed above the PXTransactionScope.<br>Reference: PXLongOperation Class |

| | | |
|---|---|---|
| TranBeg(),<br>TranEnd(),<br>TranAbort() | Begin, end, commits for database transaction. | The PXTransactionScope is used to initialize a new transaction. You can wrap-up one or more changes into a transaction and the system will revert all changes together on any exception. Do not forget to call commit for the transaction scope before dispose it.<br><br>*using (PXTransactionScope ts = new PXTransactionScope())*<br>*{*<br>*string[] desc = ((ARTran)tran).TranDesc.Split(':');*<br>*((ARTran)tran).TranDesc = desc[1];*<br>*Base.Transactions.Cache.IsDirty = true;*<br>*Base.Transactions.Update((ARTran)tran);*<br>*Base.Actions.PressSave();*<br>*Base.Persist();*<br>*ts.Complete();*<br>*}*<br><br>Also see: **PXLongOperation** |
| | | |

| | | |
|---|---|---|
| sql() | Initialize a new database view. Takes the specified SQL text, compiles it, and then runs it.<br><br>Used typically to run a stored procedure. | PXDatabase.Execute() is used to execute a stored procedure<br><br>*var pars = new List<PXSPParameter>();*<br>*PXSPParameter p1 = new PXSPInParameter("@prPayrollRefNbr", PXDbType.NChar, details.PayrollRefNbr);*<br>*PXSPParameter p2 = new PXSPInParameter("@prPayrollDetailsID", PXDbType.NChar, details.PRPayrollDetailID);*<br>*pars.Add(p1);*<br>*pars.Add(p2);*<br>*PXDatabase.Execute("DeletePrTranByPrPayrollAndPayrollDetailsId", pars.ToArray());*<br><br>See the section on **SQL-BQL** |
| SQL – BQL | SL uses standard SQL statements | BQL stands from Business Query Language and is used throughout the Acumatica framework<br><br>The code in **bold** below is a sample of a BQL statement including where, and join clauses. See section 3.1 in the T200 Acumatica framework fundamentals document.<br><br>*public PXSelectJoin<**SupplierProduct, LeftJoin<Product, On<Product.productID,***<br>***Equal<SupplierProduct.productID>>>,***<br>***Where<SupplierProduct.supplierID,***<br>***Equal<Current<Supplier.supplierID>>>>***<br>*SupplierProducts;* |
| TimeStamp usage | In SL the field tstamp<br>(timestamp, not null)<br>is used to control record locking. | The SQL Field tstamp<br>(timestamp, not null )<br>is also used in Acumatica but with a [PXDBTimestamp] attribute in the DAC |

| | | |
|---|---|---|
| Data Access Layer | The Solomon Data Object Class from the SDK is used to represent each SQL table. Commonly referred to as buffers. | The Data Access Class know as DAC, uses an IBqlTable interphase to manage each field within a Table.<br>It's important to gain an understanding of how class attributes are used and how powerful they are for controlling how various fields behave.<br><br>PXDefault, PXUIField, PXSelector, and all the possible PX data fields are typical for each field within the class.<br>See section on **Typical Data Fields**<br><br>Sample DAC section:<br><br>*[PXDBInt(IsKey = true)]*<br>*[PXDefault]*<br>*[PXUIField(DisplayName = "Product ID")]*<br>*[PXSelector(typeof(Search<Product.productID>),*<br>*typeof(Product.productCD),*<br>*typeof(Product.productName),*<br>*typeof(Product.unitPrice),*<br>*SubstituteKey = typeof(Product.productCD))]*<br>*public virtual int? ProductID*<br>*…*<br><br>  |
| | | |

```
namespace PX.Objects.AR
{
    [System.SerializableAttribute()]
    [PXCacheName(Messages.CustomerMaster)]
    public partial class CustomerMaster : Customer
    {
        #region BAccountID
        public new abstract class bAccountID : PX.Data.IBqlField
        {
        }
        [Customer(IsKey = true, DisplayName = "Customer ID")]
        public override Int32? BAccountID
        {
            get
            {
                return this._BAccountID;
            }
            set
            {
                this._BAccountID = value;
            }
        }
        #endregion
        #region AcctCD
        public new abstract class acctCD : PX.Data.IBqlField
        {
        }
        [PXDBString(30, IsUnicode = true)]
        [PXDefault()]
        [PXUIField(DisplayName = "Customer ID", Visibility = PXUIVisibility.SelectorVisible)]
        public override String AcctCD
        {
            get
            {
                return this._AcctCD;
            }
            set
            {
                this._AcctCD = value;
            }
        }
        #endregion
```