Oracle® Agile Product Lifecycle Management for Process

Product Quality Management Extensibility Guide

Extensibility Pack 3.0

E39502-01

March 2013

ORACLE®

Copyrights and Trademarks

Agile Product Lifecycle Management for Process, Extensibility Pack 3.0

E39502-01

Contents

Overview

The Product Quality Management (PQM) application is a full featured, fully integrated module for Enterprise Quality Management. It is designed to tightly integrate Issues, Actions, and Audits, with the rest of the Agile PLM for Process application suite, including GSM, SCRM, and NPD, but is flexible enough to allow for integrations with external systems from within the user interface. Additionally, a rich set of PQM web services allows for most of the core PQM functionality to be managed from other systems, if desired, thus providing many options for deployment and product rollout. Furthermore, PQM provides many useful extension points found throughout the application suite, such as Validation, Notifications, Workflow Actions and Guard Conditions, customized emails, and more.

This document discusses the following extensibility points available for the PQM application:

- Custom Read and Write Permissions
- Workflow Actions and Guard Conditions
- Workflow Triggered Email Notifications
- Related Items Display
- Custom Data Calculation
- Validation
- Notification Panel
- Event Model
- Supplier Extensibility
- Affected Items Extensibility
- Related Project Extensibility
- Web Services
- Supplier PQM

Many of the implementation details of the different extensibility points can be found in the Extensibility Pack.

Additionally, there are several useful utility classes available for PQM custom code development, as described in the Appendix

Custom Read and Write Permissions

Basic PQM read, write, and workflow permissions for Issues, Actions, and Audits are based on the workflow templates set up in Workflow Administration. PQM adds two useful extensibility points to further customize Read and Write permissions on a PQM Item:

Custom Read Permission

A Validate Plugin class can be created to extend the Read permission logic of a PQM Item, if desired.

To customize the Read permission checks for PQM, create a new Validate Plugin and add an entry into the CustomPluginExtensions.xml file in config\Extensions, in the ValidatePlugins node, using the plugin name "**HasPQMReadPermissionPlugin**", like so:

```
<Plugin name="HasPQMReadPermissionPlugin"
ignoreInheritFromPluginName="true"
FactoryURL="{Your custom class using ObjectLoaderURL syntax}" />
```

Custom Write Permission

A Validate Plugin class can be created to extend the Write permission logic of a PQM Item, if desired.

To customize the Write permission checks for PQM, create a new Validate Plugin and add an entry into the CustomPluginExtensions.xml file in config\Extensions, in the ValidatePlugins node, using the plugin name "**HasPQMWritePermissionPlugin**", like so:

```
<Plugin name="HasPQMWritePermissionPlugin"
ignoreInheritFromPluginName="true"
FactoryURL="{Your custom class using ObjectLoaderURL syntax}" />
```

TECHNICAL NOTE

The Validate Plugin class gets passed the current PQM data object (the PQM Issue, Action, or Audit) as a Xeno.Data.PQM.IPQMItemBase interface, which can be cast as an IPQMActionDO, IPQMAuditDO , or IPQMIssueDO. The current User is also available to the plugin, via the Xeno.Prodika.PluginExtensions.Context.ValidatePluginContext User property.

To learn more about Validate Plugins, see the PluginExtensions document in the \ReferenceImplementations\PluginExtensions\Documentation folder. Reference implementations of Validate Plugins can be found in the \ReferenceImplementations\PluginExtensions\SourceCode\ReferencePlugins\ValidatePlugins folder.

Workflow actions & guard conditions

Workflow Actions are extension points that trigger the execution of custom classes when a workflow transition occurs. Guard conditions are extensibility points that help determine if a workflow transition can occur. Workflow actions and workflow guard conditions are specified in a configuration file, and are then assignable to workflow transitions in the WFA user interface.

Custom workflow actions and guard conditions can be created for PQM Actions, Audits, and Issues. When configuring the Workflow Actions and Guard Conditions in CustomWFAExtensionsConfig, use the:

```
processTemplateTypes="PQM"
```

attribute value. Refer to the Workflow Actions and Guard Conditions documentation in the Extensibility Pack for details on writing Workflow Actions and Guard Conditions.

Existing PQM Workflow Actions

Two workflow actions are available for use for PQM Actions:

1. **PQM Action - Release Related Issues**: For a PQM Action, this marks any related Issues as Released unless the issue is already in a cancelled state. This assumes that statuses in WFA are assigned the relevant PQM workflow tags.
2. **PQM Action - Cancel Related Issues**: For a PQM Action, this marks any related Issues as Cancelled. This assumes that statuses in WFA are assigned the relevant PQM workflow tags.

These workflow actions are made available by un-commenting them from the CustomWFAExtensionsConfig.xml in config\Extensions.

TECHNICAL NOTE

Workflow Action classes are passed an IPQMLinearTransitionContext (Xeno.Prodika.Services.PQM.Workflow) object which contains the current PQM item business object (IPQMItemBaseBO) and the PQM service IPQMItemService.

To learn more about Workflow Actions and Guard Conditions, see the Workflow Actions and Guard Conditions document in the \ReferenceImplementations\WorkflowActions\Documentation folder. Reference implementations of Workflow Actions and Guard Conditions can be found in the \ReferenceImplementations\ WorkflowActions\SourceCode\ReferenceWorkflows folder.

Workflow Email Notifications

PLM for Process provides various automated email notifications when PQM Actions, Issues, and Audits, move from one workflow status to another. Emails can be sent to owners of the PQM items informing them that the item is now in their action items listing, or that they need to sign off on the item, or as a simple notification that the item moved from one status to another. The email recipients are specified in the WFA application, using the Owners, Notifications, and Signature Request grids.

Clients wishing to customize the existing emails may do so by modifying the translation templates used for the PQM workflows, or by writing custom classes for more fine-grained control.

See the PLM for Process Email Extensibility document for more details.

Related Item Display

The various PQM Related Items listings show basic information on the related PQM items. The information shown can be customized to include additional display data. Clients can create custom Format Plugins classes for the Related Issues, Related Actions, and Related Audits listings by using the following Format Plugin extension points:

PQMRelatedIssues
PQMRelatedActions
PQMRelatedAudits

When configuring any of the above FormatPlugins, be sure to include the following XML attribute:

```
ignoreInheritFromPluginName="true"
```

See the PluginExtensions documentation, available in the Extensibility Pack, for details on writing FormatPlugins.

Extended Attribute Calculation

Calculated Extended Attributes (Calculated Numeric/Boolean/Text) on PQM items allow you to create a read-only extended attribute (EA) that displays results of a calculation to the user. The calculation (written in JScript), specified in the Data Admin user interface for Extended Attributes, can access data from other extended attributes, custom sections, and other data from the PQM item the EA is attached to. Additionally, the script can execute a call to a custom class to return additional data to the script.

Clients wishing to have more control over calculations, consolidate their calculation logic, or access other data not directly available through JScript (and the predefined functions), may call out to custom classes from their scripts. The custom classes get executed and return a result back to the script. They may optionally receive parameter data from the script. Custom calculation classes have access to the PQM Item business object (IPQMItemBaseBO) using the Entity property, which provides full access to the PQM item.

PQM Calculation Veto Plugin

Clients may limit the times when PQM Custom Section calculation should occur by creating a ValidatePlugin class using the extension name “IsPQMCalculationAllowed”. This plugin can be used to determine if calculation should be turned on or off. This can be combined with the Format plugin “IsPQMCalculationAllowedOverrideMessage” to provide a customized message to indicate if the spec calculation was enabled or disabled.

Validation

The validation framework allows you to configure custom validation rules to specific UI events in the system. For example, when a user selects the Save button on a PQM Action, code can be put in place to make sure specific required fields are properly filled out. If any required fields are left blank, an error message can be displayed preventing the user from saving the Action until all of the data is provided.

Custom validation rules can be written that validate against PQM Actions (type=7003), Issues (type=7002), and Audits (type=7004) for Save, Copy, and Workflow events. PQM item Templates are also available for validation (7002T, 7003T, & 7004T) for the same events.

See the Validation Training documentation in the Extensibility Pack for details.

Notification Panel

Notification Panels are available on the PQM user interface. To create custom notifications that can show up in PQM, clients can create Notification Plugins and configure them using the `usedIn="PQMItem"` XML attribute.

See the PluginExtensions documentation, available in the Extensibility Pack, for details on writing NotificationPlugins.

Event Model

As specific events occur in PQM, their details are captured and recorded in the `pqmLifecycleEventLog` database table. Clients can watch for events added to this table to trigger some custom actions.

Each event captured may include the following information:

- Event Type—The type of event that occurred (1: Create, 2: Save, 3: Workflow, 4: Copy)
- Event Source— What caused the event (a PQM Edit, a web service, etc.)
- Actor—User who performed the event
- Timestamp —Date and time stamp of when the event happened
- Affected Object—PQM business object that was acted upon (object that was saved/copied, etc.)
- Related Object—Related object when appropriate (Workflow step)
- Reason—Reason the action occurred when appropriate (ex: Workflow comments)

Suppliers Extensibility

The source of PQM Supplier data (Company and Facility entries in the Action, Issue, and Audit Supplier gird) can be customized to pull data from the current (local) SCRM instance, some other instance of PLM for Process SCRM, and/or from entirely different applications.

Supplier Source Data

The out of the box configuration assumes the Agile PLM for Process SCRM application is local and on a shared database. Using an alternate instance of SCRM, in which PQM and SCRM reside on *different* databases, can be done by some simple configuration changes. Pulling in Supplier data from other systems requires that a custom ASCX control be created and deployed into PQM, as well as a class which takes the PQM supplier data and generates the formatted view for display in the UI. Multiple configurations of source data are permitted, which allow PQM Supplier data to be pulled from multiple applications.

Additionally, the configuration for PQM Suppliers is flexible enough to allow different configurations for Actions, Audits, and Issues, if needed.

Local SCRM

Assumes the PQM Supplier data is populated from SCRM residing on the same database. This is the default configuration for PQM Suppliers, so no configuration changes are required to the out-of-the box installation.

External SCRM

Agile PLM for Process SCRM that is deployed on a *separate* database from PQM requires configuration changes that specify the server URL and database connection. Additionally, this configuration requires that the **Reporting DB connection pool** is configured (on the PQM application's configuration files) to point to the database connection where SCRM is hosted.

See the Configuration > SCRM Residing on a Different Database section in this document.

Trusted Site

When searching for SCRM Companies and Facilities on a different server from PQM, users must update their Internet Explorer browser settings to add that SCRM web site as a Trusted Site. Failing to do so will result in the Add Suppliers popup and the SCRM EQT Popup being unable to post the results back to PQM.

SCRM EQT Feature Configuration

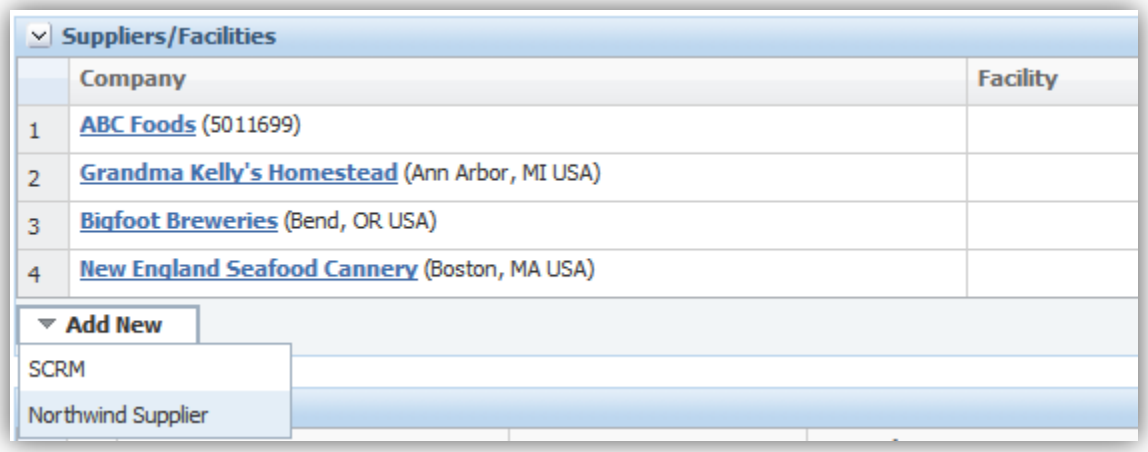
When using PQM to search for Actions, Issues, or Audits that have a specific SCRM Companies or Facilities, different configurations control whether or not the search options should include a local SCRM search and/or a remote SCRM search. The following FeatureConfiguration entries enable the behaviors:

```
<add key="PQM.EQT.Action.SupplierSCRMSearch.Local.Enabled"        value="true"
configDescription="Sets PQM Action EQT Search of Suppliers by local SCRM DB"/>
```

```
<add key="PQM.EQT.Action.SupplierSCRMSearch.External.Enabled"     value="false"
configDescription="Sets PQM Action EQT Search of Suppliers by External SCRM DB"/>

<add key="PQM.EQT.Audit.SupplierSCRMSearch.Local.Enabled"         value="true"
configDescription="Sets PQM Audit EQT Search of Suppliers by local SCRM DB"/>
<add key="PQM.EQT.Audit.SupplierSCRMSearch.External.Enabled"      value="false"
configDescription="Sets PQM Audit EQT Search of Suppliers by External SCRM DB"/>

<add key="PQM.EQT.Issue.SupplierSCRMSearch.Local.Enabled"         value="true"
configDescription="Sets PQM Issue EQT Search of Suppliers by local SCRM DB"/>
<add key="PQM.EQT.Issue.SupplierSCRMSearch.External.Enabled"      value="false"
configDescription="Sets PQM Issue EQT Search of Suppliers by External SCRM DB"/>
```

Alternate Supplier Source Systems

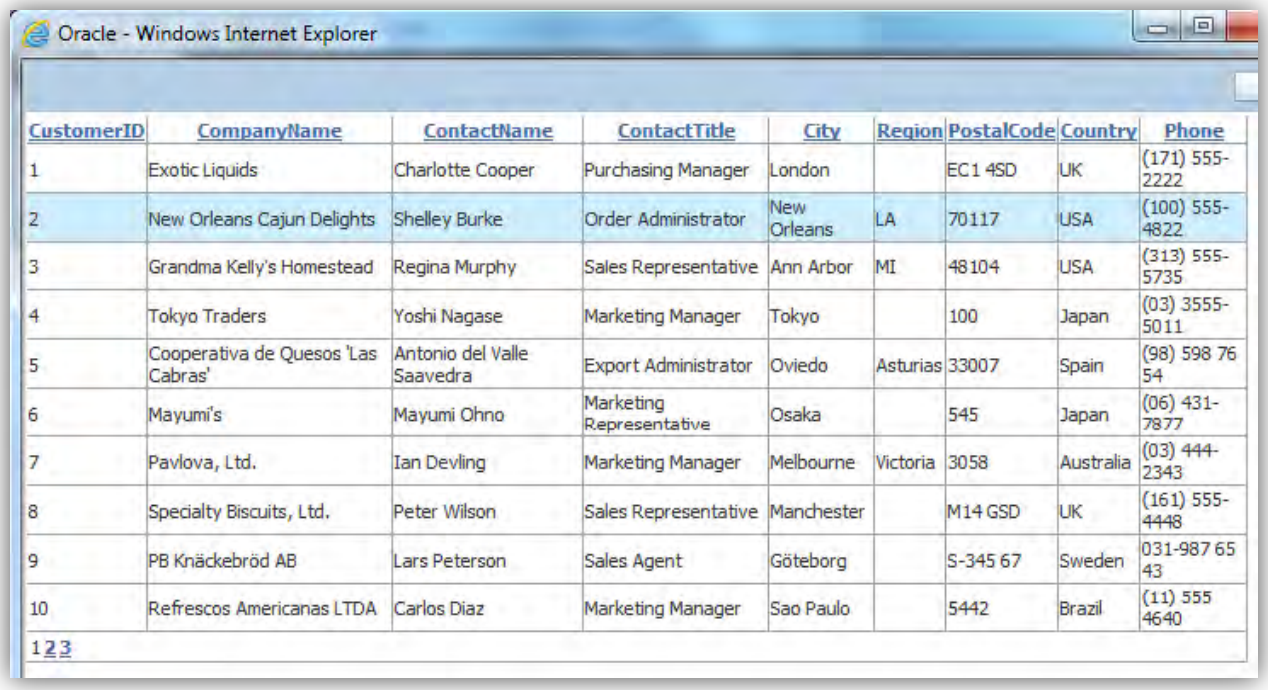
When pulling Supplier data from alternate systems, a custom ASCX control must be created and plugged into PQM that allows for searching and selecting a company or facility. This company or facility is then added to the Suppliers listing in PQM via the hosting aspx page's AddItem Javascript function.



The configured list of possible suppliers is added to the Add New button. Clicking on the external supplier system launches the custom control.

ASCX control

The custom control's responsibility it to provide a mechanism for users to search for supplier data (ex company or facility data), and then select the entry that will be populated in the Suppliers listing. The control could, for example, allow users to enter some search criteria, and pass those values to a web service call to another system, returning any matching suppliers. Alternatively, the control could simply display a listing of all suppliers from a different database (see example below). Selecting the desired supplier entry must then call an existing Javascript function, AddItem() to add the entry to the PQM item. For more details, see the following reference example, and the Configuration section :



CustomerID	CompanyName	ContactName	ContactTitle	City	Region	PostalCode	Country	Phone
1	Exotic Liquids	Charlotte Cooper	Purchasing Manager	London		EC1 4SD	UK	(171) 555-2222
2	New Orleans Cajun Delights	Shelley Burke	Order Administrator	New Orleans	LA	70117	USA	(100) 555-4822
3	Grandma Kelly's Homestead	Regina Murphy	Sales Representative	Ann Arbor	MI	48104	USA	(313) 555-5735
4	Tokyo Traders	Yoshi Nagase	Marketing Manager	Tokyo		100	Japan	(03) 3555-5011
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra	Export Administrator	Oviedo	Asturias	33007	Spain	(98) 598 76 54
6	Mayumi's	Mayumi Ohno	Marketing Representative	Osaka		545	Japan	(06) 431-7877
7	Pavlova, Ltd.	Ian Devling	Marketing Manager	Melbourne	Victoria	3058	Australia	(03) 444-2343
8	Specialty Biscuits, Ltd.	Peter Wilson	Sales Representative	Manchester		M14 GSD	UK	(161) 555-4448
9	PB Knäckebröd AB	Lars Peterson	Sales Agent	Göteborg		S-345 67	Sweden	031-987 65 43
10	Refrescos Americanas LTDA	Carlos Diaz	Marketing Manager	Sao Paulo		5442	Brazil	(11) 555 4640

1 2 3

The above screenshot is an example of a control that displays data from a different database, using a .NET GridView control. This code for this simple GridView control is shown below.

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="NorthwindSuppliers.ascx.cs" Inherits="Referen
cePQMExtensions.Suppliers.NorthwindSuppliers" %>
<div>

    <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
        AllowSorting="True" AutoGenerateColumns="False" DataKeyNames="SupplierID"
        DataSourceID="SqlDataSource1">
        <Columns>
            <asp:BoundField DataField="SupplierID" HeaderText="CustomerID" ReadOnly="True"
                SortExpression="SupplierID" />
            <asp:BoundField DataField="CompanyName" HeaderText="CompanyName"
                SortExpression="CompanyName" />
            <asp:BoundField DataField="ContactName" HeaderText="ContactName"
                SortExpression="ContactName" />
            <asp:BoundField DataField="ContactTitle" HeaderText="ContactTitle"
                SortExpression="ContactTitle" />
            <asp:BoundField DataField="City" HeaderText="City" SortExpression="City" />
            <asp:BoundField DataField="Region" HeaderText="Region"
                SortExpression="Region" />
            <asp:BoundField DataField="PostalCode" HeaderText="PostalCode"
                SortExpression="PostalCode" />
            <asp:BoundField DataField="Country" HeaderText="Country"
                SortExpression="Country" />
            <asp:BoundField DataField="Phone" HeaderText="Phone" SortExpression="Phone" />
        </Columns>
    </asp:GridView>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="Data Source=(local);Initial Catalog=northwind;Integrated Security=true"
        SelectCommand="SELECT * FROM [Suppliers]"></asp:SqlDataSource>

</div>
```

In the code-behind, we add an event for when the row is clicked, which calls the AddItem function:

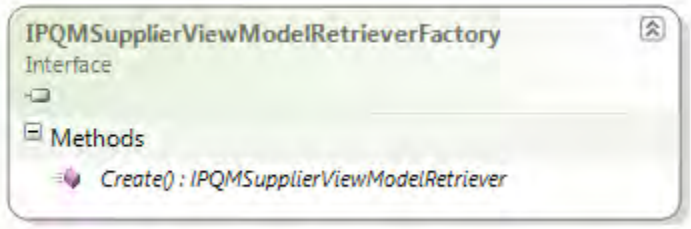
```
using System;
using System.Web.UI.WebControls;

namespace ReferencePQMExtensions.Suppliers
{
    public partial class NorthwindSuppliers : System.Web.UI.UserControl
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            GridView1.RowDataBound += gvSearch_RowDataBound;
        }

        protected void gvSearch_RowDataBound(object sender, GridViewRowEventArgs e)
        {
            if (e.Row.RowType == DataControlRowType.DataRow)
            {
                e.Row.Attributes.Add("onmouseover", "this.style.backgroundColor='#ceedfc'");
                e.Row.Attributes.Add("onmouseout", "this.style.backgroundColor=''");
                e.Row.Attributes.Add("style", "cursor:pointer;");
                e.Row.Attributes.Add("onclick", "AddItem('" + e.Row.Cells[0].Text + "','','Nor
thwindCompany','Northwind');");
            }
        }
    }
}
```

The control adds the item to the Suppliers listing of the PQM item using only the item's internal identifier, some external identifier, an item type, and the SourceSystemID. Next, a new class must be created to handle displaying data returned by this control. The company and facility names are not stored in the database, so that there is no issue with keeping multiple data sources in synch. This class is therefore responsible for retrieving the company (and optionally the facility name), along with an additional description and an optional URL to link to that company/facility.

View Model Retriever

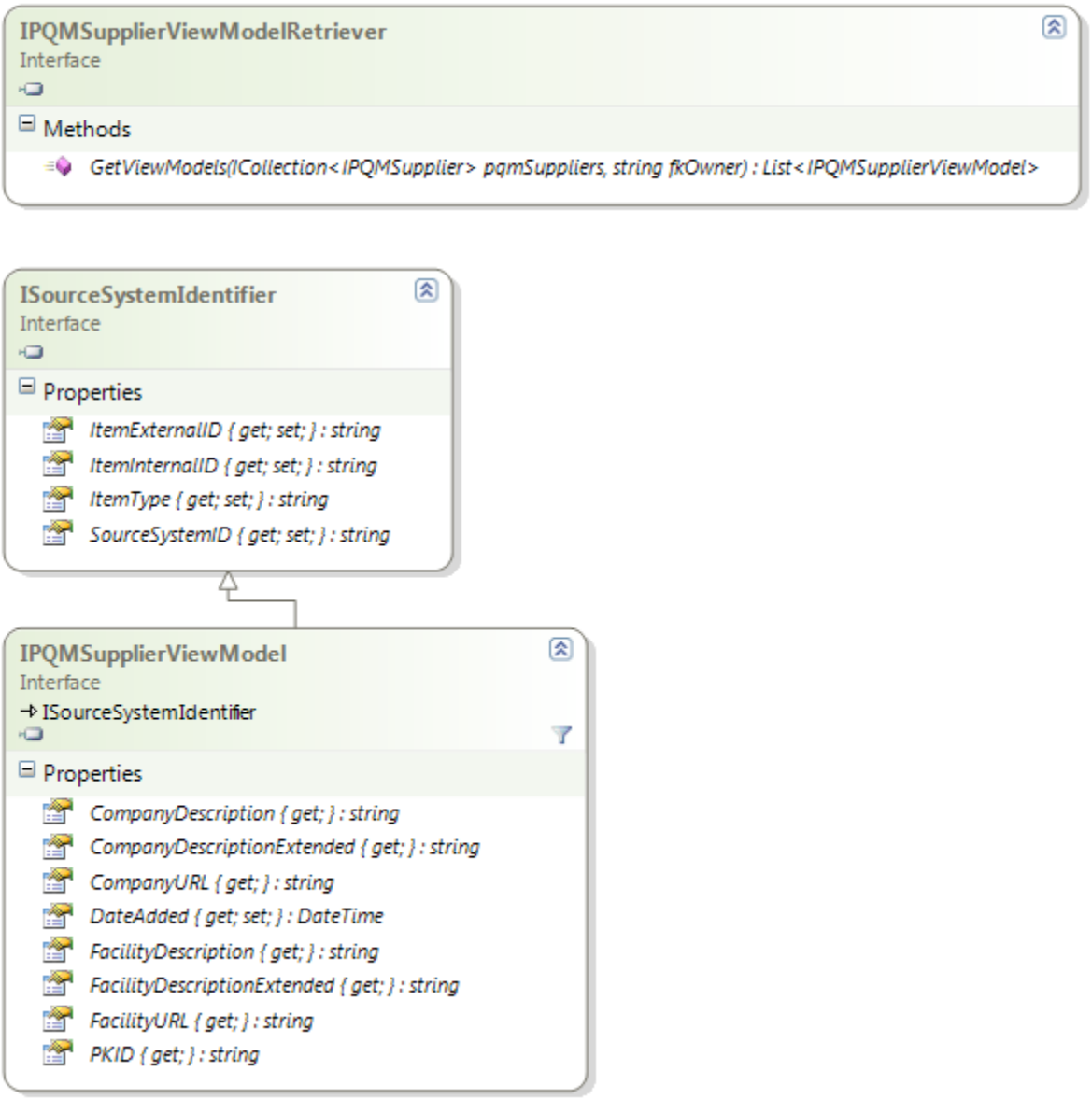
A custom SupplierRetriever class must be created that creates a ViewModel object for each supplier in the listing. The View Model is responsible for display of the supplier data in the User Interface.

The PQMSupplierViewModelRetrieverFactory simply creates and returns a PQMSupplierViewModelRetriever



The PQMSupplierViewModelRetriever's GetViewModels method takes a collection of PQMSupplier data objects which are stored on the PQM item, and takes the PKID of the owning PQM business object. The owner would be a PQM Action, Issue, or Audit, or alternatively, a PQMIssueAffectedItem data object which also has a supplier property. The retriever must then return a corresponding list of view models.

Given the list of PQMSupplier data objects, which hold the ItemInternalID, ItemExternalID, etc, those properties are then used by the retriever to query another system (using web services, for example) and retrieve the relevant information for display in the View Models.



Custom View Model classes can extend the PQMSupplierViewModelBase abstract class, to leverage some common properties and methods.

Example Code

The following code represents a simple implementation of a supplier view model and retriever class. The Retriever factory class simply creates a new view model Retriever. The Retriever's GetViewModels method takes the pqmSupplier collection, which holds the internal identifiers of the supplier, and queries the Northwind database to retrieve the Supplier's CompanyName and location information.

Note that the URL links for the company link to a non-existent web site. In this example, the Supplier data represents a company only, and not a facility.

```
public class NorthwindSupplierViewModelRetrieverFactory :IPQMSupplierViewModelRetrieverFactory
{
    public IPQMSupplierViewModelRetriever Create()
    {
        return new NorthwindSupplierViewModelRetriever();
    }
}

    public class NorthwindSupplierViewModelRetriever : IPQMSupplierViewModelRetriever
    {

        private const string _queryString = "SELECT CompanyName, City, Region, Country FROM dbo.Suppliers
where SupplierID = @SupplierID;";

        public List<IPQMSupplierViewModel> GetViewModels(ICollection<IPQMSupplier> pqmSuppliers, string fk
Owner)
        {
            List<IPQMSupplierViewModel> supplierViewModels = new List<IPQMSupplierViewModel>();
            foreach (var pqmSupplier in pqmSuppliers)
            {
                var supplier = LoadSupplier(pqmSupplier, fkOwner);
                if (supplier != null)
                    supplierViewModels.Add(supplier);
            }
            return supplierViewModels;
        }

        private IPQMSupplierViewModel LoadSupplier(IPQMSupplier pqmSupplier, string fkOwner)
        {

            IPQMSupplierViewModel northwindSupplier = null;

            using (var connection = new SqlConnection(NorthwindDBHelper.ConnectionString))
            {
                var command = new SqlCommand(_queryString, connection);
                command.Parameters.AddWithValue("@SupplierID", pqmSupplier.ItemInternalID);

                try
                {
                    connection.Open();
                    SqlDataReader reader = command.ExecuteReader();

                    if (reader.Read())
                    {
                        string companyDescription = reader.GetString(0);
                        string companyURL = "http://somenorthwind.com/supplier/" + pqmSupplier.ItemInterna
lID;

                        string city = reader.IsDBNull(1) ? String.Empty : reader.GetString(1);
                        string region = reader.IsDBNull(2) ? String.Empty : reader.GetString(2);
                        string country = reader.IsDBNull(3) ? String.Empty : reader.GetString(3);
                        string companyDescriptionExtended = String.Format("{0}, {1} {2}", city, region, co
untry);


                        northwindSupplier = new NorthwindSupplierViewModel(pqmSupplier
                            , fkOwner
                            , companyDescription
                            , companyDescriptionExtended
                            , companyURL);
                    }
                    reader.Close();
                }
                catch (Exception ex)
                {
                    northwindSupplier = new NorthwindSupplierViewModel(pqmSupplier
```

```
                        , fkOwner
                        , "Error Occurred"
                        , String.Format("Northwind record # {0} -
{1}", pqmSupplier.ItemInternalID, ex.Message )
                        , "http://somenorthwind.com/supplier/" + pqmSupplier.ItemInternalID);
                }
                return northwindSupplier;
            }
        }
    }
```

The sample View Model class extends the PQMSupplierViewModelBase abstract class. It sets the company information, and marks the Facility information as blank (the suppliers table used only has a single entity, so only a company is used here). In other scenarios, both the Company and the Facility can be used.

```
using System;
using Xeno.Data.PQM;
using Xeno.Prodika.Services.PQM.Models.Suppliers;

namespace ReferencePQMExtensions.Suppliers
{
    public class NorthwindSupplierViewModel : PQMSupplierViewModelBase
    {
        public NorthwindSupplierViewModel(IPQMSupplier pqmSupplier, string fkOwner, string com
panyName, string companyLocation, string homepage) : base(pqmSupplier, fkOwner)
        {
            CompanyDescription = companyName;
            CompanyDescriptionExtended = companyLocation;
            CompanyURL = homepage;
            FacilityDescription = String.Empty;
            FacilityDescriptionExtended = String.Empty;
            FacilityURL = String.Empty;
        }
    }
}
```

Configuration

In the CustomerSettings.config file, the PQM node has individual nodes for each PQM business object (Action/Audit/Issue). The SupplierRetrievers node contains PQMItemRetrieverConfig child nodes, which specify each data source.

The following configuration entry shows the default supplier configuration (where Supplier data is retrieved from a local SCRM instance), and is followed by the configuration used by the example shown above (SourceSystemID=”Northwind”):

```
<PQM>
  <Action>
  ...
    <SupplierRetrievers configChildKey="SourceSystemID">
     <PQMItemRetrieverConfig
SourceSystemID="SCRM"
SourceSystemTranslationID="lblSCRM"
       ItemRetrieverObjectURL="Class:Xeno.Prodika.Services.PQM.Models.Suppliers.SCRM.SCRMSuppl
ierViewModelRetrieverFactory,PQMLib$SCRM"
```

```
       UseEQT="true"
       EQTBaseURL="@@VAR:Prodika.PQM.URL@@"
       EQTConfiguration="SearchableView:Config:ProdikaSettings/EQTConfiguration/SearchableMult
iSelectViewsSCRM,CompanyFacilitySingleViewPopup"
       ItemViewBaseURL="@@VAR:Prodika.SCRM.URL@@"
       ItemAddControl="" />

   <PQMItemRetrieverConfig
       SourceSystemID="Northwind"
       SourceSystemTranslationID="lblNorthwind"

       ItemRetrieverObjectURL="Class:ReferencePQMExtensions.Suppliers.NorthwindSupplierViewMod
elRetrieverFactory,ReferencePQMExtensions"
       UseEQT="false"
       EQTBaseURL=""
       EQTConfiguration=""
       ItemViewBaseURL="http://northwinddb.com/suppliers/"
       ItemAddControl="NorthwindSuppliers.ascx" />
...
```

The configuration details are as follows:

Attribute	Description
SourceSystemID	Identifies where the Supplier data comes from. This value should be unique within the individual item type (Action/Audit/Issue) – that is, the Action SupplierRetrievers node should not contain 2 PQMItemRetrieverConfig entries with the same SourceSystemID values.
SourceSystemTranslationID	Will be used to display the name in the supplier section of the user interface if there is more than one entry configured for the PQM item type. When adding new translations, add a new entry into the `commonXLAExtensionCacheItem` table, where the `fkParent` value is the pkid of the '`frmPQM/Extension`' entry in the `commonXLAExtensionCache` table. For example: `insert into commonXLAExtensionCacheItem values ('1059'+UPPER(NEWID()), '10586A177FC6-F446-4FC2-885D-788B8C89AAF3', 0, 'lblNorthwind', 'Northwind Supplier');`
ItemRetrieverObjectURL	This customizable class is a factory (IPQMSupplierViewModelRetrieverFactory) that, given a list of supplier data objects (IPQMSupplier) stored in PQM, retrieves the related View Models (IPQMSupplierViewModel), which are classes that are used to generate the UI details. Custom classes can leverage the abstract base view model class, PQMSupplierViewModelBase, when creating their own view models.
UseEQT	true if using Agile PLM for Process SCRM; false otherwise
EQTBaseURL	Required if using SCRM; app location of the SCRM company/facility selection popup.
EQTConfiguration	Required if using SCRM; can be customized to specify different EQT views when searching for SCRM Companies and Facilities.
ItemViewBaseURL	Supplier company and facility entries in the user interface can be links – this setting specifies the base URL of SCRM based links, and can be used by

	the ViewModel retriever to create the URL links.
ItemAddControl	Required when supplier data is coming from a non-SCRM source. This value specifies the ASCX control that should be loaded in a PQM pop and that can manage the display of Suppliers from external systems. This value is ignored if the UseEQT attribute value is set to true. The custom control's responsibility is to provide a mechanism for users to search for company and/or facility supplier data, and select the entry that will be populated in the Suppliers listing. The control could, for example, allow users to enter some search criteria, and pass those values to a web service call to another system, returning any matching suppliers. Selecting the desired supplier entry must then call an existing javascript function, AddItem() to add the entry to the PQM item. The AddItem function takes the following parameters: **internalID** – the internal unique identifier of the entry. This will not be visible to users **ExternalID** – a visible identifier. This can be modified by providing additional ExternalID options in the ItemRetriever class, if needed. **ItemType** – a value that can be used to distinguish companies from facilities. **SourceSystemID** – a value that should match the SourceSystemID entry in the configuration, telling the system where the data came from. For PLM for Process SCRM, the value "SCRM" is used. The ASCX control must pass specific data to the main PQM window by calling the PQMSupplierItemAdd.aspx's AddItem() javascript function. The .ascx file must be placed in the PQMItemControls\Extensions folder in PQM. The compiled DLL for this control must be included in the PQM\bin directory.

The @@VAR:Prodika.SCRM.URL@@ value gets automatically replaced by the value in environmentvariables.config, but a different value can be specified here if needed.

SCRM residing on a different database

If SCRM data resides in a different database, the configuration entry of the ItemRetrieverObjectURL must be changed by appending the following, starting with the pipe (|) symbol:

|Class:Xeno.Prodika.Services.PQM.Models.DataManagerStrategy.ReportingDBDataManagerStrategy,PQMLib

So the attribute would be:

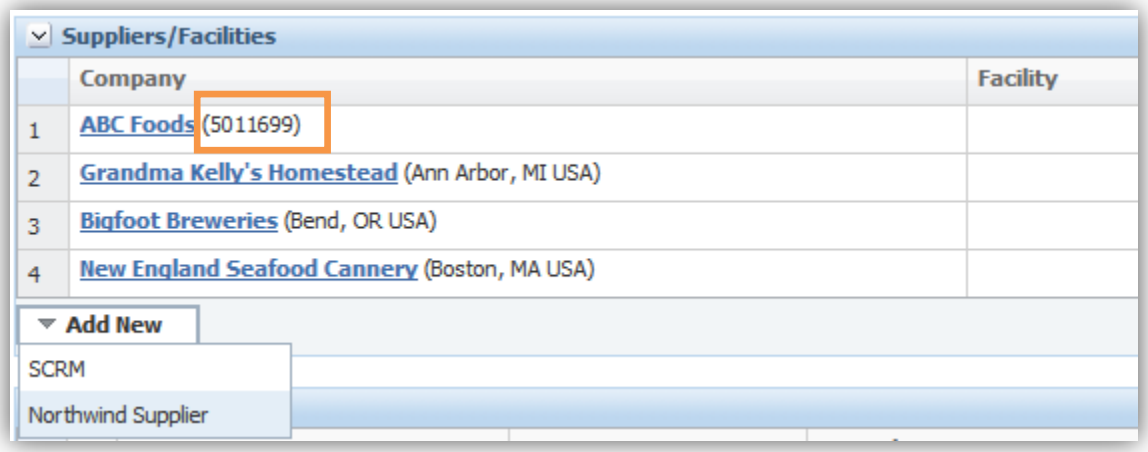
```
ItemRetrieverObjectURL="Class:Xeno.Prodika.Services.PQM.Models.Suppliers.SCRM.SCRMSupplierViewModelRetrieverFactory,PQMLib$SCRM|Class:Xeno.Prodika.Services.PQM.Models.DataManagerStrategy.ReportingDBDataManagerStrategy,PQMLib"
```

This configuration requires that the **Reporting DB connection pool** is configured (on the PQM application's configuration files) to point to the database connection where SCRM is hosted.

Additionally, the ItemViewBaseURL and EQTBaseURL must point to that separate SCRM web application.

Additional Supplier Formatting Extensibility

When using SCRM for facility and company supplier data, the user interface display in the Supplier listing can be customized by implementing any of the following Format Plugins.



- PQMActionSupplierFacility
- PQMActionSupplierCompany
- PQMAuditSupplierFacility
- PQMAuditSupplierCompany
- PQMIssueSupplierFacility
- PQMIssueSupplierCompany

The output of each plugin is used to display additional facility or company information. By default, these plugins will display the equivalent value of the user's SCRM preferred cross reference, if available; otherwise, the company or facility number is displayed.

When configuring any of the above FormatPlugins, be sure to include the following XML attribute:

ignoreInheritFromPluginName="true"

See the PluginExtensions documentation, available in the Extensibility Pack, for details on writing FormatPlugins.

Note that in the above screenshot, the entries that came from an external system (Northwind) are specifying the extended description on the ViewModelRetriever class, rather than pulling in the Format Plugins, while the data from SCRM is using the default FormatPlugin.

Affected Items Extensibility

The source of PQM Affected Items data for PQM Actions, Issues, and Audits can be customized to pull data from the current/local GSM instance, some other instance of PLM for Process GSM, or from entirely different applications.

Affected Item Source Data

The out of the box configuration assumes the Agile PLM for Process GSM application is local and on a shared database. Using an alternate instance of GSM, in which PQM and GSM reside on *different* databases, can be done by some simple configuration changes. Pulling in Affected Item data from other systems requires that a custom ASCX control be created and deployed into PQM, as well as a class which takes the PQM affected item data and generates the formatted view for display in the UI. Multiple configurations of affected item data are permitted, which allow PQM affected item data to be pulled from multiple applications.

Additionally, the configuration for PQM Affected Items is flexible enough to allow different configurations for Actions, Audits, and Issues, if needed.

Local GSM

Assumes the PQM Affected Item data is populated from GSM residing on the same database. This is the default configuration for PQM Affected Items, so no configuration changes are required.

External GSM

Agile PLM for Process GSM that is deployed on a *separate* database from PQM, configuration changes are required that specify the server URL and database connection. The configuration setting changes indicate the GSM server URL. Additionally, this configuration requires that the **Reporting DB connection pool** is configured (on the PQM application's configuration files) to point to the database connection where GSM is hosted.

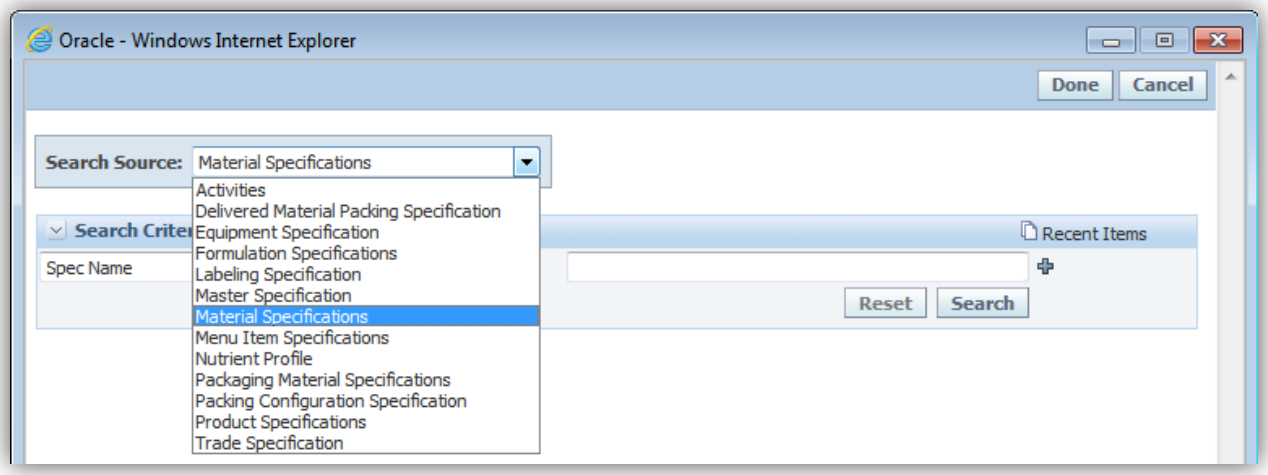
See the Configuration > GSM Residing on a Different Database section in this document.

Trusted Site

Additionally, when searching for GSM Specification data in another server from PQM, users must update their Internet Explorer browser settings to add that GSM web site as a Trusted Site. Failing to do so will result in the GSM EQT Popup being unable to post the search results back to PQM.

GSM EQT Feature Configuration

When adding a GSM Specifications to the affected items listing, the search popup includes all specification types.



To limit which specification types should be available, configuration entries can be configured separately for Actions, Audits, and Issues. The following FeatureConfiguration entries control this access, where the 4 digit number represents the specification types:

Action	Audit	Issue
PQM.Action.AffectedItem.1004.Enabled	PQM.Audit.AffectedItem.1004.Enabled	PQM.Issue.AffectedItem.1004.Enabled
PQM.Action.AffectedItem.1005.Enabled	PQM.Audit.AffectedItem.1005.Enabled	PQM.Issue.AffectedItem.1005.Enabled
PQM.Action.AffectedItem.1006.Enabled	PQM.Audit.AffectedItem.1006.Enabled	PQM.Issue.AffectedItem.1006.Enabled
PQM.Action.AffectedItem.1009.Enabled	PQM.Audit.AffectedItem.1009.Enabled	PQM.Issue.AffectedItem.1009.Enabled
PQM.Action.AffectedItem.1010.Enabled	PQM.Audit.AffectedItem.1010.Enabled	PQM.Issue.AffectedItem.1010.Enabled
PQM.Action.AffectedItem.2076.Enabled	PQM.Audit.AffectedItem.2076.Enabled	PQM.Issue.AffectedItem.2076.Enabled
PQM.Action.AffectedItem.2121.Enabled	PQM.Audit.AffectedItem.2121.Enabled	PQM.Issue.AffectedItem.2121.Enabled
PQM.Action.AffectedItem.2147.Enabled	PQM.Audit.AffectedItem.2147.Enabled	PQM.Issue.AffectedItem.2147.Enabled
PQM.Action.AffectedItem.2280.Enabled	PQM.Audit.AffectedItem.2280.Enabled	PQM.Issue.AffectedItem.2280.Enabled
PQM.Action.AffectedItem.2283.Enabled	PQM.Audit.AffectedItem.2283.Enabled	PQM.Issue.AffectedItem.2283.Enabled
PQM.Action.AffectedItem.5750.Enabled	PQM.Audit.AffectedItem.5750.Enabled	PQM.Issue.AffectedItem.5750.Enabled
PQM.Action.AffectedItem.5816.Enabled	PQM.Audit.AffectedItem.5816.Enabled	PQM.Issue.AffectedItem.5816.Enabled
PQM.Action.AffectedItem.6500.Enabled	PQM.Audit.AffectedItem.6500.Enabled	PQM.Issue.AffectedItem.6500.Enabled
PQM.Action.AffectedItem.6501.Enabled	PQM.Audit.AffectedItem.6501.Enabled	PQM.Issue.AffectedItem.6501.Enabled

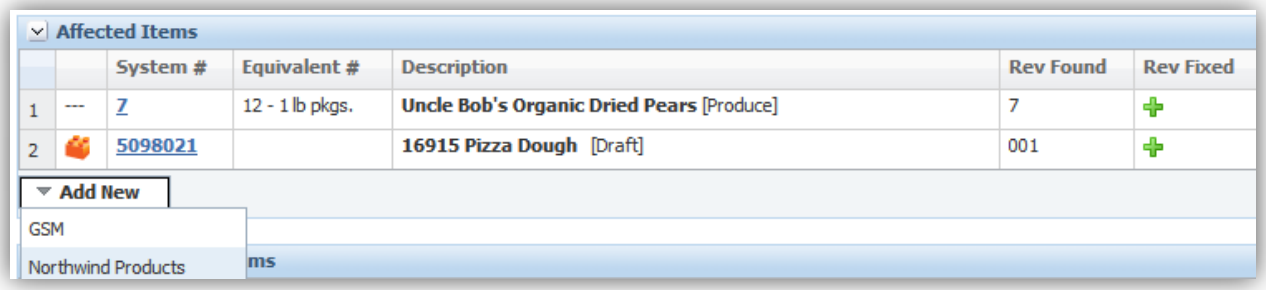
Also, when using PQM to search for Actions, Issues, or Audits that have a specific GSM Specification, different configurations control whether or not the search options should include a local (same DB) GSM spec search and/or a remote/external GSM spec search. The following FeatureConfiguration entries enable the behaviors:

```
<add key="PQM.EQT.Action.AffectedItemGSMSearch.Local.Enabled"     value="true"
configDescription="Sets PQM Action EQT Search of Affected Items by local GSM DB"/>
<add key="PQM.EQT.Action.AffectedItemGSMSearch.External.Enabled"  value="false"
configDescription="Sets PQM Action EQT Search of Affected Items by External GSM DB"/>

<add key="PQM.EQT.Audit.AffectedItemGSMSearch.Local.Enabled"      value="true"
configDescription="Sets PQM Audit EQT Search of Affected Items by local GSM DB"/>
```

```
<add key="PQM.EQT.Audit.AffectedItemGSMSearch.External.Enabled"   value="false"
configDescription="Sets PQM Audit EQT Search of Affected Items by External GSM DB"/>

<add key="PQM.EQT.Issue.AffectedItemGSMSearch.Local.Enabled"      value="true"
configDescription="Sets PQM Issue EQT Search of Affected Items by local GSM DB"/>
<add key="PQM.EQT.Issue.AffectedItemGSMSearch.External.Enabled"   value="false"
configDescription="Sets PQM Issue EQT Search of Affected Items by External GSM DB"/>
```

Alternate Affected Item Source Systems

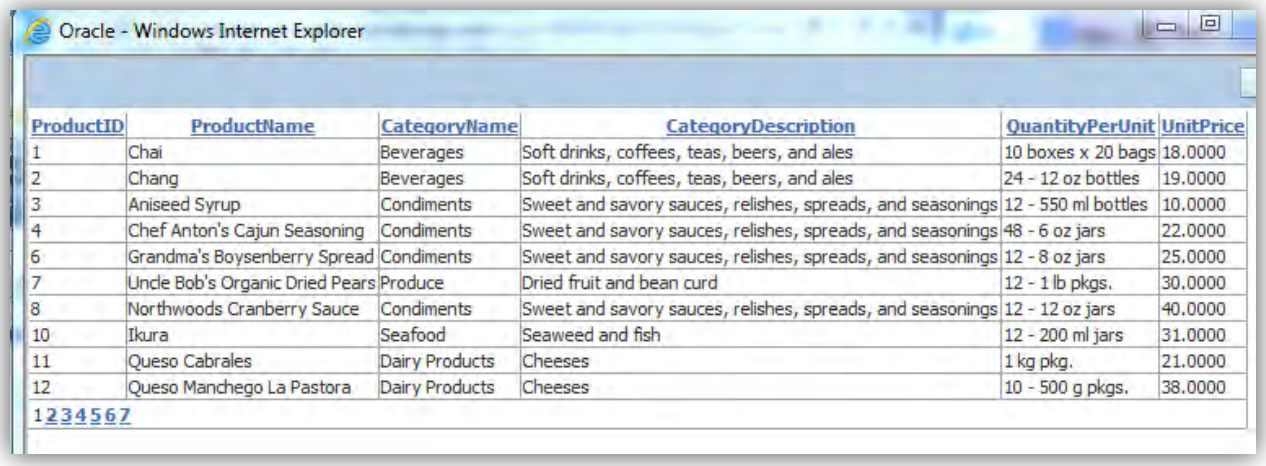
When pulling Affected Item data from alternate systems, a custom ASCX control must be created and plugged into PQM that allows for searching and selecting an affected item such as a specification. This item is then added to the Affected Items listing in PQM via the hosting aspx page's AddItem Javascript function.



The configured list of possible Affected Item sources is added to the Add New button. Clicking on the external system launches the custom control.

ASCX control

The custom control's responsibility is to provide a mechanism for users to search for affected item data and then select the entry that will be populated in the Affected Items listing. The control could, for example, allow users to enter some search criteria, and pass those values to a web service call to another system, returning any matching suppliers. Alternatively, the control could simply display a listing of all products from a different database (see the following example). Selecting the desired affected item entry must then call an existing Javascript function, AddItem() to add the entry to the PQM item. For more details, see the following reference example, and the Configuration section.



ProductID	ProductName	CategoryName	CategoryDescription	QuantityPerUnit	UnitPrice
1	Chai	Beverages	Soft drinks, coffees, teas, beers, and ales	10 boxes x 20 bags	18.0000
2	Chang	Beverages	Soft drinks, coffees, teas, beers, and ales	24 - 12 oz bottles	19.0000
3	Aniseed Syrup	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	12 - 550 ml bottles	10.0000
4	Chef Anton's Cajun Seasoning	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	48 - 6 oz jars	22.0000
6	Grandma's Boysenberry Spread	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	12 - 8 oz jars	25.0000
7	Uncle Bob's Organic Dried Pears	Produce	Dried fruit and bean curd	12 - 1 lb pkgs.	30.0000
8	Northwoods Cranberry Sauce	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	12 - 12 oz jars	40.0000
10	Ikura	Seafood	Seaweed and fish	12 - 200 ml jars	31.0000
11	Queso Cabrales	Dairy Products	Cheeses	1 kg pkg.	21.0000
12	Queso Manchego La Pastora	Dairy Products	Cheeses	10 - 500 g pkgs.	38.0000

1 2 3 4 5 6 7

The above screenshot is an example of a control that displays data from a different database, using a .NET GridView control. The example code for this simple GridView control is shown below.

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="NorthwindProducts.ascx.cs" Inherits="Referenc
ePQMExtensions.AffectedItems.NorthwindProducts" %>
<div>
    <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
        AllowSorting="True" AutoGenerateColumns="False" DataKeyNames="ProductID"
        DataSourceID="SqlDataSource1">
        <Columns>
            <asp:BoundField DataField="ProductID" HeaderText="ProductID" ReadOnly="True"
                SortExpression="ProductID" />
            <asp:BoundField DataField="ProductName" HeaderText="ProductName"
                SortExpression="ProductName" />
            <asp:BoundField DataField="CategoryName" HeaderText="CategoryName"
                SortExpression="CategoryName" />
            <asp:BoundField DataField="Description" HeaderText="CategoryDescription"
                SortExpression="Description" />
            <asp:BoundField DataField="QuantityPerUnit" HeaderText="QuantityPerUnit" SortExpression="Quant
ityPerUnit" />
            <asp:BoundField DataField="UnitPrice" HeaderText="UnitPrice"
                SortExpression="RegionUnitPrice" />

        </Columns>
    </asp:GridView>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="Data Source=(local);Initial Catalog=northwind;Integrated Security=true"
        SelectCommand="SELECT p.*, c.CategoryName, c.Description FROM Products p inner join Categories c o
n p.CategoryID = c.CategoryID where discontinued = 0"></asp:SqlDataSource>
</div>
```

In the code-behind, we add an event for when the row is clicked, which calls the AddItem function:

```
using System;
using System.Web.UI.WebControls;

namespace ReferencePQMExtensions.AffectedItems
{
    public partial class NorthwindProducts : System.Web.UI.UserControl
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            GridView1.RowDataBound += gvSearch_RowDataBound;
        }
```

```
        protected void gvSearch_RowDataBound(object sender, GridViewRowEventArgs e)
        {
            if (e.Row.RowType == DataControlRowType.DataRow)
            {
                e.Row.Attributes.Add("onmouseover", "this.style.backgroundColor='#ceedfc'");
                e.Row.Attributes.Add("onmouseout", "this.style.backgroundColor=''");
                e.Row.Attributes.Add("style", "cursor:pointer;");
                e.Row.Attributes.Add("onclick", "AddItem('" + e.Row.Cells[0].Text + "','" + e.
Row.Cells[2].Text + "','NorthwindProduct','" + e.Row.Cells[0].Text + "','" + e.Row.Cells[0].Te
xt + "','Northwind');");
            }
        }
    }
}
```

The control adds the item to the Affected Items listing of the PQM item using only the item's internal identifier, some external identifier, an item type, and the SourceSystemID. Next, a new class must be created to handle displaying data returned by this control. The affected item's name is not stored in the database, so that there is no issue with keeping multiple data sources in synch. This class is therefore responsible for retrieving the product name, along with an additional description and an optional URL to link to that product.

View Model Retriever

A custom Affected Item Retriever class must be created that creates a View Model object for each Affected Item in the listing. The View Model is responsible for display of the affected item data in the User Interface. Affected Items for Issues have slightly different Retriever and View Model interfaces.

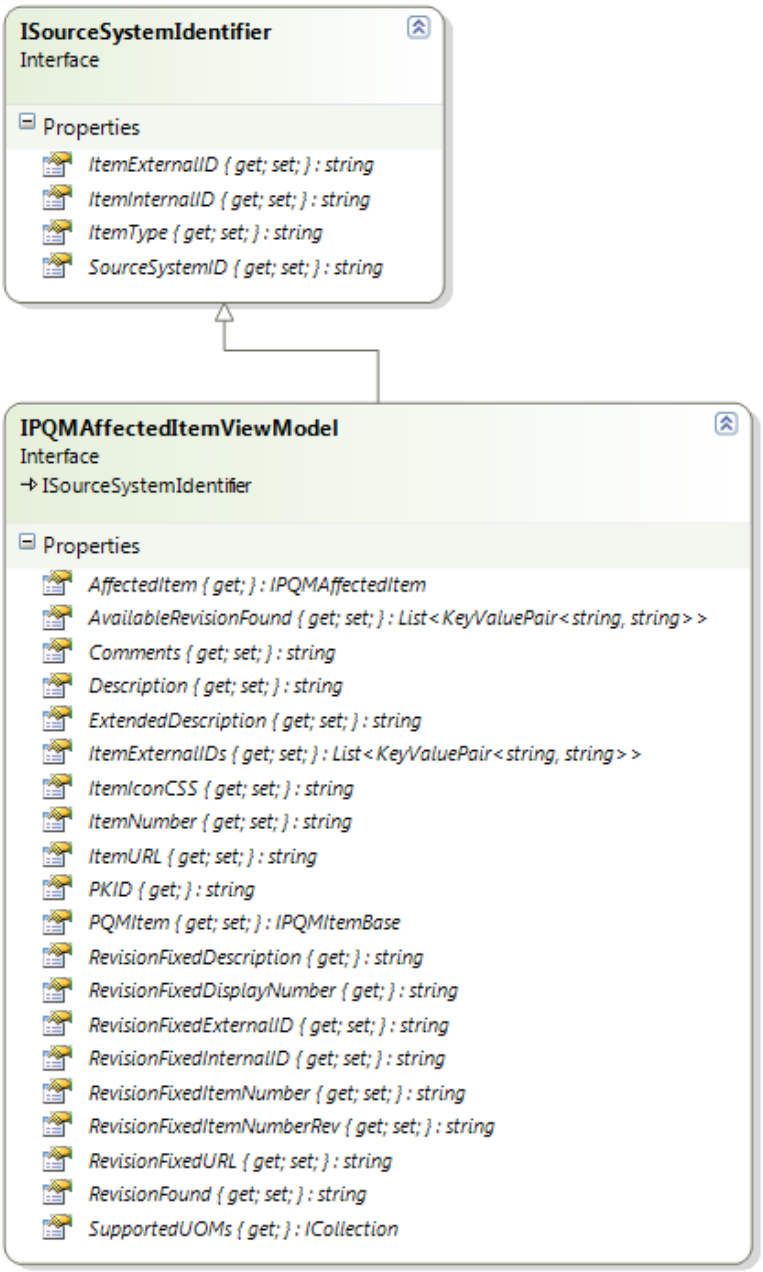
The PQMAffectedItemViewModelRetrieverFactory simply creates and returns a PQMAffectedItemViewModelRetriever

```
public interface IPQMAffectedItemViewModelRetrieverFactory<TViewModel, TDataObject>
   where TViewModel : IPQMAffectedItemViewModel
   where TDataObject : IPQMAffectedItem
{
   IPQMAffectedItemViewModelRetriever<TViewModel, TDataObject> Create();
}

public interface IPQMAffectedItemViewModelRetrieverActionFactory :
IPQMAffectedItemViewModelRetrieverFactory<IPQMAffectedItemViewModel, IPQMAffectedItem> { }

public interface IPQMAffectedItemViewModelRetrieverAuditFactory  :
IPQMAffectedItemViewModelRetrieverFactory<IPQMAffectedItemViewModel, IPQMAffectedItem> { }

public interface IPQMAffectedItemViewModelRetrieverIssueFactory  :
IPQMAffectedItemViewModelRetrieverFactory<IPQMIssueAffectedItemViewModel,
IPQMIssueAffectedItem> { }
```

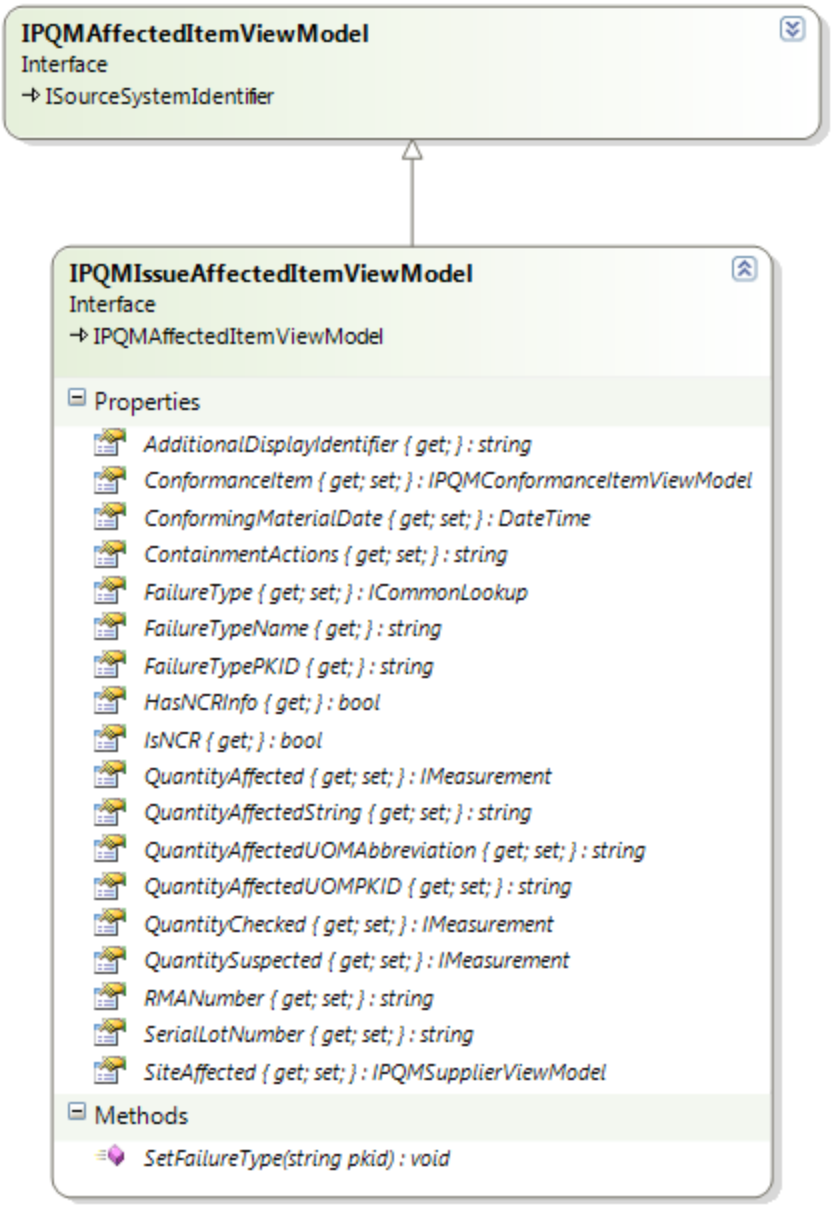
Each Retriever returns a list of View Models via the GetViewModels method.

The Retriever's GetViewModels method takes a collection of PQMAffectedItem data objects (or PQMIssueAffectedItem data objects for PQM Issues) which are stored on the PQM item. The retriever must then return a corresponding list of view models. Given the list of PQMAffectedItem data objects, which hold the ItemInternalID, ItemExternalID, etc, those properties are then used by the retriever to query another system (using web services or SQL queries for example) and retrieve the relevant information for display in the View Models.

View Models of Actions and Audits must implement the IPQMAffectedItemViewModel:



PQM Issue View Models must implement the IPQMIssueAffectedItemViewModel:



Custom AffectedItemViewModel classes can extend the PQMAffectedItemViewModelBase abstract class. Note that the View Model has a property named ItemIconCSS, which is used to display custom icons representing the affected item type.

Example Code

The following code represents a simple implementation of an Affected Items view model retriever. The retriever factory simply creates a new retriever class. The Retriever class delegates the load of Northwind product data to the View Model.

```
using System.Collections.Generic;
using Xeno.Data.PQM;
using Xeno.Prodika.Services.PQM.Models;
```

```
namespace ReferencePQMExtensions.AffectedItems
{
    public class NorthwindProductsAffectedItemViewModelRetrieverActionFactory : IPQMAffectedIt
emViewModelRetrieverActionFactory
    {
        public IPQMAffectedItemViewModelRetriever<IPQMAffectedItemViewModel
, IPQMAffectedItem> Create()
        {
            return new NorthwindProductsAffectedItemViewModelRetriever();
        }
    }

    public class NorthwindProductsAffectedItemViewModelRetriever : IPQMAffectedItemViewModelRe
triever<IPQMAffectedItemViewModel, IPQMAffectedItem>
    {
        public List<IPQMAffectedItemViewModel> GetViewModels(ICollection<IPQMAffectedItem> pqm
AffectedItems)
        {
            List<IPQMAffectedItemViewModel> viewModels =new List<IPQMAffectedItemViewModel>();

            foreach (var pqmAffectedItem in pqmAffectedItems)
            {
                viewModels.Add(new NorthwindProductsAffectedItemViewModel(pqmAffectedItem));
            }

            return viewModels;
        }
    }
}
```

The sample View Model class extends the PQMAffectedItemViewModelBase abstract class. It loads and assigns the product information from the Northwind database, and if the Revision Fixed internalID is set, retrieves and assigns that from the database.

```
using System.Collections.Generic;
using Xeno.Data.PQM;
using Xeno.Prodika.Services.PQM.Models;

namespace ReferencePQMExtensions.AffectedItems
{
    public class NorthwindProductsAffectedItemViewModel : PQMAffectedItemViewModelBase
    {
        private IPQMAffectedItem PqmAffectedItem { get; set; }
        private string _revisionFixedDescription;
        private string _revisionFixedDisplayNumber;

        public NorthwindProductsAffectedItemViewModel(IPQMAffectedItem pqmAffectedItem) : base
(pqmAffectedItem)
        {
            PqmAffectedItem = pqmAffectedItem;
            LoadProduct();
        }

        private void LoadProduct()
        {
            //Load Revision Found
```

```
            var itemDesc = RevisionDescriptionLoader.GetDescriptionForProduct(PqmAffectedItem.
ItemInternalID);
            Description         = itemDesc.Description;
            ExtendedDescription = itemDesc.DescriptionExtended;
            ItemExternalID      = itemDesc.ExternalID;
            ItemURL             = itemDesc.ItemURL;

            AvailableRevisionFound = new List<KeyValuePair<string, string>>() { new KeyValuePa
ir<string, string>(ItemInternalID, ItemInternalID) };
            ItemExternalIDs = new List<KeyValuePair<string, string>>() { new KeyValuePair<stri
ng, string>(ItemExternalID, ItemExternalID) };


            //Load Revision Fixed
            var revisionItem = RevisionDescriptionLoader.GetDescriptionForProduct(RevisionFixe
dInternalID);
            _revisionFixedDescription = revisionItem.Description;
            _revisionFixedDisplayNumber = RevisionFixedInternalID;
            RevisionFixedItemNumberRev = RevisionFixedInternalID;
            RevisionFixedURL = revisionItem.ItemURL;
        }

        public override string RevisionFixedDisplayNumber { get { return _revisionFixedDisplay
Number; } }
        public override string RevisionFixedDescription {get { return _revisionFixedDescriptio
n; }}
        public override string RevisionFixedURL { get; set; }
        public override string ItemIconCSS { get; set; }
        public override string ItemURL { get; set; }
        public override string ExtendedDescription { get; set; }
        public override List<KeyValuePair<string, string>> AvailableRevisionFound { get; set;
}
        public override List<KeyValuePair<string, string>> ItemExternalIDs { get; set; }
        public override string Description { get; set; }

    }
}
```

The RevisionDescriptionLoader queries a database for product information to display for the affected item.

```
internal class RevisionDescriptionLoader
    {
        private const string _queryString = @"SELECT p.ProductName, c.CategoryName, p.Quantity
PerUnit  FROM Products p inner join Categories c on p.CategoryID = c.CategoryID
         where p.ProductID = @ProductID;";

        internal static RevisionDescription GetDescriptionForProduct(string productID)
        {
            if (String.IsNullOrEmpty(productID))
                return new RevisionDescription() {Description = String.Empty};

            RevisionDescription item = new RevisionDescription();
            using (var connection = new SqlConnection(NorthwindDBHelper.ConnectionString))
            {
                var command = new SqlCommand(_queryString, connection);
                command.Parameters.AddWithValue("@ProductID", productID);
```

```
                try
                {
                    connection.Open();
                    SqlDataReader reader = command.ExecuteReader();

                    if (reader.Read())
                    {

                        item.Description = reader.IsDBNull(0) ? String.Empty : reader.GetStrin
g(0);
                        item.DescriptionExtended = reader.IsDBNull(1) ? String.Empty : String.
Format("[{0}]", reader.GetString(1)); //category name
                        item.ExternalID = reader.IsDBNull(2) ? String.Empty : reader.GetString
(2); //quantityperunit
                        item.ItemURL = "http://somenorthwind.com/product/" + productID;


                    }
                    reader.Close();
                }
                catch (Exception ex)
                {
                    item.Description = "Error Occurred";
                    item.DescriptionExtended = String.Format("Northwind Product record # {0} -
 {1}", productID, ex.Message);
                }

            }
            return item;
        }
    }
```

Remember that affected items for a PQM Issue have additional fields if the Issue is a Non-Conformance report (NCR) issue type. Therefore, the view model and retriever interfaces are different than for the affected items of an Action or Audit. See the previous class diagram.

Configuration

In the CustomerSettings.config file, the PQM node has individual nodes for each PQM business object (Action/Audit/Issue). Like the PQM Suppliers configuration, the AffectedItemRetrievers node contains PQMItemRetrieverConfig child nodes, which specify each data source.

The following configuration entry shows the default affected item configuration (where Affected Item data is retrieved from a local GSM instance), and is followed by the configuration used by the example shown above (SourceSystemID=”Northwind”):

```
<PQM>
  <Action>
    <AffectedItemRetrievers configChildKey="SourceSystemID">
      <PQMItemRetrieverConfig
    SourceSystemID="GSM"
```

```
    SourceSystemTranslationID="lblGSM"
ItemRetrieverObjectURL="Class:Xeno.Prodika.Services.PQM.Models.GSM.GSMSpecAffectedItemViewMode
lRetrieverActionFactory,PQMLib$GSM"
    UseEQT="true"
    EQTBaseURL="@@VAR:Prodika.PQM.URL@@"
EQTConfiguration="SearchableView:Config:ProdikaSettings/EQTConfiguration/SearchableMultiSelect
ViewsGSM,PQMActionSpecSummaryView"
    ItemViewBaseURL="@@VAR:Prodika.GSMView.URL@@"
    ItemAddControl="" />

<PQMItemRetrieverConfig
       SourceSystemID="Northwind"
       SourceSystemTranslationID="lblNorthwindProducts"

       ItemRetrieverObjectURL="Class:ReferencePQMExtensions.AffectedItems.NorthwindProductsAff
ectedItemViewModelRetrieverActionFactory,ReferencePQMExtensions"
       UseEQT="false"
       EQTBaseURL=""

       EQTConfiguration="SearchableView:Config:ProdikaSettings/EQTConfiguration/SearchableMult
iSelectViewsGSM,PQMActionSpecSummaryViewMultiSelect"
       ItemViewBaseURL="http://northwinddb.com/products/"
       ItemAddControl="NorthwindProducts.ascx" />
...
```

Due to a bug, the EQTConfiguration entry is required even for external systems, though it is not used.

Attribute	Description
SourceSystemID	Identifies where the Affected Item data comes from. This value should be unique within the individual item type (Action/Audit/Issue) – that is, the Action AffectedItemRetrievers node should not contain 2 PQMItemRetrieverConfig entries with the same SourceSystemID values.
SourceSystemTranslationID	The translation value will be used to display the name in the Affected Item section of the user interface if there is more than one entry configured for the PQM item type. When adding new translations, add a new entry into the `commonXLAExtensionCacheItem` table, where the `fkParent` value is the pkid of the `'frmPQM/Extension'` entry in the `commonXLAExtensionCache` table. For example: `insert into commonXLAExtensionCacheItem values ('1059'+UPPER(NEWID()), '10586A177FC6-F446-4FC2-885D-788B8C89AAF3', 0, 'lblNorthwindProducts', 'Northwind Products');`
ItemRetrieverObjectURL	This customizable class is a factory (IPQMAffectedItemViewModelRetrieverFactory) that, given a list of Affected Item data objects (IPQMAffectedItem or IPQMIssueAffectedItem) stored in PQM, retrieves the related View Models (IPQMAffectedItemViewModel or IPQMIssueAffectedItemViewModel), which are classes that are used to generate the UI details. Custom classes can leverage the abstract base view model class,

	PQMAffectedItemViewModelBase, when creating their own view models.
UseEQT	true if using GSM data; false otherwise
EQTBaseURL	Required if using GSM; app location of the GSM spec selection popup.
EQTConfiguration	Required if using GSM; can be customized to specify different EQT views when searching for GSM Specifications. ** Note that due to a bug, the EQTConfiguration entry is required even for external systems, though it is not used.
ItemViewBaseURL	Affected Item entries in the user interface can be links – this setting specifies the base URL of GSM Spec-based links, and can be used by the ViewModel retriever to create the URL links.
ItemAddControl	Only required when supplier data is coming from a non-GSM data source. This value specifies the ASCX control that should be loaded in a PQM pop and that can manage the display of Affected Item data from external systems. This value is ignored if the UseEQT attribute value is set to true. The custom control's responsibility is to provide a mechanism for users to search for affected item data, and select the entry that will be populated in the Affected Items listing. The control could, for example, allow users to enter some search criteria, and pass those values to a web service call to another system, returning any matching items. Selecting the desired affected item entry must then call an existing javascript function, AddItem() to add the entry to the PQM item. The AddItem function takes the following parameters: **internalID** – the internal unique identifier of the entry. This will not be visible to users **ExternalID** – a visible identifier. This can be modified by providing additional ExternalID options in the ItemRetriever class, if needed. **ItemType** – a value that can be used to distinguish affected item types. For GSM specifications, for instance, this represents the 4 digit specification type (e.g. 1004 for Material Specs) **ItemNumber** – the main user interface identifier for the item. For GSM specifications, for instance, this value is the specification number (without the issue number). **ItemRevision** – the version number for the item. For GSM specifications, for instance, this value is the specification's issue number. **SourceSystemID** – a value that should match the SourceSystemID entry in the configuration, telling the system where the data came from. For PLM for Process GSM, the value "GSM" is used. The ASCX control must pass specific data to the main PQM window by calling the PQMCustomAffectedItemAdd.aspx's AddItem() javascript

	function. The .ascx file must be placed in the PQMItemControls\Extensions folder in PQM. The compiled DLL for this control must be included in the PQM\bin directory.

The @@VAR:Prodika.GSM.URL@@ value gets automatically replaced by the value in environmentvariables.config, but a different value can be specified here if needed.

GSM Residing on a Different Database

If GSM data resides in a different database, the configuration entry of the ItemRetrieverObjectURL must be changed by appending the following, starting with the pipe (|) symbol:

|Class:Xeno.Prodika.Services.PQM.Models.DataManagerStrategy.ReportingDBDataManagerStrategy,PQMLib

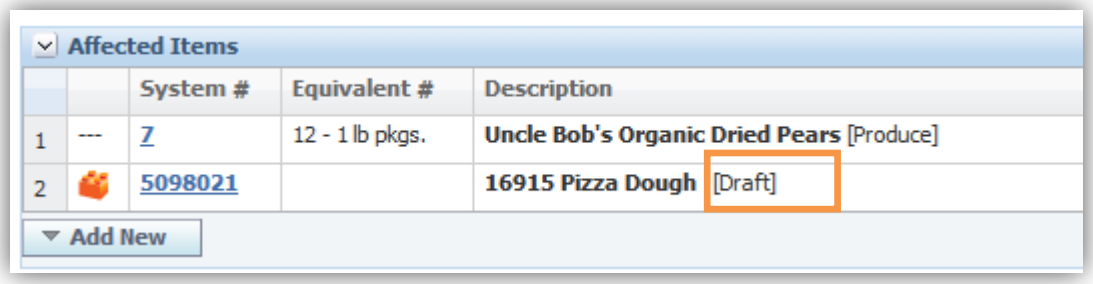
So the attribute would be:

ItemRetrieverObjectURL="Class:Xeno.Prodika.Services.PQM.Models.GSM.GSMSpecAffectedItemViewModelRetrieverActionFactory,PQMLib$GSM|Class:Xeno.Prodika.Services.PQM.Models.DataManagerStrategy.ReportingDBDataManagerStrategy,PQMLib"

This configuration requires that the **Reporting DB connection pool** is configured (on the PQM application's configuration files) to point to the database connection where GSM is hosted.

Additional Affected Items Formatting Extensibility

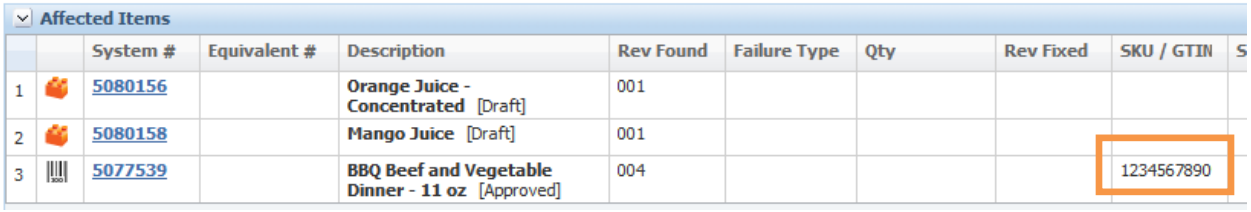
When using GSM for affected item data, the user interface display in the Affected Items listing can be customized by implementing Format Plugins. By default, the GSM Specs listed here use the SpecStatusIdentityPlugin to append the spec's status to the description.



The following Format Plugins are used:

- PQMActionAffectedItems
- PQMActionAffectedItemsRevFixed
- PQMAuditAffectedItems
- PQMAuditAffectedItemsRevFixed
- PQMIssueAffectedItems
- PQMIssueAffectedItemsRevFixed

The affected items for an Issue have an additional format plugin (plugin name is PQMIssueAffectedItemsAdditionalIdentifier) available used for the SKU/GTIN column. By default, the GSM Specs listed here use a plugin which shows the GTIN number.



Affected Items

		System #	Equivalent #	Description	Rev Found	Failure Type	Qty	Rev Fixed	SKU / GTIN	S
1		5080156		Orange Juice - Concentrated [Draft]	001					
2		5080158		Mango Juice [Draft]	001					
3		5077539		BBQ Beef and Vegetable Dinner - 11 oz [Approved]	004				1234567890	

The output of each plugin is used to display any additional desired information.

When configuring any of the above FormatPlugins, be sure to include the following XML attribute:

ignoreInheritFromPluginName="true"

See the PluginExtensions documentation, available in the Extensibility Pack, for details on writing FormatPlugins.

Note that in the above screenshot, the entries that came from an external system (Northwind) are specifying the extended description on the ViewModelRetriever class, rather than pulling in the Format Plugins, while the data from GSM is using the default FormatPlugins.

Affected Item Persistence

When Affected Items data pulled into PQM, it is saved in the pqmAffectedItem table for PQM Actions and Audits and in the pqmIssueAffectedItem table for PQM Issues. The following core data is saved from the source system, and is then passed to the Retriever classes to generate a View Model for displaying user friendly information.

- SourceSystemID – name of the source system, as configured in the PQMItemRetrieverConfig xml entries. When data is retrieved from the table for display, the Retriever class with a matching SourceSystemID is used to generate the View Models.
- ItemInternalID – internal, non-display, unique identifier for the affected item. For GSM Specs, this field contains the Spec ID (PKID). This is used by the Retriever classes to load relevant details for display
- ItemRevisionFound – the revision of the Affected Item. For GSM Specs, this field contains the issue number.
- ItemType – an identifier to specify the affected item type. For GSM Specs, this represents the spec type (1004: Material Specs, 2147: Trade Specs, etc.)
- ItemNumber – the primary identifier of the affected item. For GSM Specs, this represents the specification number (without the issue number)
- ItemExternalID – an alternate identifier for the affected item, displayed in the user interface. The Retriever class can provide a list of possible selections for the user to choose from.

PQM Web Services

A comprehensive set of web services are available for integration with PQM. See the Web Service Guide for details.

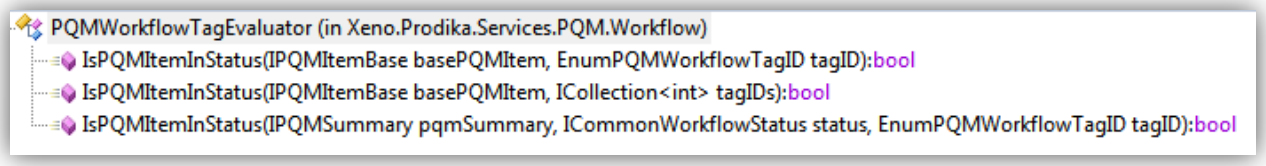
Utility Classes

Several useful utility classes are available to assist external developers with Agile PLM for Process PQM extensibility development. Custom Validators, Workflow Actions and Workflow Guard Conditions, Plugins, Calculation Extensions, and other extensibility points can leverage these utility classes by referencing the PQMLib.dll.

The following utility classes are available:

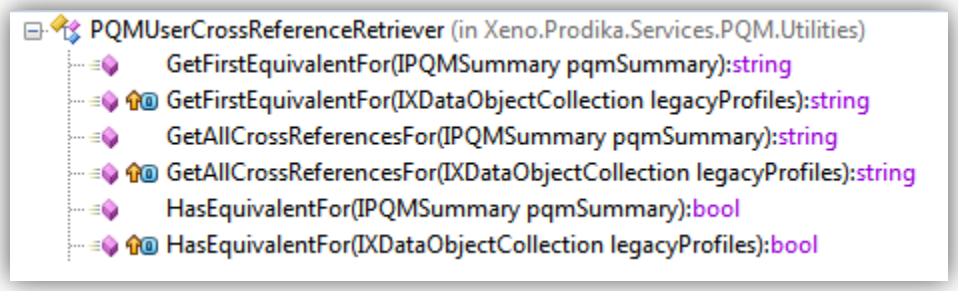
PQMWorkflowTagEvaluator

Provides methods to help determine which workflow status a PQM Item is in. The EnumPQMWorkflowTagID enum can be passed in as a parameter, to check for predefined workflow tags, such as IsReleased.



PQMUserCrossReferenceRetriever

Provides methods to retrieve the list of (a User's preferred) cross references assigned to a PQMItem.



PQMPermissionManager

Available via the PQMItemService, provides permission-related information for PQM