Microsoft Dynamics® AX 2012

Enterprise Portal Development Cookbook

White paper

This white paper provides guidance on Enterprise Portal application development and customization.

www.microsoft.com/dynamics/ax

Mey Meenakshisundaram, Principal Group Program Manager
Anees Ansari, Program Manager

Send suggestions and comments about this document to adocs@microsoft.com. Please include the title with your feedback.

Table of Contents

Introduction 6
User interface 7
List pages 7
Details pages 11
Architecture 12
Page processing 13
Web Parts 15
Application Object Tree elements 21
Development tools and prerequisites 23
Before you get started 23
MorphX 23
Visual Studio 23
SharePoint Products and Technologies 24
Data sources and data sets 25
Data sources 25
Properties 25
Methods 25
Data sets 29
Methods 30
Aggregation 32
Framework controls 35
AxDataSource 35
Properties 35
Filtering 36
Methods 36
Events 37
AxForm 38
Properties 38
Events 39
EPFormAction enumeration 40
Requiring record context 41
Table inheritance and row creation 41
Layout controls 42
AxMultiSection 42

AxSection ... 42
AxMultiColumn ... 43
AxColumn ... 43
AxGroup ... 44
AxGridView ... 46
Properties ... 46
Events ... 47
Examples ... 49
AxHierarchicalGridView ... 52
AxFilter ... 53
AxContextMenu ... 54
BoundField controls ... 54
Examples ... 55
Summary fields ... 56
Template fields ... 56
Action Pane and toolbars ... 59
AxActionPane ... 59
AxToolbar ... 61
AxLookup ... 67
Multi-selection ... 69
PreLoad ... 72
LookupCacheScope ... 73
AxDatePicker ... 74
AxDateTimeHelper ... 74
FactBox controls ... 75
AxPartContentArea ... 75
AxFormPart ... 75
AxInfoPart ... 75
CueGroupPartControl ... 75
AxContentPanel ... 76
AxPopup ... 78
AxModalPrompt ... 82
AxReportViewer ... 83
AxEnhancedPreview ... 83
List and details pages ... 84

Model-driven list pages ... 84
Details pages ... 86
Advanced list page development ... 88
List page interaction classes ... 88
Secondary list pages ... 89
Multi-selection ... 92
General guidelines ... 92
Preview pane title ... 94
Converting Microsoft Dynamics AX 2009 client list pages ... 95
Converting Microsoft Dynamics AX 2009 secondary list pages ... 96
Report list pages ... 97
Advanced details page development ... 98
Creating a new project ... 98
Editing an existing project ... 98
Creating a new user control ... 99
Editing an existing user control ... 99
Removing a web control ... 99
General guidelines ... 100
Auto Action Pane buttons ... 101
Embedding user controls ... 102
Modal window properties ... 103
Modal windows – Interaction patterns ... 104
Modal windows – Programming ... 110
CommonControls.ControlHelper ... 113
General development ... 114
Sessions ... 114
Windows Server AppFabric ... 115
Context ... 116
Data ... 119
Metadata ... 121
Changing metadata at run time ... 122
Pages ... 124
Page definition properties ... 124
Events ... 124
Page titles ... 127

Custom filters 128
Creating custom filters in model-driven list pages 128
Creating an advanced filter via a user control 129
Creating a custom filter in a non-modeled list page by modifying the data set 130
Creating a custom filter in a non-modeled list page without modifying the data set 131
Custom lookup 132
Creating a custom lookup in managed code 132
Creating a custom lookup in X++ 133
Getting the selected row 133
AutoPostback when OK is clicked 134
Proxies 135
Creating a new proxy project 135
Proxy deployment 137
General guidelines 137
Formatting 138
Validation 139
Error handling 140
Debugging 141
Infolog 144
ViewState 145
WebLink and AxUrlMenuItem 146
HTMLUtilities 148
Adding JavaScript 149
Images 149
ASP.NET chart controls 150
Web modules 152
ShowParentModule 152
Cues 153
SharePoint integration 155
Enterprise Search 155
Themes 158
Tips 158
Workflow 159

Introduction

Enterprise Portal for Microsoft Dynamics AX is an organization's window into its business. It enables users to directly access relevant business information, and to collaborate and conduct business transactions with Microsoft Dynamics AX. Enterprise Portal also serves as a web platform. It enables developers to web-enable and customize existing business applications in Microsoft Dynamics AX, or to create new business applications.

Enterprise Portal expands the reach of your enterprise resource planning (ERP) solutions. It provides a simple and intuitive user interface that helps improve productivity. Developers can easily customize and extend Enterprise Portal to quickly respond to changing business needs. The framework enables rapid application development, so that you can jump-start your solutions. Simplified installation, deployment, and management tasks help system administrators be more efficient.

The primary help resources for developing Enterprise Portal are available at Enterprise Portal for Microsoft Dynamics AX on MSDN.

User interface

Pages in Enterprise Portal provide the main mechanism for organizing content. Enterprise Portal uses a set of standard page types throughout the application to enforce a consistent layout, making the application intuitive for users. Two main page types are the list page and the details page.

List pages

List pages in Enterprise Portal display a list of records. This section describes the various user interface components of a list page.

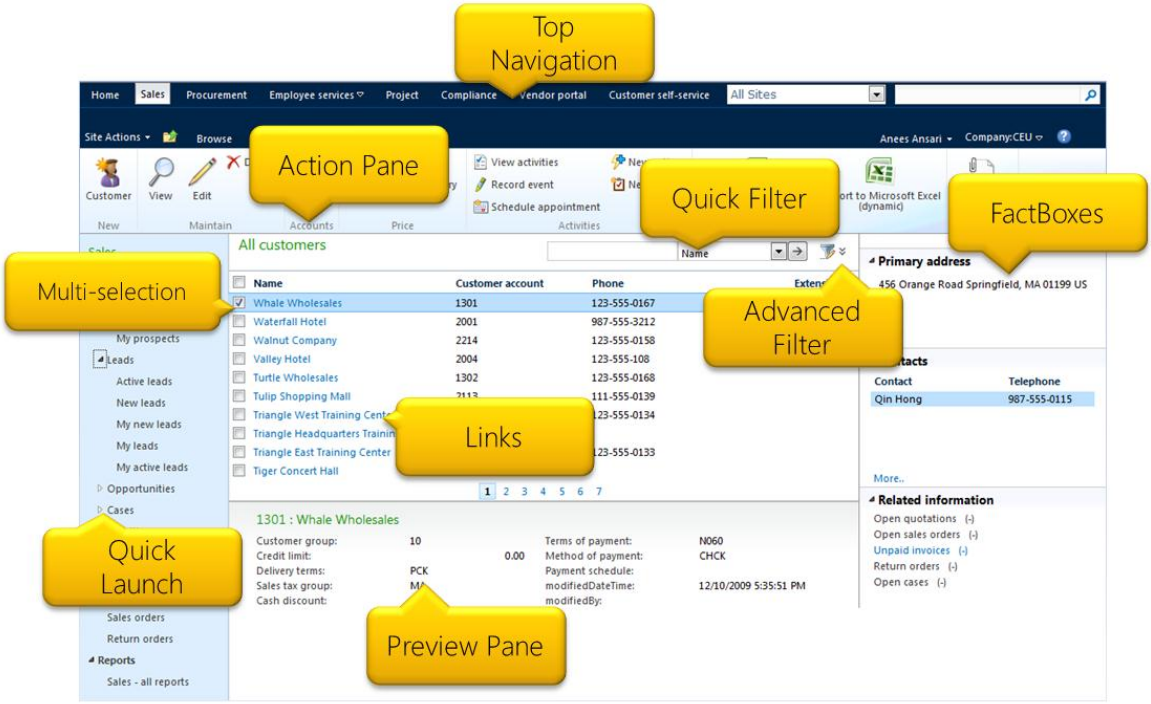


Figure 1 An Enterprise Portal list page

Top navigation bar The top navigation bar is a set of links at the top of the page. Users can use this to navigate between the various modules that are visible to them, such as Sales and Procurement. Each link in the top navigation bar points to the default page of the corresponding module.

Figure 2 A top navigation bar

Action Pane The Action Pane uses the familiar Microsoft Office ribbon layout. It categorizes a set of buttons on contextual tabs and in button groups. This enhances simplicity and discoverability, because the actions that are available vary, based on the permissions that the user has been granted. For example, in the following figure:

- The buttons are first grouped on various tabs, such as Customer, Sell, and General.
- The Customer tab is currently selected, and its contents are visible.
- On the tab, the buttons are further grouped into button groups, such as New, Maintain, Accounts, and Price.

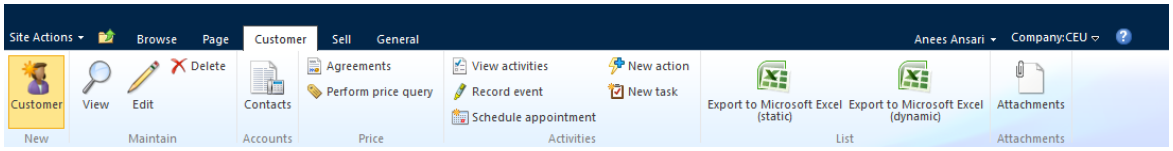


Figure 3 An Action Pane

Quick Launch Quick Launch is a set of links on the left side of the page. Users can use this to navigate to the various areas and pages within a module. In Microsoft Dynamics AX 2012, support was added for displaying a hierarchy, so that you can show multiple levels of links. By default, Quick Launch contains links to both primary and secondary list pages. Secondary list pages apply additional conditions to filter and display a subset of the data displayed on the primary list pages.

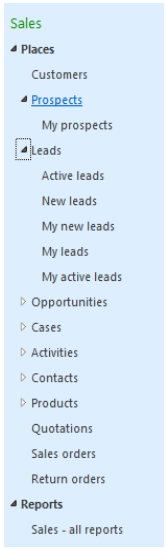


Figure 4 Quick Launch

Quick filter A quick filter is a control used to apply a simple filter condition on a list page. The quick filter provides a menu of the fields displayed in the grid. A user can select a field and specify a value to search for in that field. The quick filter was added in Microsoft Dynamics AX 2012 to help reduce the number of clicks required to find data.

Figure 5 A quick filter

Advanced filter An advanced filter is a control used to apply complex filter conditions on a list page. The advanced filter enables filtering for all fields, not just the fields displayed in the grid. It also provides a range of operators and enables multiple simple filter conditions to be combined to create a complex filter condition.

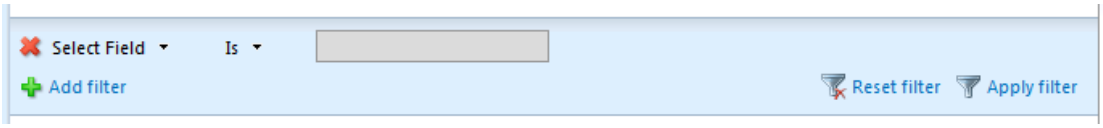
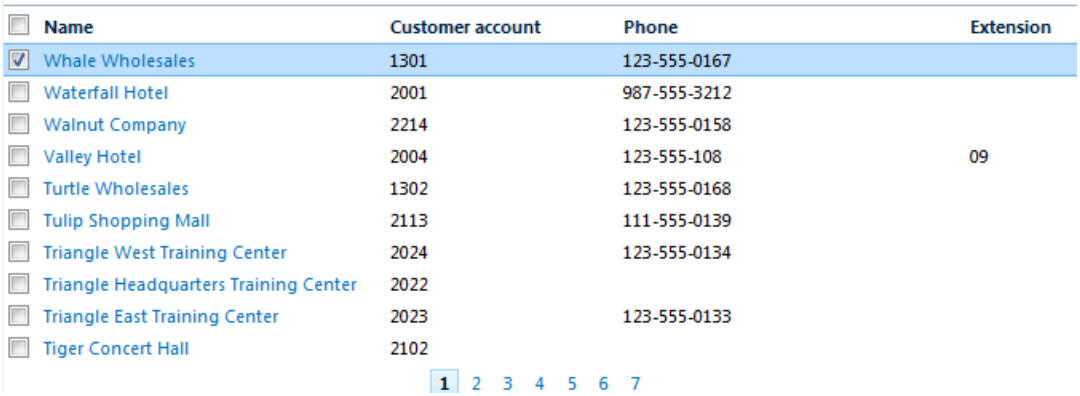


Figure 6 An advanced filter

List The list displays the records on the page in a grid. It provides the ability to sort, select, and page through the records. The conditions applied by using the quick and advanced filters filter the records displayed in the list.



Name	Customer account	Phone	Extension
Whale Wholesales	1301	123-555-0167	
Waterfall Hotel	2001	987-555-3212	
Walnut Company	2214	123-555-0158	
Valley Hotel	2004	123-555-108	09
Turtle Wholesales	1302	123-555-0168	
Tulip Shopping Mall	2113	111-555-0139	
Triangle West Training Center	2024	123-555-0134	
Triangle Headquarters Training Center	2022		
Triangle East Training Center	2023	123-555-0133	
Tiger Concert Hall	2102		

1 2 3 4 5 6 7

Figure 7 A list

Preview pane The preview pane is the area below the grid and is used to display extended information for the record that is currently selected in the list. The preview pane is automatically updated, based on the selected record. The preview pane is a great way to show users more data on the same page, without making them navigate away.

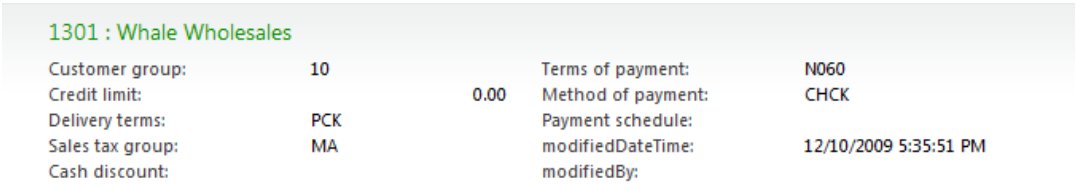


Figure 8 A preview pane

FactBox FactBoxes are displayed on the right side of the page. A FactBox is used to display information related to the record that is currently selected in the list. Like the preview pane, FactBoxes are automatically updated, based on the selected record. They help show the user more data on the same page.

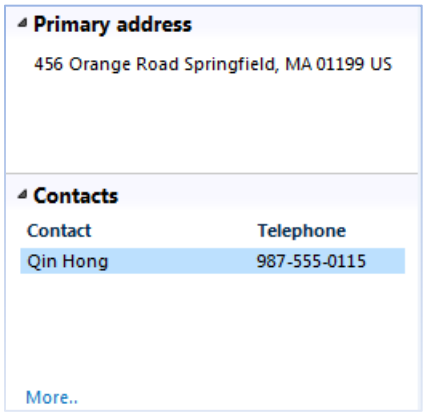


Figure 9 A FactBox

Details pages

To see the details about a particular record, a user can open the corresponding details page. In most cases, this is done by clicking the link that is available in the list. As the name implies, the details page displays detailed information about a specific selected record.

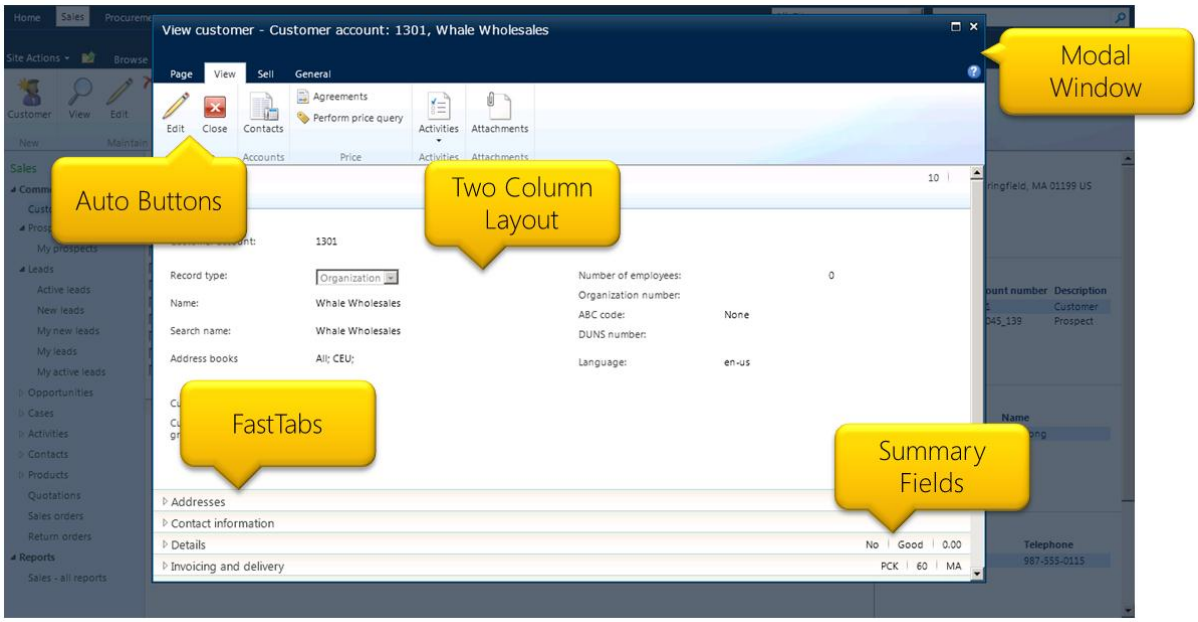


Figure 10 An Enterprise Portal details page

Modal window All details pages in Enterprise Portal open as a modal window. This helps preserve the context of the list page, such as the page number in the list and applied filters, while the user views or edits the details of a specific record. Notice that the two-column layout makes optimum use of the screen space.

Auto buttons The framework automatically adds buttons for common actions, such as Save and Close, and Close.

FastTabs FastTabs are the collapsible sections of the page that are used to display field groups. They help organize information by giving users the ability to show or hide data.

Summary fields Summary fields display the values of the most important fields on a FastTab directly in the header of the FastTab. This lets users view the important data at a glance, without expanding the FastTab.

Architecture

Enterprise Portal brings the best of Microsoft Dynamics AX, ASP.NET, and Microsoft SharePoint technologies together to provide a rich web-based business application. It combines the rich content and collaboration functionality of SharePoint with the structured business data of Microsoft Dynamics AX. It also leverages the power and flexibility of ASP.NET and Microsoft Visual Studio, so that developers can quickly build and extend Enterprise Portal.

Developers can use Microsoft MorphX to leverage the rich programming model to define data access and business logic in Microsoft Dynamics AX. They can build web user controls and define the web user interface elements by using Visual Studio. The web controls can contain Microsoft Dynamics AX components, such as AxGridView, as well as standard ASP.NET controls, such as TextBox. The data access and business logic defined in Microsoft Dynamics AX are exposed to the web user controls through data binding, data/metadata APIs, and proxy classes.

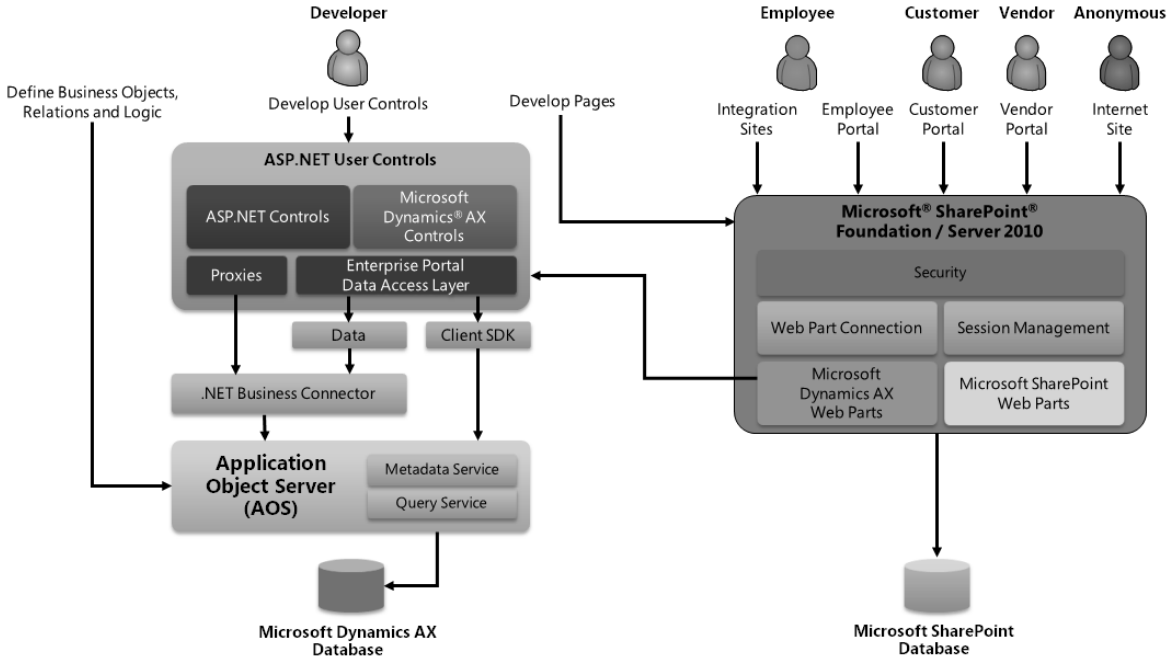


Figure 11 The Enterprise Portal architecture

Enterprise Portal uses the Web Part page framework from SharePoint. This lets developers build webpages that enable easy customization and personalization. It also makes it easy to integrate content, collaborate, and use third-party applications. The webpages can contain both Microsoft Dynamics AX Web Parts and SharePoint Web Parts.

The Microsoft Dynamics AX Web Parts present information and expose functionality from Microsoft Dynamics AX. The User Control Web Part can host any ASP.NET web user control, including the Enterprise Portal web user controls, and can connect to Microsoft Dynamics AX through the Enterprise Portal framework. SharePoint Web Parts can be used to fulfill other content and collaboration needs.

Page processing

The first step in developing or customizing an application in Enterprise Portal is to understand the interactions between the user's browser on the client and Enterprise Portal on the server when the user accesses Enterprise Portal.

The following sequence of interactions occurs when a user accesses an Enterprise Portal page:

1. The user opens the browser on his or her machine, and navigates to Enterprise Portal.
2. The browser establishes a connection with the Internet Information Services (IIS) web server.
3. Based on the authentication mode that is enabled, IIS authenticates the user.
4. After the user is authenticated, SharePoint checks the user's permission to access the site.
5. If the user is authorized, the request is passed to the SharePoint module.
6. SharePoint gets the Web Part page data from the SharePoint content database (by using the virtual path provider) or from the file system. This data contains information such as the page layout, the master page, the Web Parts used, and their properties.
7. SharePoint processes the page. It creates and initializes the Web Parts on the page, applying any properties and personalization data.
8. To display the top navigation bar, Quick Launch, and Action Pane, a custom navigation provider is used to get the information from Microsoft Dynamics AX (modules, menus, submenus, and menu items).
9. Enterprise Portal initializes the Microsoft Dynamics AX Web Parts, and it also initializes a web session with the framework through .NET Business Connector to Application Object Server (AOS).
10. The web framework checks for Microsoft Dynamics AX authorization and then calls the appropriate web handlers in the web framework to process the Enterprise Portal objects that the Web Part points to.
11. The User Control Web Part runs the web user control that it references. The web user control connects to Microsoft Dynamics AX through .NET Business Connector and renders the HTML to the Web Part.
12. The webpage assembles all the HTML returned by all the Web Parts and renders the page to the user's browser.
13. The Enterprise Portal web session ends.

As you can see in this sequence, AOS processes all the business logic and data retrieval, ASP.NET processes the user interface elements, and SharePoint handles the overall page layout and personalization.

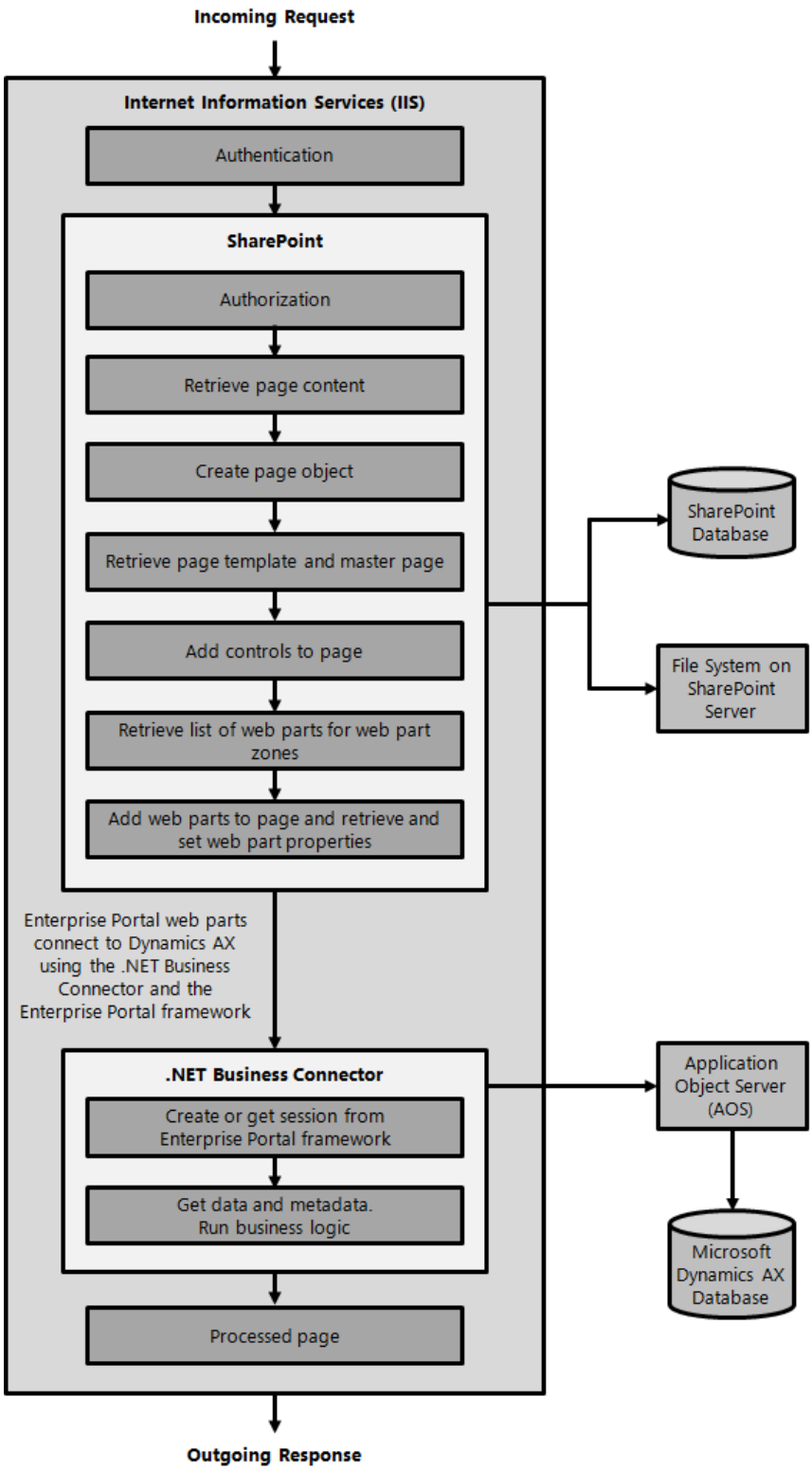


Figure 12 Enterprise Portal page processing

Web Parts

Web Parts are reusable SharePoint components that generate HTML and provide the foundation for the modular presentation of data. Web Parts are easily integrated to assemble a webpage, and they support customization and personalization. Enterprise Portal comes with a standard set of Web Parts that expose business data from Microsoft Dynamics AX.

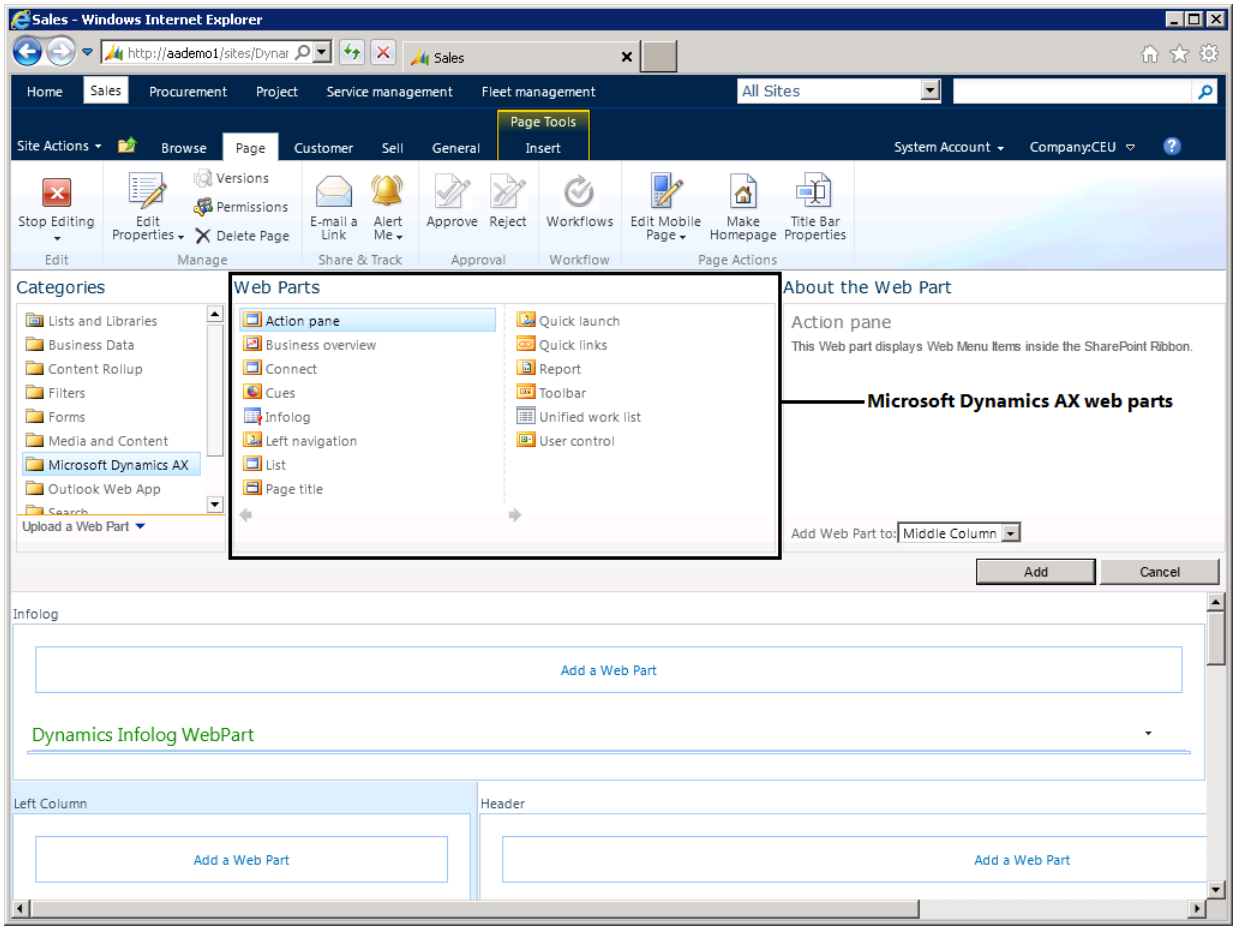


Figure 13 Adding Microsoft Dynamics AX Web Parts to a page

The following is a list of Microsoft Dynamics AX Web Parts that are included with Enterprise Portal.

Action Pane Used to display the Action Pane on the page. The Action Pane is similar to the SharePoint ribbon. It points to a web menu in the Application Object Tree (AOT), and displays buttons on tabs and in groups, improving their discoverability. The AxActionPane control can be used in a web control as an alternative to using the Action Pane Web Part.

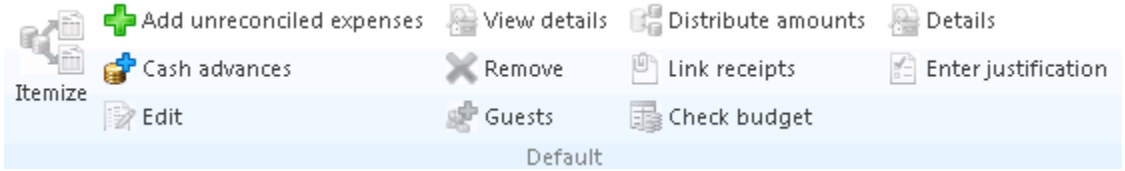


Figure 14 An Action Pane Web Part

Business Overview Used to display Business Intelligence (BI) information, such as key performance indicators (KPIs) and other analytical data, on Role Center pages.

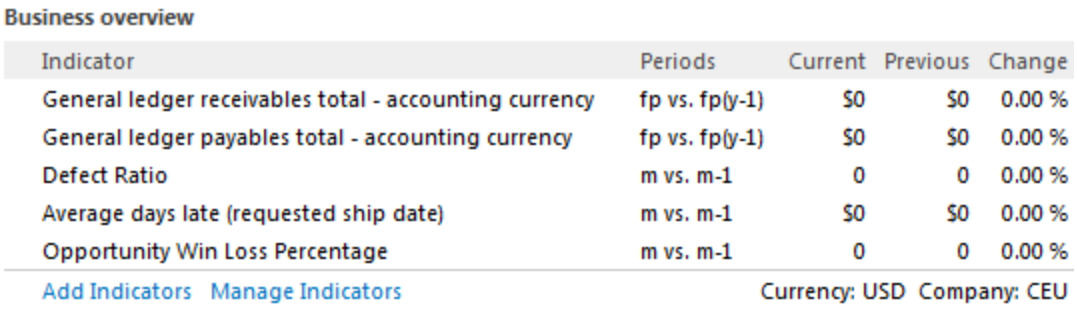


Figure 15 A Business Overview Web Part

Connect Used to display links to information from the Microsoft Dynamics AX community. This is typically used on Role Center pages.

Figure 16 A Connect Web Part

Cues Used to display numeric information, such as active opportunities and new leads, as a paper stack. It is generally added to Role Center pages and points to a cue group in the AOT.

Figure 17 A Cues Web Part

Infolog Used to display Microsoft Dynamics AX Infolog messages on the webpage. When a new Web Part page is created by using Enterprise Portal page templates, the Infolog Web Part is automatically added to the top of the page in the Infolog Web Part zone. Any error, warning, or information message that Microsoft Dynamics AX generates is automatically displayed by the Infolog Web Part. If you need to display some information from your web user control in the Infolog Web Part, you need to send the message through the C# proxy class for the X++ Infolog object.

Figure 18 An Infolog Web Part

Left Navigation Used instead of the Quick Launch Web Part if you want to display page-specific navigation instead of module-specific navigation. It points to a web menu in the AOT.

List Used to display the contents of a model-driven list page. When you deploy a model-driven list page to Enterprise Portal, the page template automatically adds the List Web Part to the middle column zone of the page. It points to the Display menu item for the model-driven list page form.

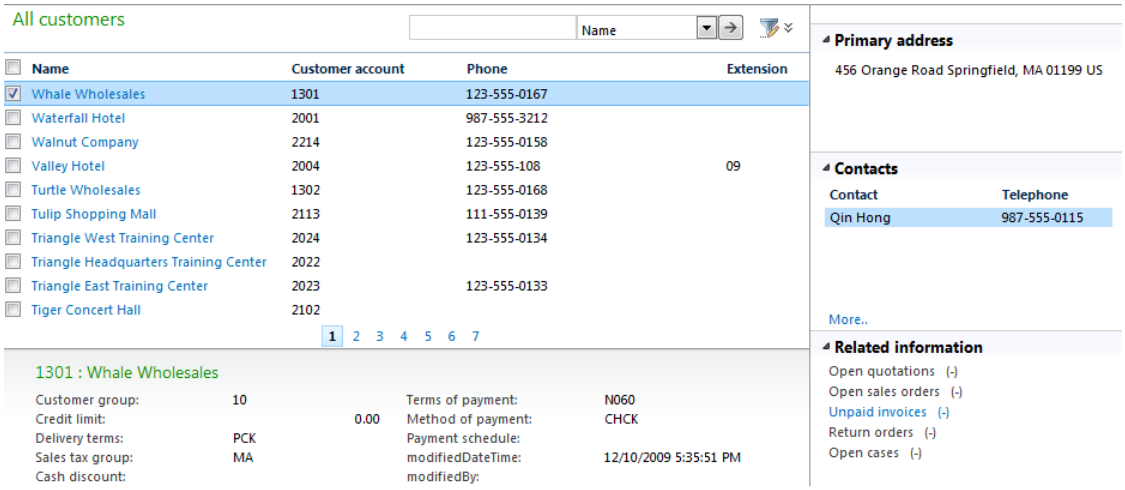


Figure 19 A page with a List Web Part

Page Title Used to display the page title. When you create a new Web Part page, the Page Title Web Part is automatically added to the Title Bar zone. By default, the Page Title Web Part displays the text specified in the PageTitle property of the Page Definition node in the AOT. If no page definition exists, the page name is displayed. You can override this default behavior and make this Web Part get the title from another Web Part on the webpage by using a Web Part connection. For example, you're developing a list page, and you want to display some record information, such as the customer account number and name, as the page title. In this case, you can connect the User Control Web Part that displays the grid to the Page Title Web Part. Then, when you select a different record in the customer list, the page title changes to display the account number and name of the currently selected customer.

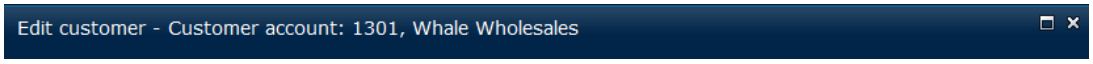


Figure 20 A Page Title Web Part

Quick Launch Used to display module-specific navigation links on the left side of the page in Enterprise Portal. When you create a new Web Part page, the Quick Launch Web Part is automatically added to the Left Column zone, if the template that you select has this zone. The Quick Launch Web Part displays the web menu specified

in the QuickLaunch property of the corresponding web module in the AOT. All the pages in a given web module (subsite) display the same left navigation.

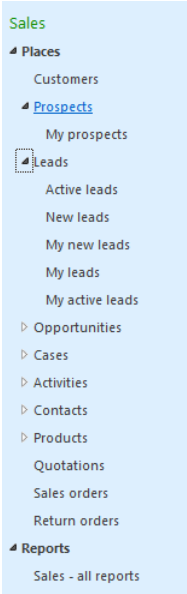


Figure 21 A Quick Launch Web Part

Quick Links Used to display a collection of links to frequently used menu items and external websites. It is generally added to Role Center pages.

Figure 22 A Quick Links Web Part

Report Used to display Microsoft SQL Server Reporting Services (SSRS) reports for Microsoft Dynamics AX.

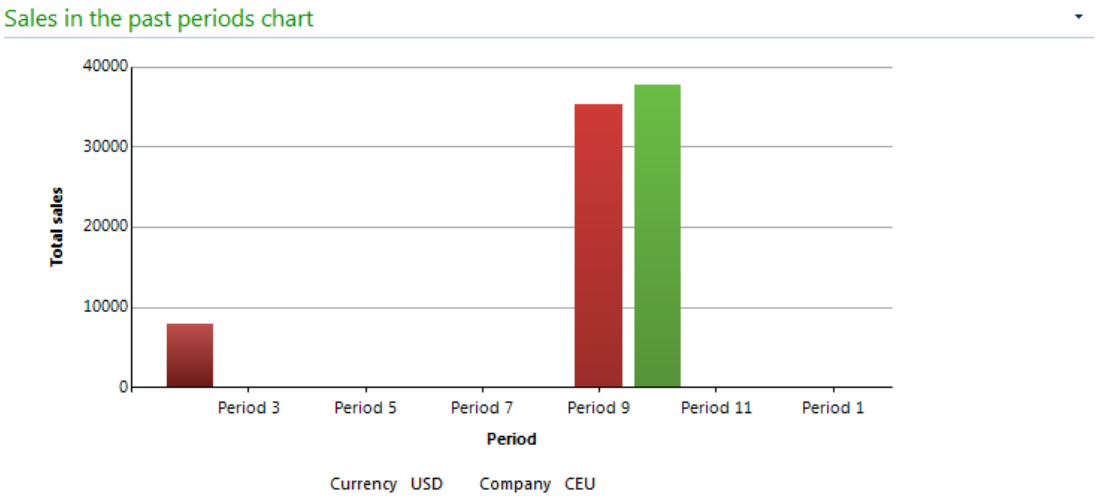


Figure 23 A Report Web Part

Toolbar Used to display a toolbar on the page. When you use the Action Pane Web Part, the buttons are displayed in the ribbon area at the top of the page. When you use the Toolbar Web Part, the buttons are displayed in the same location as the Toolbar Web Part. For example, you can place Add, Edit, and Remove buttons for a grid control right above this Web Part. The Toolbar Web Part points to a web menu in the AOT. The AxToolbar control can be used in a web user control as an alternative to using the Toolbar Web Part.

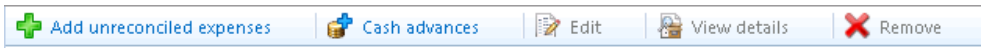


Figure 24 A Toolbar Web Part

Unified Work List Used to display workflow actions, alert notifications, and tasks. It is generally added to Role Center pages.

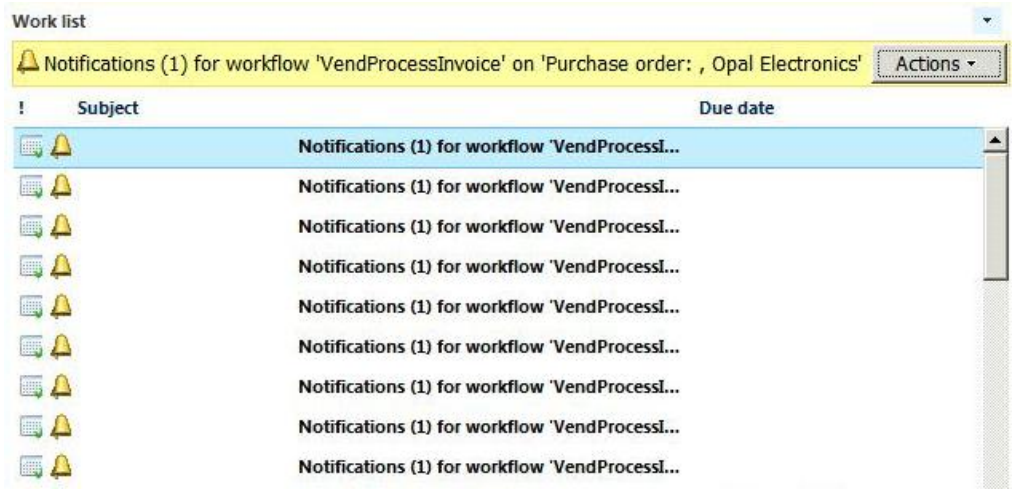


Figure 25 A Unified Work List Web Part

User Control Used to host any ASP.NET control, including the Microsoft Dynamics AX web controls that you develop. It points to a managed web content Item that identifies the web user control. The User Control Web Part can pass record context information to other Web Parts, consume it from them, or both. To configure the behavior, set the role of the User Control Web Part to Provider, Consumer, or Both, and then connect it to other Web Parts. User Control Web Parts automatically use AJAX, so that they can update the content that they display without requiring a refresh of the entire page.

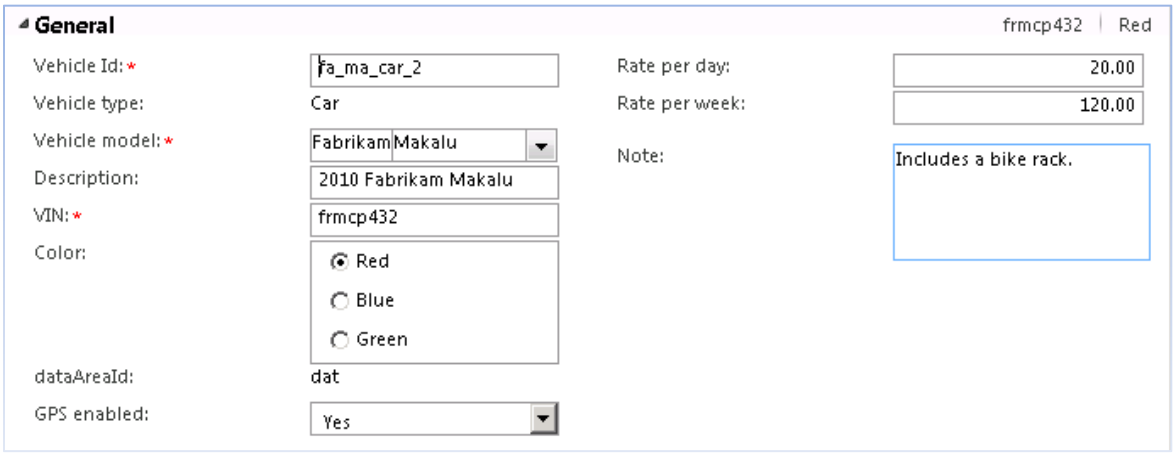


Figure 26 A User Control Web Part

For more information about the Web Parts in Microsoft Dynamics AX, see Web Part Types on MSDN.

Application Object Tree elements

The AOT contains several elements, such as forms, classes, and tables. Each type of element represents an object that serves a certain purpose in developing an application. This section describes the elements used for Enterprise Portal applications.

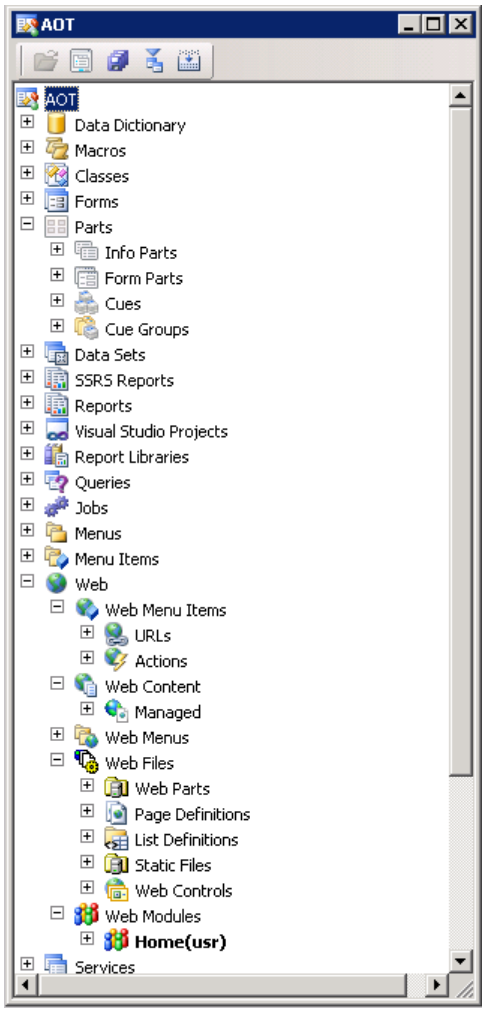


Figure 27 Enterprise Portal AOT elements

Tables (Data Dictionary\Tables) Represent table objects that contain the data for the system.

Classes Represent class objects that contain business logic. The interaction classes used for model-driven list pages are defined here. (See the "Model-driven list pages" section for details.)

Forms These are the main objects that represent windows and dialog boxes for the Microsoft Dynamics AX client. They serve as overall containers for the other user interface elements. Model-driven list pages in Enterprise Portal are also defined by using form objects. (See the "Model-driven list pages" section for details.)

Info Parts (Parts\Info Parts) A *part* is a control used to show a group of fields. *Info parts* are modeled parts that use metadata to define the fields and layout. This generic definition enables them to be rendered on both the Microsoft Dynamics AX client and in Enterprise Portal. Model-driven list pages can reference info parts and dis-

play them in the Preview Pane or the FactBox area. To display info parts in web user controls, you can use the AxInfoPart and AxPartContentArea framework controls.

Form Parts (Parts\Form Parts) Used when you need more flexibility to display the data than an info part provides. A form part simply links to a form and a managed content item. The linked form determines how the form part is rendered on the Microsoft Dynamics AX client, and the managed content item (web user control) determines how it is rendered in Enterprise Portal. Model-driven list pages can reference form parts and display them in the Preview pane or the FactBox area. To display form parts in web user controls, use the AxFormPart and AxPartContentArea framework controls.

Data Sets Used to provide access to Microsoft Dynamics AX tables and define data access logic. Data sets represent a collection of data that is usually presented in tabular form. Enterprise Portal uses the AxDataSource framework control to access data. The AxDataSource control connects to a data set and uses it to interact with data.

Queries Represent modeled query objects that are used to retrieve data for forms and info parts.

Display Menu Items (Menu Items\Display) Forms that define model-driven list pages are linked to the corresponding page definitions used in Enterprise Portal through display menu Items. Model-driven list pages also use display menu Items to reference info parts, form parts, and details pages. (See the "Model-driven list pages" and "Details pages" sections for details.)

URL Web Menu Items (Web\Web Menu Items\URLs) Represent menu item objects that contain links to the Web Part pages in Enterprise Portal. For example, a URL web menu item button can open a page so that you can add or edit a record. The URL web menu item objects have metadata settings that specify whether the linked page opens in a modal window. The configuration keys can also be applied to these.

Action Web Menu Items (Web\Web Menu Items\Actions) Represent menu item objects that cause an action. For example, an action web menu item button can delete the records that are selected in the grid. Typically, these are linked to class objects that are run when these menu items are invoked.

Managed Web Content (Web\Web Content\Managed) Represent the managed content items corresponding to the web user controls. The configuration keys can be applied to these. You can also use these to pass parameters to the page that contains the web user control.

Web Menus (Web\Web Menus) Represent a set of URL or action web menu items. These are used to define the Quick Launch hierarchy (navigation) for a web module. They are also used to define the contents (tabs, button groups, and buttons) of Action Panes and toolbars.

Page Definitions (Web\Web Files\Page Definitions) Contain the XML definitions of the SharePoint Web Part pages. These also contain properties that indicate the web module that the page belongs to, as well the page title.

Static Files (Web\Web Files\Static Files) These are static files objects, such as SharePoint Web Part page templates, SharePoint master pages, script files, and cascading style sheets.

Web Controls (Web\Web Files\Web Controls) These are the web user controls that can be deployed to Enterprise Portal.

Web Modules (Web\Web Modules) Define the SharePoint sites and subsites in Enterprise Portal (for example, Sales, Financial, or Employee Services). These also provide properties to define Quick Launch and other aspects of each web module.

Resources These are resource objects, such as images.

Development tools and prerequisites

For Enterprise Portal applications, you develop the presentation tier components in Visual Studio, and you design the web pages by using tools in SharePoint. The AOT controls all metadata for Enterprise Portal. It stores all the controls and pages that you develop in Visual Studio and SharePoint. It also stores other supporting files, definitions, and features under the Web node.

Before you get started

Note that, in many examples in this document, we assume that you are familiar with basic Microsoft Dynamics AX development, and that your machine has Enterprise Portal, the Microsoft Dynamics AX client (including the Development Workspace), and Visual Studio 2010 with the Microsoft Dynamics AX Visual Studio Tools.

We also assume that you have Microsoft Dynamics AX development objects, such as tables, queries, and data sets, available to you in the AOT. If you do not have these, you can use the Fleet Management sample application, which is available as a free download from Microsoft. The download includes detailed information and the steps to import the Fleet Management application into Microsoft Dynamics AX 2012.

Download Fleet Management sample application

You also need to run most programs (Visual Studio, MorphX, and so on) with administrative privileges.

MorphX

MorphX is the integrated development environment (IDE) in Microsoft Dynamics AX. The data and business tier components for Enterprise Portal are developed by using MorphX. MorphX is also used for the following tasks:

- Defining the navigation elements
- Storing unified metadata and files
- Importing and deploying controls, pages, and list definitions
- Generating proxies

Visual Studio

Visual Studio is used for developing and debugging web user controls. The Visual Studio Add-in for Enterprise Portal provides a project and control templates to speed the development process. Visual Studio also provides the following:

- An easy way to add new controls to the AOT
- Tools to import controls or style sheets from the AOT
- The ability to work with proxies

The Enterprise Portal framework provides various APIs to enable data and metadata access.

SharePoint Products and Technologies

SharePoint is used to develop Web Part pages and lists. It is also used to edit master pages, which contain the common elements for all the pages in a site. With a browser, you can use the Create or Edit Page tools of SharePoint to design your Web Part page. You can also use Microsoft SharePoint Designer to create or edit both Web Part pages and master pages.

Data sources and data sets

Enterprise Portal has a built-in set of framework controls that you can use to access, display, and manipulate Microsoft Dynamics AX data. It also includes APIs for programmatic access to data and metadata, a C# proxy class framework for accessing Microsoft Dynamics AX business logic and helper classes, and a Visual Studio add-in for adding files to, and importing files from, the AOT. This section reviews and discusses common development techniques used in Enterprise Portal development.

Data sources

A data source acts as the provider of data. It references tables, views, or queries to retrieve the data.

Properties

ChangeGroupMode ChangeGroupMode is a new property that is added at the root of the data source node. By default, the property is set to None, and each data source commits independently of the other data sources that are inner-joined or outer-joined. If you set the value of this property to ImplicitInnerOuter, all the data sources that are inner-joined or outer-joined work as one unit. Therefore, they all either commit successfully or roll back.

InsertAtEnd, InsertIfEmpty For Enterprise Portal, you should set these to No.

Allow* Remember to set these properties (AllowCreate, AllowDelete, and so on) based on your requirements.

ValidTime* Remember to set these properties for date-effective data sources.

LinkType For Enterprise Portal, only InnerJoin, OuterJoin, and Active are supported.

Methods

init Typically, the init method is used to add ranges, set a temporary table mode, and perform similar tasks, as shown in the following examples.

Example 1: Adding a range

```
public void init()
{
    super();

    // Need a workaround to get count(*) more efficiently using the record id during insert
    if (element.args().parmEnumType() == enumnum(EPFormAction) &&
        element.args().parmEnum() == EPFormAction::CreateMode)
    {
        smmOpportunityTable_ds.query().dataSourceName(tablestr(smmOpportunityTable)).addRange(
            fieldnum(smmOpportunityTable,RecId)).value(SysQuery::value(0));
    }
}
```

Example 2: Adding multiple ranges

```
public void init()
{
    super();

    qbrName = this.query().dataSourceTable(tablenum(InventDimCombination)).addRange(
        fieldnum(InventDimCombination,Name));

    qbrName.status(RangeStatus::Hidden);

    qbrItem = this.query().dataSourceTable(tablenum(InventDimCombination)).addRange(
        fieldnum(InventDimCombination,ItemId));

    qbrItem.status(RangeStatus::Hidden);
}
```

Example 3: Setting the temporary table mode

```
public void init()
{
    super();

    salesTable.setTmp();
}
```

initValue The initValue method is used to initialize values.

```
public void initValue()
{
    super();

    // Is form called with a record
    if (element.args() && element.args().record())
    {
        callerRecord = element.args().record();

        // Init opportunity from lead record
        if (callerRecord.TableId == tablenum(smmLeadTable))
        {
            smmCreateEntity::initFromCommon(callerRecord, smmOpportunityTable);
            info(DirParty::toolTip(smmOpportunityTable));
        }
    }
}
```

validateWrite The validateWrite method is used for validations that are specific to the data set. (Common table-level validation can be performed at the table level.)

```
public boolean validateWrite()
{
    #VendRequest
    boolean ret;
    ret = super();

    if  (ret && !VendTable::find(vendrequest.AccountNum))
        ret = checkFailed(strfmt("@SYS301682",vendrequest.AccountNum));

    if  (ret && element.args().managedContentItemName() == #VendRequestStatusChangeAddEdit &&
            DirPerson::find(vendRequest.ContactParty) == null)
        ret = checkFailed(strfmt("@SYS301683",vendrequest.ContactParty));|

    return ret;
}
```

write The write method is called when the user inserts or updates a record.

```
public void write()
{
    if (purchLine.RecId)
    {
        purchLine.InventDimId = InventDim::findOrCreate(inventDim).InventDimId;
        purchLine_ds.setCurrentInventDim();

        mapLines.insert(purchLine.RecId, purchLine);
    }

    super();
}
```

research The research method runs the database search query again. It preserves the filtering and sorting information. Use research instead of executeQuery if the query has not changed, and you only want to refresh the form to show new records or remove records that were deleted.

executeQuery The executeQuery method runs the database search query as defined at initialization. It does not preserve the filtering and sorting information. Use this method if the query has changed, and you want to re-run the query to show different records—for example, from a modified filter condition.

<field>.dataSetLookup The dataSetLookup method is defined at the field level and is used to customize the field lookup. This does not appear in the override method list, but you should be able to manually create this method, which will act as an override.

```
void dataSetLookup(SysDataSetLookup sysDataSetLookup)
{
    List          list;
    Query         query;

    args = new Args();
    list = new List(Types::String);
    list.addEnd(fieldstr(ProjTable, ProjId));
    list.addEnd(fieldstr(ProjTable, Name));
    list.addEnd(fieldstr(ProjTable, Status));
    sysDataSetLookup.parmLookupFields(list);
    sysDataSetLookup.parmSelectField('ProjId');

    // Call query
    query = projTimeSheet_ds.getProjectIDQuery();

    // Pass the query to SysDataSetLookup so it result is rendered in the lookup page.
    sysDataSetLookup.parmQuery(query);
}
```

<field>.modified The modified method is defined at the field level and is used to perform an action when the value of a field is changed.

```
public void modified()
{
    super();

    purchLine.modifyItemDim(inventDim,
        fieldnum(InventDim,ConfigId),
        InventTable::itemProductDimensionGroup(purchLine.ItemId));
    purchLine_ds.setCurrent();
}
```

Data sets

Data sets are used to define the data access logic. A data set is a collection of data that is usually presented in tabular form. The data set brings the familiar data and programming model known from Microsoft Dynamics AX forms together with ASP.NET data binding. In addition, the data set offers an extensive X++ programming model for validating and manipulating the data when it is created, read, updated, or deleted in Enterprise Portal. The AxDatasource control is used to access data sets to display and manipulate data from any ASP.NET control that supports data binding.

Data sets are created by using MorphX, and they can contain one or more data sources that are joined together. The data sources can point to a table or a view in Microsoft Dynamics AX. The data from joined data sources or from parent/child data sets is surfaced by using dynamic data set views (DataSetView). With a view-based interface, tables are accessed through dynamic data set views rather than directly.

- **Inner and outer joins** Used to display data from multiple tables as a single data source. With inner or outer joins, only one SQL query is issued against the database. You can access inner-joined or outer-joined tables through only one view, which has the same name as the primary data source.
- **Active join** Used to display parent/child data. With active joins, one query is issued for each unit involved in the active join. With active-joined data sources, two or more views are available: one with the same name as the parent data source, and one or more with the same name as the child data sources. The child data source contains records related only to the current active record in the parent data source.
- **Exist and Not exist joins** These are not supported by data sets.

Each data set view can contain zero or more records, depending on the data. Each data set view also has a corresponding special view, which contains just the current active record. This view has the same name as the original view, but with _Current appended.

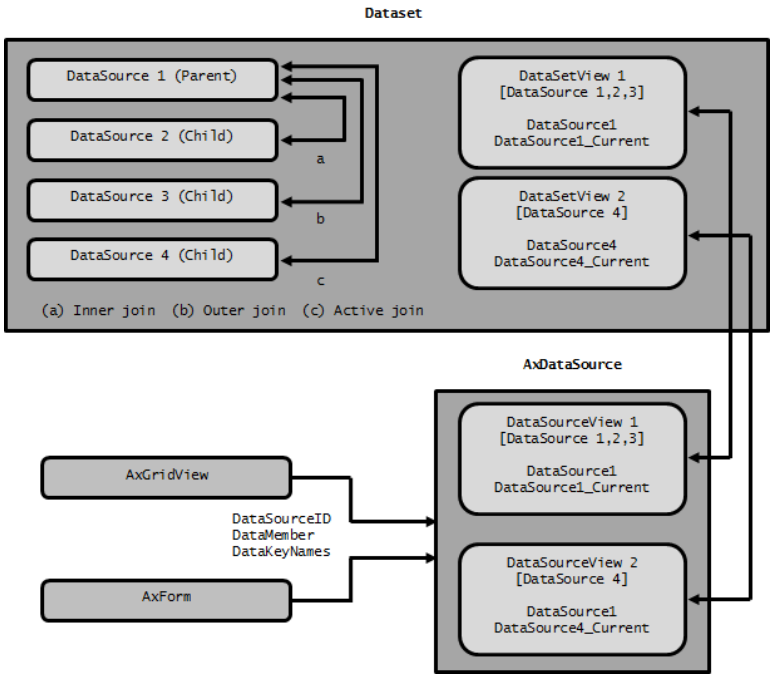


Figure 28 Enterprise Portal data set views

Figure 28 shows the data set views inside a data set and the data binding. DataSource 1, DataSource 2, and DataSource 3 are joined by using inner and outer joins; therefore, they are surfaced by using a single data set view (DataSetView1). DataSource 4 is active joined and is therefore surfaced as a separate data set view (DataSetView2).

Methods

init The init method is called when the data set is initialized. It is activated immediately after new and creates the run-time image of the data set. Typical uses of init include adding ranges to restrict the data, checking the arguments and parameters that are passed, and initializing and modifying variables and queries.

run The run method is called immediately after init. Typical uses of run include conditionally making fields visible or hidden, changing the access level on fields, and modifying queries.

pack and unpack The pack/unpack pattern is used to save and store the state of an object, which you can later re-instantiate. The pack method is called after the data set is run. A typical use of pack is to persist a variable that is used in the data set between postbacks. The unpack method is called if a data set was previously packed and then accessed. If a data set was previously packed, init and run aren't called. Instead, only unpack is executed.

```
public container pack()
{
    return [#CurrentVersion, #CurrentList] + [super()];
}

public boolean unpack(container _pack)
{
    Integer     version = conpeek(_pack,1);
    container   base;
    boolean     ret;

    switch (version)
    {
        case #CurrentVersion    :   [version, #CurrentList, base] = _pack;
                                    ret = super(base);
                                    break;
        default: super(_pack);
    }

    return ret;
}
```

Example 1: Getting the external arguments and checking for record context

```
public void init()
{
    super();
    EPPersonalize::find(curuserid());
    if (element.args() && element.args().dataset() == tablenum(EmplTable))
    {
        callerEmplTable = element.args().record();
        emplId = callerEmplTable.EmplId;
    }
}
```

Example 2: Passing a parameter from managed code to X++

```
protected void Page_Init(object sender, EventArgs e)
{
    this.AxDataSource1.CreatingDataSetRun +=
        new EventHandler<CreatingDataSetRunEventArgs>(AxDataSource1_CreatingDataSetRun);
}

void AxDataSource1_CreatingDataSetRun(object sender, CreatingDataSetRunEventArgs e)
{
    e.DataSetRunArgs.parm = "4000";
}
```

In the AOT, override or add a method in the data set, and use element.args().parm() to get the parameter.

```
public void executeQuery()
{
    QueryBuildRange custRange;
    custRange = SysQuery::findOrCreateRange(this.query().dataSourceNo(1),
        fieldnum(CustTable, AccountNum));

    custRange.value(element.args().parm());
    super();
}
```

Example 3: Passing an enum from managed code to X++

```
protected void Page_Init(object sender, EventArgs e)
{
    this.AxDataSource1.CreatingDataSetRun +=
        new EventHandler<CreatingDataSetRunEventArgs>(AxDataSource1_CreatingDataSetRun);
}

void AxDataSource1_CreatingDataSetRun(object sender, CreatingDataSetRunEventArgs e)
{
    e.DataSetRunArgs.parmEnumType = EnumMetadata.EnumNum(this.AxSession, "EPFormAction");
    e.DataSetRunArgs.parmEnum(2);
}
```

In the AOT, override or add a method in the data set, and use element.args().parmEnumType() and element.args.parmEnum() to get the enum type and value.

```
public void init()
{
    super();
    if (element.args().parmEnumType() == enumnum(EPFormAction) &&
        element.args().parmEnum() == EPFormAction::CreateMode)
    {
        //Do something
    }
}
```

Aggregation

Enterprise Portal provides two simple and efficient ways to display aggregated values in a simple grid interface on a webpage.

Modifying the query by using code

If a data set has tables that are simply joined, you can create a query to group the fields and aggregate the values in those fields throughout the code.

Example

1. In the AOT, create a new data set named MyDataSet, with two tables (CustTable and SalesTable) that are inner joined.
2. Override the init method of the CustTable data source, and add the following code.

```
public void init()
{
    Query query;
    QueryBuildDataSource qB;

    super();

    query = CustTable_ds.query();
    qB = query.dataSourceTable(tableNum(CustTable));
    qB.addGroupByField(fieldNum(CustTable, AccountNum));
}
```

3. Override the init method of the SalesTable data source, and add the following code.

```
public void init()
{
    Query query;
    QueryBuildDataSource qB;

    super();

    query = CustTable_ds.query();
    qB = query.dataSourceTable(tableNum(SalesTable));
    qB.addSelectionField(fieldNum(SalesTable, SalesId), SelectionField::Count);
}
```

4. Create a new user control.
5. Add an AxDataSource control named AxDataSource1 that points to MyDataSet.
6. Add an AxGridView control that uses AxDataSource1 for data binding.

7. Add the fields that were used in aggregation to the grid control, as shown in the following code.

```
<dynamics:AxDataSource ID="AxDataSource1" runat="server"
    DataSetName="MyDataSet" ProviderView="CustTable">
</dynamics:AxDataSource>
<dynamics:AxGridView ID="AxGridView1" runat="server" BodyHeight=""
    DataKeyNames="RecId,SalesTable!RecId" DataMember="CustTable"
    DataSetCachingKey="093056ec-5081-421b-b5e0-5f2fa69b65cf"
    DataSourceID="AxDataSource1" EnableModelValidation="True">
    <Columns>
        <dynamics:AxBoundField DataField="AccountNum" DataSet="MyDataSet"
            DataSetView="CustTable" SortExpression="AccountNum">
        </dynamics:AxBoundField>
        <dynamics:AxBoundField DataField="SalesTable!SalesId" DataSet="MyDataSet"
            DataSetView="CustTable" SortExpression="SalesTable!SalesId">
        </dynamics:AxBoundField>
    </Columns>
</dynamics:AxGridView>
```

8. Add the user control to a page. When it is rendered, it should be similar to the following figure. The actual account numbers and number of sales orders will vary, based on the data in your system.

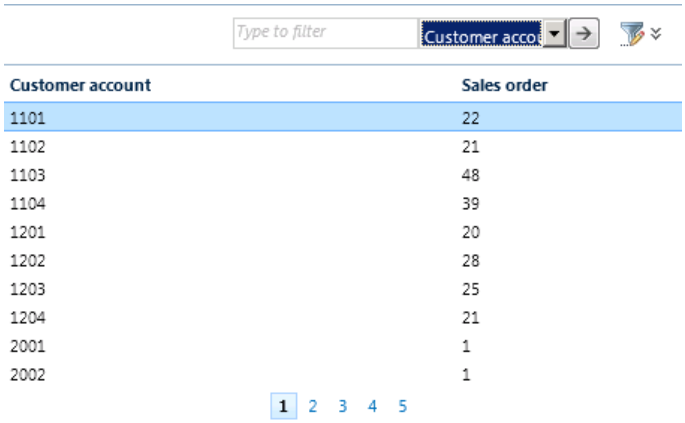


Figure 29 Aggregation in data sets by modifying a query through code

Tip Set the AutoSearch and AutoQuery properties on the data set data sources if the query is modified through code or through ranges that are added before the values are used. This will be beneficial mainly for create and edit scenarios where the query is always modified or, for create scenarios where the result set is not needed.

Modeling the query in the AOT

Another way to achieve aggregation is by modeling the query in the AOT.

Example

1. In the AOT, create a new query named MyQuery.
2. Add the CustTable as the data source.
3. Under the CustTable data source, add the SalesTable as a data source.

4. Under the SalesTable data source, add the relation CustTable.AccountNum == SalesTable.CustAccount.
5. Add CustTable.AccountNum to the Group By node of the CustTable data source.
6. Add AccountNum to the Fields node of the CustTable data source.
7. Right-click the Fields node of the SalesTable data source, point to New, and then click the aggregation that you want (AVG, SUM, COUNT, MIN, or MAX).
8. Set the field that you added in the preceding step to SalesId.
9. Create a new data set named MyDataSet.
10. Set the query property of MyDataSet to MyQuery.
11. Follow steps 4 through 8 from the "Modifying the query by using code" approach to create a user control and display the aggregate data in a grid.

Framework controls

AxDataSource

The AxDataSource control extends the ASP.NET Framework's DataSourceControl to provide a declarative and data store–independent way to read and write data from Microsoft Dynamics AX. Data sets that you create in the AOT are exposed to ASP.NET through the AxDataSource control. ASP.NET data-bound user interface controls can be associated with the AxDataSource control through the DataSourceID property. This lets a developer who does not have any specific domain knowledge of Microsoft Dynamics AX connect to and access data from Microsoft Dynamics AX, and bind the data to the control.

The data source control is simply a container for one or more uniquely named views of type AxDataSourceView. The AxDataSourceView class extends the Microsoft .NET Framework's DataSourceView, and implements the functionality to read and write data. A data-bound control can identify the set of enabled capabilities from the properties of AxDataSourceView, and use it to show, hide, enable, or disable the user interface components. AxDataSourceView maps to the data set view. The AxDataSource control will automatically create AxDataSourceView objects based on the data set that it references. The number of objects created depends on the data sources and the joins that are defined for the data set. You can use the DataMember property of a data-bound control to select a particular view.

The AxDataSource control also supports filtering records within and across other AxDataSource controls and data source views. When you set the active record on the data source view within an AxDataSource control, all the child data source views are also filtered, based on the active record. You can filter across AxDataSource controls through record context. In record context, one AxDataSource control acts as the provider of the context, and one or more AxDataSource controls act as consumers. One AxDataSource can act as both a provider and a consumer. When the active record changes on the provider AxDataSource control, the record context is passed to other consuming AxDataSource controls, and they apply that filter as well. You use the Role and ProviderView properties of AxDataSource for this purpose. One web user control can contain any number of AxDataSource controls; however, only one of them can have the Role property set to Provider. Any number of them can have the Role property set to Consumer.

Properties

DataSetName The DataSetName property points to a data set in the AOT.

Role Make sure that only one AxDataSource control in your user control has its Role property set to Provider or ProviderConsumer. If the user control contains other user controls, only one data source control should play the Provider role across all of them. If its role is Provider or ProviderConsumer, the ProviderView property must be set.

ProviderView The ProviderView property specifies the view in the data set that is used for the current context.

EnableViewState and PersistState Always set the EnableViewState and PersistState properties to True. (These are the defaults. Do not change them.)

<Bound Control>.DataMember Data-bound controls like AxGridView use the AxDataSource control for data binding. The DataMember property of these data-bound controls is used to select a particular view. The AxDataSourceView maps to the DataSetView. So, for each dynamic DataSetView that is based on the number of data sources and the type of join, the AxDataSource control has an equivalent number of AxDataSourceViews that any data-bound control can bind to.

Filtering

The AxDataSource control also supports filtering records within and across AxDataSource controls and DataSourceViews. Within an AxDataSource control, when the active record on the DataSourceView is set, all the child DataSourceViews are also filtered, based on the active record. Filtering across AxDataSource controls is done through record context. In this case, one AxDataSource control acts as the context provider, and zero or more AxDataSource controls act as the consumer. One AxDataSource control can act as both a provider and a consumer. When the active record changes on the provider AxDataSource control, the record context is passed to other consuming AxDataSource controls, and they apply that filter as well. The Role and ProviderView properties of the AxDataSource control are used for this purpose. One web user control can contain any number of AxDataSource controls; however, only one of them can have the Role property set to Provider. Any number of them can have the Role property set to Consumer.

Methods

GetDataSet The GetDataSet method on the AxDataSource control returns the data set that it binds to. The DataSetRun property provides the run-time instance of the data set. You can use the AxpataObjectAdapter to call the methods that are defined in the data set.

```
this.AxDataSource1.GetDataSet().DataSetRun.AxaptaObjectAdapter.Call("method1");
```

DataSetView.GetCurrent You can use the DataSetView object to get the rows in a DataSetView. The GetCurrent method returns the current row.

```
private DataSetViewRow CurrentRow
{
    get
    {
        try
        {
            DataSetView dsv = this.CustomersDS.GetDataSet().DataSetViews["CustTable"];
            return (dsv == null) ? null : dsv.GetCurrent();
        ...
}
```

DataSetViewRow.GetField You can use the DataSetViewRow.GetField object to get the value of a field in a row.

```
private SomeOtherMethod()
{
    ...
    using (IAxaptaRecordAdapter custTable = this.CurrentRow.GetRecord())
    {
        customerName = custTable.GetField("Name").ToString();
    }
    ..
}
```

DataSourceView.Select You can use the DataSourceView.Select object for asynchronous data retrieval.

```
private void GetData()
{
    DataSourceViewSelectCallback callback = new DataSourceViewSelectCallback(PrintData);
    AxDataSource1.GetDataSourceView("CustTable").Select(DataSourceSelectArguments.Empty,callback);
}

private void PrintData(IEnumerable data)
{
    IEnumerator i = data.GetEnumerator();
    while (i.MoveNext())
    {
        Response.Write(DataBinder.Eval(i.Current, "AccountNum").ToString() + "<BR/>");
    }
}
```

Events

CreatingDataSetRun This event occurs when a data set is created or unpacked.

```
void ActivityDS_CreatingDataSetRun(object sender, CreatingDataSetRunEventArgs e)
{
    e.DataSetRunArgs.parmEnumType = ENUMID_SMMACTIVITYCATEGORY;
    e.DataSetRunArgs.parmEnum((int)this.category);
}
```

AxForm

The AxForm control lets you view, create, and update a single record. It displays a single record from a data source in a form layout. It is a data-bound control with built-in data modification capabilities. When you use AxForm with the declarative AxDataSource control, you can easily configure it to display and modify data without needing to write any code.

Properties

DataSourceID Used to specify the AxDataSource control that you want to use.

DataKeyNames Used to specify a comma-separated list of one or more unique fields.

DataMember Used to specify the current view to bind to. Set this to <DataSetView>_Current.

DefaultMode Used to specify the default mode of the form. By default, this is set to ReadOnly.

AutoGenerate*Buttons Used to indicate which buttons (Save and Close, Close, and so on) the framework should automatically add to the Action Pane. This is used in combination with the DefaultMode property.

ShowButtonsInActionPane Used to enable or disable the addition of framework buttons to the Action Pane as specified by the AutoGenerate*Buttons properties.

UpdateOnPostBack Used to update the record cursors on postback. This is generally set to false. If you need to access the field values that are entered in the control before the record is persisted in a postback, set this to true. For example, to change the lookup of a second control based on the value entered in the first control, you need to set this property.

EnableViewState Used to enable view state. By default, this is set to true and should not be changed.

AllowPrompting and DerivedControlMode Used for supertype and subtype tables. If your table is not a polymorphic table, leave the default settings.

Events

Init Occurs when the server control is initialized, which is the first step in its life cycle.

ItemCommand Occurs when a button is clicked in the form control.

```
private void ActivityForm_ItemCommand(object sender, DetailsViewCommandEventArgs e)
{
    if (e.CommandName == "Cancel")
    {
        this.DataSourceView.CancelEdit();
        this.RedirectToPreviousPage();
    }
}
```

ItemCreated Occurs after a new item has been created by the form control.

ItemInserted Occurs after a new item has been inserted by the form control.

ItemInserting Occurs before the new item has been inserted by the form control.

ItemUpdated Occurs after an item has been updated by the form control.

```
private void ActivityForm_ItemUpdated(object sender, DetailsViewUpdatedEventArgs e)
{
    //check if successful, then redirect to appropriate page
    if (e.Exception == null)
    {
        this.RedirectToPreviousPage();
    }
}
```

ItemUpdating Occurs before an item has been updated by the form control.

Load Occurs when the server control is loaded into the System.Web.UI.Page object.

PreRender Occurs after the System.Web.UI.Control object is loaded but prior to rendering.

EPFormAction enumeration

Use the EPFormAction enumeration if you want to use the same AxForm control for different modes. On the web-managed content item corresponding to the user control, set the EnumTypeParameter property to EPFormAction. If required, you can also set the EnumParameter property to pass in a value. Then, in your user control, you can check the value of the parameter and take the required action.

Note To use the EPForm action in your user control, you need to use proxies. This is described in a later section.

```
/// <summary>
/// Returns the current form mode.
/// </summary>
ApplicationProxy.EPFormAction FormMode
{
    get
    {
        return (ApplicationProxy.EPFormAction)Convert.ToInt16(
            this.Page.Request.QueryString.Get("mode"));
    }
}

protected void Page_Init(object sender, EventArgs e)
{
    this.SetupMode();


}

private void SetupMode()
{
    switch (this.FormMode)
    {
        case ApplicationProxy.EPFormAction.EditMode:
            if (this.PageDefinition != null)
            {
                this.PageDefinition.pageTitle = Labels.GetLabel("@SYS131104");
            }

            this.formVendRequestProfileEdit.DefaultMode = DetailsViewMode.Edit;
            this.formVendRequestProfileEdit.AutoGenerateEditButton = true;
            break;

            default:
                this.formVendRequestProfileEdit.DefaultMode = DetailsViewMode.ReadOnly;
                break;
    }
}
```

Requiring record context

If you don't want to display anything when no record context is passed, you can derive your user control from AxBaseUserControl and override the RequiresExternalContext property. In the following example, we expect a record context except in create mode.

```
protected override bool RequiresExternalContext
{
        get
        {
            if (this.FormAction == AppProxy.EPFormAction.CreateMode)
            {
                return false;
            }
            else
            {
                return true;
            }
        }
    }
```

Table inheritance and row creation

If the tables that you use are part of an inheritance hierarchy, the framework automatically prompts you to select a specific type when you create a record or row. If you want to avoid this, and instead want to create a specific type of record, you can do that by specifying the data source name through either a query string or code.

If you have separate menu items for each subtype creation, set the query string parameter to pass the data source name as the instance type.

If you want to do this by using code, override the AxRowCreating event. Use FormViewPloymorphicRowCreateEventArgs to set one or more subtypes. If more than one subtype is set, the user will be prompted to select one.

```
void CaseCreateForm_AxRowCreating(object sender, FormViewRowCreateEventArgs e)
    {
        FormViewPolymorphicRowCreateEventArgs args = ((FormViewPolymorphicRowCreateEventArgs)e);
        args.ConcreteTypes.Clear();

        args.ConcreteTypes.Add(args.ViewMetadata.DataSources["CaseDetailBase_CaseDetail"]);
    }
```

If the instance type is sent neither through a query string nor through code, the framework will automatically prompt the user to select one, as indicated earlier.

Layout controls

This section covers the layout controls used in Enterprise Portal.

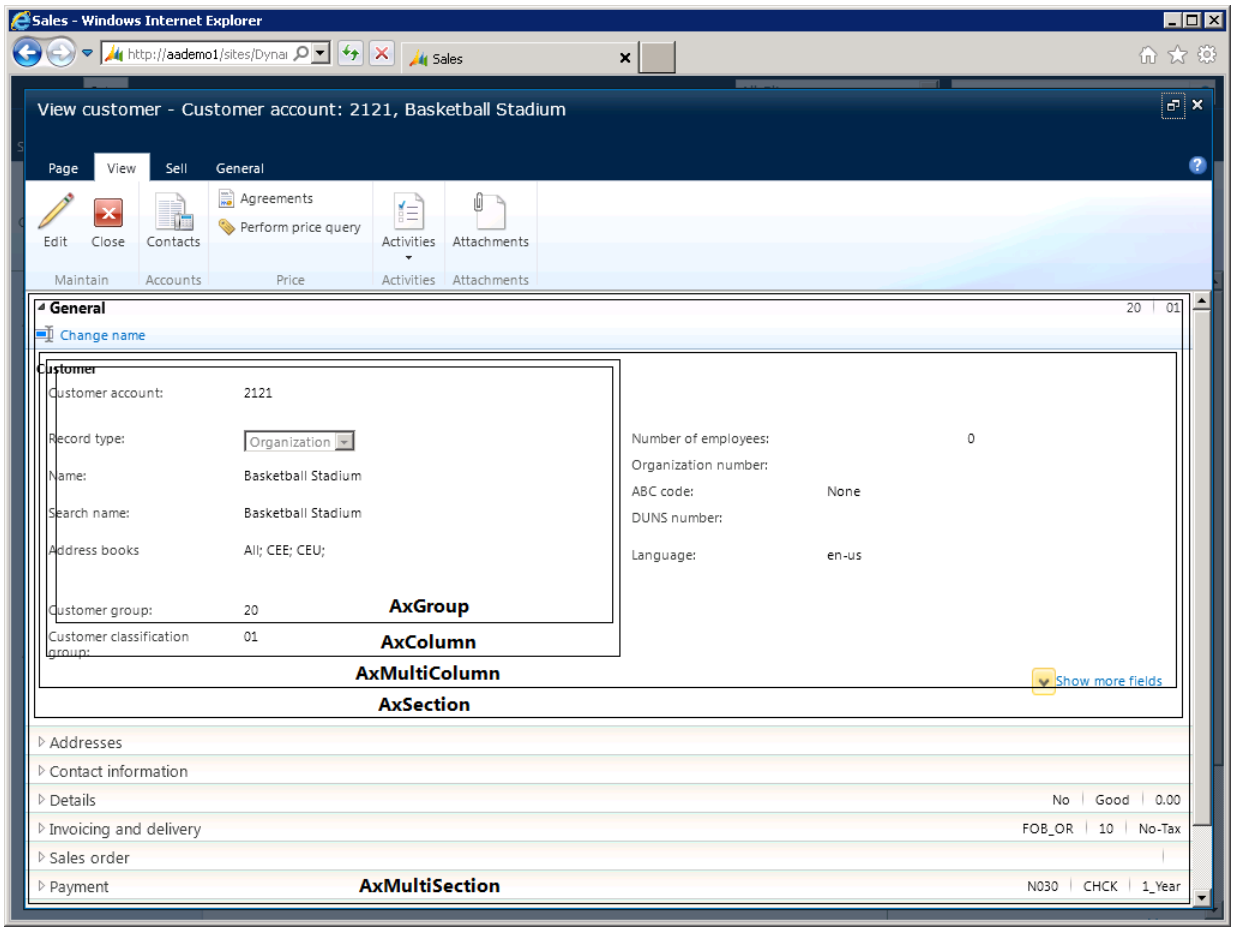


Figure 30 Enterprise Portal details page with section, column, and group controls

AxMultiSection

The AxMultiSection control acts as a container for a collection of AxSection controls. All AxSection controls within an AxMultiSection control are rendered as a stacked set of rows. Users can expand or collapse any of the rows. You can configure AxMultiSection so that only one section is expanded at one time. In this mode, expanding a section causes it to become active. Any previously expanded section is collapsed. To enable this behavior, set the ActiveMode property to true. You can then use the ActiveSectionIndex property to get or set the active section.

AxSection

The AxSection control is a generic control container. Any control can be placed in an AxSection control. Each AxSection control includes a header that contains the title of the section, and a button that lets the user expand or collapse the section. AxSection provides properties for displaying or hiding the header and border. Through the events exposed by the AxSection control, you can write code that runs when the section is expanded or collapsed. The AxSection control can be placed only within an AxMultiSection control.

AxMultiColumn

The AxMultiColumn control acts as a container for a collection of AxColumn controls. The AxColumn controls within an AxMultiColumn control are rendered as a series of columns. The AxMultiColumn control makes it easy to achieve a multi-column layout and make optimum use of the screen space. The AxMultiColumn control is usually placed within an AxSection control.

AxColumn

The AxColumn control is a generic control container. Any control can be a child of AxColumn. The AxColumn control can be placed only within an AxMultiColumn control.

AxGroup

The AxGroup control contains the bound field collection that displays the record information. You place AxGroup controls inside AxSection or AxColumn controls.

Example 1: Instead of writing table markup (TR and TD elements), you should use AxMultiColumn and AxColumn controls to achieve a multi-column layout. Use of these controls also automatically handles the alignment of columns across sections. The following example shows how of you can modify your existing controls to take advantage of the new AxMultiColumn and AxColumn controls.

Before

```
<dynamics:AxSection ID="AxSectionDefaults" runat="server" Expanded="true"
    Caption="<%$AxLabel SYS116474 %>">
    <table width="100%">
        <tr valign="top">
            <td>
                <dynamics:AxGroup ID="AxFieldGroupDefaultsPosting" runat="server"
                    Caption="<%$AxLabel SYS116475 %>">
                    <Fields>
                        <dynamics:AxBoundField DataSet="TrvExpTableNew" DataSetView="TrvExpTable"
                            DataField="Accountnum" LookupButtonDisplaySettings="Always"
                            OnLookup="Account_Lookup" />
                    </Fields>
                </dynamics:AxGroup>
            </td>
            <td rowspan="2" width="50%">
                <dynamics:AxGroup ID="AxFieldGroupDefaultsDimensions" runat="server"
                    Caption="<%$AxLabel SYS5951 %>">
                    <Fields>
                        <dynamics:AxBoundFieldGroup DataSet="TrvExpTableNew"
                            DataSetView="TrvExpTable" FieldGroup="DimensionName" />
                    </Fields>
                </dynamics:AxGroup>
            </td>
        </tr>
        <tr>
            <td>
                <dynamics:AxGroup ID="AxFieldGroupDefaultsProject" runat="server"
                    Caption="<%$AxLabel SYS4534 %>">
                    <Fields>
                        <dynamics:AxBoundField DataSet="TrvExpTableNew" DataSetView="TrvExpTable"
                            DataField="ProjId" />
                    </Fields>
                </dynamics:AxGroup>
            </td>
        </tr>
    </table>
</dynamics:AxSection>
```

After

```
<dynamics:AxSection ID="AxSectionDefaults" runat="server" Expanded="true"
    Caption="<%$AxLabel SYS116474 %>">
    <dynamics:AxMultiColumn>
        <dynamics:AxColumn>
            <dynamics:AxGroup ID="AxFieldGroupDefaultsPosting" runat="server"
                Caption="<%$AxLabel SYS116475 %>">
                <Fields>
                    <dynamics:AxBoundField DataSet="TrvExpTableNew" DataSetView="TrvExpTable"
                        DataField="Accountnum" LookupButtonDisplaySettings="Always"
                        OnLookup="Account_Lookup" />
                </Fields>
            </dynamics:AxGroup>
            <dynamics:AxGroup ID="AxFieldGroupDefaultsProject" runat="server"
                Caption="<%$AxLabel SYS4534 %>">
                <Fields>
                    <dynamics:AxBoundField DataSet="TrvExpTableNew" DataSetView="TrvExpTable"
                        DataField="ProjId" />
                </Fields>
            </dynamics:AxGroup>
        </dynamics:AxColumn>
        <dynamics:AxColumn>
            <dynamics:AxGroup ID="AxFieldGroupDefaultsDimensions" runat="server"
                Caption="<%$AxLabel SYS5951 %>">
                <Fields>
                    <dynamics:AxBoundFieldGroup DataSet="TrvExpTableNew" DataSetView="TrvExpTable"
                        FieldGroup="DimensionName" />
                </Fields>
            </dynamics:AxGroup>
        </dynamics:AxColumn>
    </dynamics:AxMultiColumn>
</dynamics:AxSection>
```

Example 2: Hiding fields in an AxGroup control

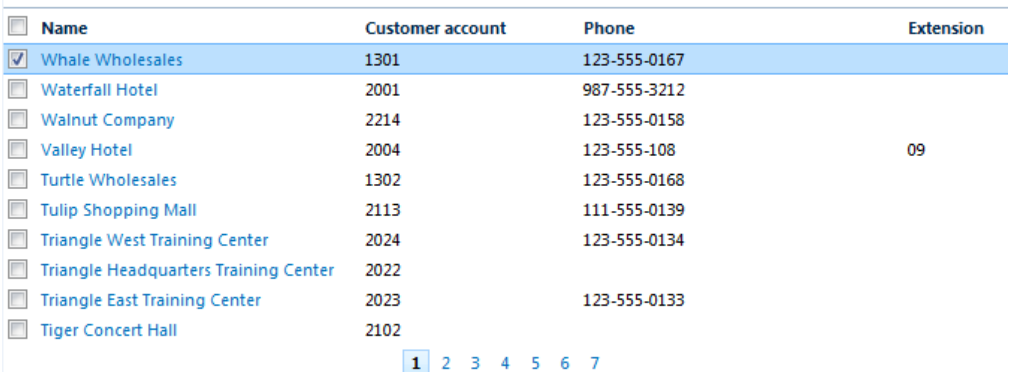
```
protected void AxFrmEmployee_PreRender(object sender, EventArgs e)
{
    AxForm frm = this.axFrmEmployee;

    for (int i = 0; i < this.AxGroup1.Fields.Count; i++)
    {
        AxBoundField fld = this.AxGroup1.Fields[i] as AxBoundField;
        if ((fld != null) && (fld.Metadata.Name == "TitleId"))
        {
            fld.Visible = (frm.CurrentMode == DetailsViewMode.ReadOnly) ? false : true;
        }
    }
}
```

AxGridView

The AxGridView control displays the values from a data source in a table format. Each column represents a field, and each row represents a record. The AxGridView control extends the ASP.NET GridView control to provide selection, grouping, expansion rows, filtering, context menus, and other enhanced capabilities.

AxGridView also includes built-in data modification capabilities. By using AxGridView with the declarative AxDataSource, you can easily configure and modify data without writing code. AxGridView also has many properties, methods, and events that you can easily customize with application-specific user interface logic.



Name	Customer account	Phone	Extension
Whale Wholesales	1301	123-555-0167	
Waterfall Hotel	2001	987-555-3212	
Walnut Company	2214	123-555-0158	
Valley Hotel	2004	123-555-108	09
Turtle Wholesales	1302	123-555-0168	
Tulip Shopping Mall	2113	111-555-0139	
Triangle West Training Center	2024	123-555-0134	
Triangle Headquarters Training Center	2022		
Triangle East Training Center	2023	123-555-0133	
Tiger Concert Hall	2102		

1 2 3 4 5 6 7

Figure 31 An AxGridView control

Properties

DataMember Used to identify the table or data set that the grid will bind to.

DataSourceID Used to identify the AxDataSource control that will be used to get the data.

DataKeyNames DataKeyNames must point to one unique field or a set of unique fields in the DataSetView.

Allow* AllowInsert, AllowEdit, and AllowDelete properties control the create, read, update, and delete operations on the grid. AllowMarking is used to enable multiselection in the grid.

FixedGridHeight Setting the FixedGridHeight property will make the grid occupy the height required to display the number of rows that are defined in the page size, even if there is less or no data to display.

Events

DataBinding Occurs when the server control binds to a data source.

```
void GridView_TrvExpTrans_DataBinding(object sender, EventArgs e)
{
    if (this.CurrentRow != null)
    {
        foreach (DataControlField dcf in this.GridView_TrvExpTrans.Columns)
        {
            AxBoundField axbf = dcf as AxBoundField;
            if (axbf != null && (axbf.DataField == "AmountCurr" || axbf.DataField == "TransDate"))
            {
                // we only postback in edit mode, no auto-postback for insert mode
                axbf.AutoPostBack = !this.CurrentRow.IsNew;
            }
        }
    }
}
```

Load Occurs when the server control is loaded into the page object.

```
void GridView_TrvExpTrans_Load(object sender, EventArgs e)
{
    this.DS_TrvExpTrans.GetDataSet().DataSetViews[0].
        Metadata.DataSources["TrvExpTrans"].CalculatedFields["editCategoryOrSubCategoryName"].
        Mandatory = true;
}
```

PreRender Occurs after the control object is loaded but prior to rendering.

```
protected void gridEPSalesLineEdit_PreRender(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        this.UpdateDesign(ApplicationProxy.InventDimFormDesignUpdate.Init);
    }
}
```

Tip Avoid changing any properties that trigger data binding in the PreRender event of the grid control. Instead, do that in the PreRender event of the page.

RowCancelingEdit Occurs after the Cancel button of a row in edit mode is clicked but before the row exits edit mode.

```
void GridView_TrvExpTrans_RowCancelingEdit(object sender, GridViewCancelEditEventArgs e)
{
    if (this.CurrentRow != null && !this.CurrentRow.IsNew)
    {
        //Undo the changes of autopostback and updateonpostback
        if (this.CurrentRecord != null)
            this.CurrentRecord.Reread();
    }
    this.UpdateDetailsFormsMode(DetailsViewMode.ReadOnly);
}
```

RowCommand Occurs when a button is clicked in a GridView control.

```
protected void gridEPSalesLineEdit_RowCommand(object sender, GridViewCommandEventArgs e)
{
    // Redirect to the configurator:
    if (e.CommandName.Equals("ConfigureLine") && this.gridEPSalesLineEdit.AllowEdit)
    {
        EPPBAUtility.LaunchConfigurator(e, sender, dsEPSalesTableEdit,
            gridEPSalesLineEdit, AxSession.AxaptaAdapter, this);
    }
}
```

RowDataBound Occurs when a data row is bound to data in a GridView control.

```
protected void gridEPSalesLineEdit_RowDataBound(Object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow &&
        this.gridEPSalesLineEdit.AllowEdit == true && (pbaIsEnabled || pcIsEnabled))
    {
        // Decide if the current line should have a configurator enabled icon:
        EPPBAUtility.ConditionallyEnableConfigureLineIcon(e, sender, AxSession.AxaptaAdapter);
    }
    deliveryScheduleFieldsReadOnly(isDeliveryLine());
}
```

SelectedIndexChanged Occurs when a row's Select button is clicked, but only after the GridView control handles the select operation.

```
protected void gridEPSalesLineEdit_SelectedIndexChanged(object sender, EventArgs e)
{
    this.UpdateDesign(ApplicationProxy.InventDimFormDesignUpdate.Init);
    deliveryScheduleFieldsReadOnly(isDeliveryLine());
}
```

DataBound Occurs after the server control binds to a data source.

Init Occurs when the server control is initialized, which is the first step in its life cycle.

RowCreated Occurs when a row is created in a GridView control.

RowDeleted Occurs when a row's Delete button is clicked, but only after the GridView control deletes the row.

RowDeleting Occurs after a row's Delete button is clicked but before the GridView control deletes the row.

RowEditing Occurs after a row's Edit button is clicked but before the GridView control enters edit mode.

RowUpdated Occurs when a row's Update button is clicked, but only after the GridView control updates the row.

RowUpdating Occurs after a row's Update button is clicked but before the GridView control updates the row.

Examples

Example 1: Inserting a row

To programmatically insert a row in the grid, set AllowInsert to true on the AxGridView control, and call the CreateRow method on the grid from the toolbar action menu item clicked event.

```
void AttendeeToolBar_ActionMenuItemClicked(object sender, ActionMenuItemEventArgs e)
{
    switch (e.MenuItem.MenuItemAOTName.ToLower())
    {
        // The add button
        case ATTENDEE_ADD:
            try
            {
                this.ActivityAttendeeGrid.AllowInsert = true;
                this.ActivityAttendeeGrid.CreateRow();
    …
}
```

Example 2: Multi-selection

To enable multi-selection in the grid, set AllowMarking to true on the AxGridView control. To get the selected rows, use the GetMarkedRowsSet method.

```
protected void OkButton_Click(object sender, EventArgs e)
{
    bool ok = true;

    //create AX container for RecIds
    IAxaptaContainerAdapter recIds = this.AxSession.AxaptaAdapter.CreateAxaptaContainer();

    //get marked rows from the grid into a Set collection
    IReadOnlySet<DataSetViewRow> rows =
        this.dsProjQuotationTableInfo.GetDataSourceView("SalesQuotationTable").
            DataSetView.GetMarkedRowsSet();

    IEnumerator<DataSetViewRow> enumRows = rows.GetEnumerator();

    //copy the RecIds of the selected row to an AX container to be sent as parameter
    while(enumRows.MoveNext())
    {
        //test if the selected line is not already transferred
        if (enumRows.Current.GetFieldValue("SalesQuotationLine!Transferred2Forecast").ToString()
            != NoYes.Yes.GetHashCode().ToString())
                recIds.Add(Convert.ToInt64(
                    enumRows.Current.GetFieldValue("SalesQuotationLine!RecId").ToString()));
    }

    //transfer SalesQuotation lines to forecast
    if (recIds.Count != 0)
    {
        ok = ProjForecastBudget.transferQuoteLines(this.AxSession.AxaptaAdapter, recIds);
    }

    if (ok)
    {
        AxUrlMenuItem urlMenuItem = new AxUrlMenuItem("EPProjTableList");
        urlMenuItem.MenuItemContext = null;
        Response.Redirect(urlMenuItem.Url.OriginalString, false);
    }
}
```

Example 3: Conditionally making a grid row editable

The following example shows how you can programmatically make a grid row editable based on certain conditions. In this example, we will make the row editable for all even account numbers.

```
public partial class AxWebUserControl : System.Web.UI.UserControl
{
    void Page_Init(object sender, EventArgs e)
    {
        this.AxGridView1.SelectedIndexChanged += new EventHandler(AxGridView1_SelectedIndexChanged);
    }

    void AxGridView1_SelectedIndexChanged(object sender, EventArgs e)
    {
        this.EnableGridEditing(this.IsCurrentAccountNumberEven());
    }

    private void EnableGridEditing(bool enable)
    {
        if (enable)
        {
            this.AxGridView1.AllowEdit = true;
            this.AxGridView1.EditIndex = this.AxGridView1.SelectedIndex;
        }
        else
        {
            this.AxGridView1.EditIndex = -1;
            this.AxGridView1.AllowEdit = false;
        }
    }

    private bool IsCurrentAccountNumberEven()
    {
        DataSet dataSet = this.CustomersInfoDS.GetDataSet();
        DataSetViewRow currentRow = dataSet.DataSetViews[this.AxGridView1.DataMember].GetCurrent();
        if (currentRow != null)
        {
            string accountNumberStr = (string)currentRow.GetFieldValue("AccountNum");

            if (!string.IsNullOrEmpty(accountNumberStr))
            {
                int accountNumber = Int32.Parse(accountNumberStr);

                return accountNumber % 2 == 0;
            }
        }

        return false;
    }
}
```

For more information about AxGridView, see http://msdn.microsoft.com/en-us/library/cc584514.aspx.

AxHierarchicalGridView

The AxHierarchicalGridView control is used when you want to display hierarchical data in a grid format. For example, you might have a grid that displays a list of tasks in a project. The hierarchical grid is useful in this case, because each task can have subtasks, and all the tasks and subtasks should be displayed in a single grid.

HierarchyIdFieldName is used to uniquely identify a row, whereas HierarchyParentIdFieldName is used to identify the parent of a specific row. The following shows sample markup for an AxHierarchicalGrid control.

```
<dynamics:AxDataSource ID="AxDataSource1" runat="server" DataSetName="Tasks" ProviderView="Tasks">
</dynamics:AxDataSource>
<dynamics:AxHierarchicalGridView ID="AxHierarchicalGridView1" runat="server"
    BodyHeight="" DataKeyNames="RecId" DataMember="Tasks"
    DataSetCachingKey="e779ece0-43b7-4270-9dc9-33f4c61d42b7"
    DataSourceID="AxDataSource1" EnableModelValidation="True"
    HierarchyIdFieldName="TaskId" HierarchyParentIdFieldName="ParentTaskId">
    <Columns>
        <dynamics:AxBoundField DataField="Title" DataSet="Tasks"
            DataSetView="Tasks" SortExpression="Title">
        </dynamics:AxBoundField>
        <dynamics:AxBoundField DataField="StartDate" DataSet="Tasks"
            DataSetView="Tasks" SortExpression="StartDate">
        </dynamics:AxBoundField>
        <dynamics:AxBoundField DataField="EndDate" DataSet="Tasks"
            DataSetView="Tasks" SortExpression="EndDate">
        </dynamics:AxBoundField>
    </Columns>
</dynamics:AxHierarchicalGridView>
```

Title	StartDate	EndDate
Task A	**1/4/2011**	**2/10/2011**
Task B	1/4/2011	1/29/2011
Task C	1/9/2011	2/10/2011
Task D	**1/20/2011**	**5/1/2011**
Task E	**1/20/2011**	**3/2/2011**
Task G	3/5/2011	4/12/2011
Task F	2/15/2011	4/12/2011

Figure 32 An example of an AxHierarchicalGridView control

AxFilter

The AxFilter control is used to filter the data that is retrieved by using a data source. It sets a filter on DataSetView by using AxDataSourceView, which is responsible for keeping the data in sync with the filter that is set by calling SetAsChanged and ExecuteQuery when data has changed. AxDataSourceView and DataSetView expose the following APIs that let you access the filter programmatically.

SystemFilter Gets the complete list of ranges on the QueryRun (including open, hidden, and locked ranges) into the conditionCollection on the filter object.

UserFilter Gets only the open ranges on the QueryRun into the conditionCollection on the filter object.

ResetFilter Clears the filter that is set on the QueryRun, and therefore resets the filter (ranges) that is set programmatically.

You can set the range in the data set in X++ as follows.

```
qbrBlocked =  qbds.addRange(fieldnum(CustTable,Blocked));
qbrBlocked.value(queryValue(CustVendorBlocked::No));
qbrBlocked.status(RangeStatus::Hidden);
```

To read the filter that is set on the data source in a web user control, use the following code.

```
this.AxDataSource1.GetDataSourceView(this.AxGridView1.DataMember).SystemFilter.ToXml();
```

Alternatively, you can use the following code.

```
this.AxDataSource1.GetDataSet().DataSetViews[this.AxGridView1.DataMember].SystemFilter.ToXml();
```

In both cases, the return value will look something like this.

```
<?xml version="1.0" encoding="utf-16"?><filter xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="CustTable"><condition attribute="Blocked"
operator="eq" value="No" status="hidden" /></filter>
```

You can also set the filter programmatically.

```
string myFilterXml = @"<filter name='CustTable'><condition attribute='CustGroup' status='open'
value='10' operator='eq' /></filter>"

this.AxDataSource1.GetDataSourceView(this.AxGridView1.DataMember).SystemFilter.AddXml(myFilterXml);
```

The AxGridView control also uses AxFilter when ShowFilter is set to true. You can access the AxFilter object by using AxGridview.FilterControl, and you can access the filter XML by using AxGridView.Filter. The filter reads the metadata from the AxDataSource that is linked to the grid and displays filtering controls dynamically, to let the user filter the data source with any of its fields that are not hidden or locked. The filtering controls are rendered above the grid.

AxContextMenu

The AxContextMenu control is used to create and display a context menu. It provides methods for adding and removing menu items and separators at run time. It also provides methods for resolving client or Enterprise Portal URLs.

```
AxUrlMenuItem myUrlMenuItem = new AxUrlMenuItem("MyUrlMenuItem");
AxContextMenu myContextMenu = new AxContextMenu();
myContextMenu.AddMenuItemAt(0, myUrlMenuItem);
```

The AxGridView control uses AxContextMenu when the ShowContextMenu property is set to true. You can access the AxContextMenu object by using the syntax AxGridview.ContextMenu.

BoundField controls

These controls are used when you use data binding to display data from the database. These controls are not displayed in the Visual Studio toolbox.

AxBoundField Displays the value of a data-bound field group as text.

AxHyperLinkBoundField Displays the value of a data-bound field as a hyperlink.

AxCheckBoxBoundField Displays the value of a data-bound field as a check box.

AxRadioButtonBoundField Displays the value of a data-bound field as a radio button.

AxDropDownBoundField Displays the value of a data-bound field as a drop-down menu.

AxReferenceBoundField Displays the value of a reference data-bound field as text.

Examples

Example 1: Getting the bound field

The following example shows how you can get the value of a specific field from a field collection.

```
static AxBoundField GetField(DataControlFieldCollection fields, string name)
{
    foreach (DataControlField field in fields)
    {
        AxBoundField boundField = field as AxBoundField;
        if (boundField != null && String.Compare(boundField.DataField, name, true) == 0)
            return boundField;
    }

    return null;
}

AxBoundField parentCaseId = (AxBoundField)GetField(this.GeneralRight.Fields, "editParentCaseId**");
```

Example 2: Disabling bound fields through code

You can make a bound field read-only or hide the Lookup button of the bound field through code in the page load event.

```
AxBoundField boundField = AxGridView1.Columns[i] as AxBoundField;
boundField.ReadOnly = true;

// Or

AxBoundField boundField = AxGridView1.Columns[i] as AxBoundField;
boundField.LookupButtonDisplaySettings = LookupButtonDisplaySettings.Never;
```

Summary fields

To show a field as SummaryField, add the field to the AutoSummary group on the table. Developers can also do this by setting the FastTabSummary attribute on a field group (AxBoundFieldGroup control) or an individual field (AxBoundField control).

The following are the possible values of FastTabSummary and their effects:

- **Auto** Honor the settings in the AutoSummary group.
- **Yes** Show the current field as SummaryField, even if it is not in the AutoSummary group.
- **No** Do not show the current field as SummaryField, even if it is in the AutoSummary group.

Example: The following example shows how to programmatically set the FastTabSummary property of a particular field (StartDateTime) in an AxGroup.

```
protected void Page_Init(object sender, EventArgs e)
{
    this.AxGroup1.Load += new EventHandler(AxGroup1_Load);
}

void AxGroup1_Load(object sender, EventArgs e)
{
    foreach (DataControlField dcf in this.AxGroup1.DataControlFieldCollection)
    {
        AxBoundField boundField = dcf as AxBoundField;
        if (boundField != null) && (boundField.DataField == "StartDateTime")
        {
            boundField.FastTabSummary = FastTabSummary.Yes;
        }
    }
}
```

Template fields

Use template fields in AxGridView and AxForm controls if you need to display data in a way that the base controls do not provide. ItemTemplate defines how a particular cell of a column will be rendered and what its contents will be in terms of constituent controls. EditItemTemplate defines how a particular cell of a column will be rendered when the parent row is in edit mode.

Example 1: Displaying an image button in a column

```
<asp:TemplateField AccessibleHeaderText="pbaColumn">
    <ItemTemplate>
        <asp:ImageButton ID="btnPBAConfigure" runat="server"
            ImageUrl="/_layouts/EP/images/PBAItemConfiguration_disabled.png"
            CssClass="PBAConfigureLineButton"
            CommandName="ConfigureLine"
            AxCtrlType="AxCmdFieldConfigureBtn"
            ToolTip="<%$ AxLabel:@SYS40223 %>" />
        </ItemTemplate>
</asp:TemplateField>
```

Example 2: Displaying a lookup and a text box in a column

```
<asp:TemplateField>
    <ItemTemplate>
        <dynamics:AxLookup ID="AxLookupPeriod" runat="server" AutoPostBack="true"
            CssClass="AxLookupButtonBF"
            ExtendedDataType="ProjId"
            HoverCssClass="AxLookupButtonHoverBF"
            TargetControlId="TextBoxPeriod"
            OnOkClicked="AxLookupPeriod_OkClicked" OnLookup="AxLookupPeriod_Lookup">
        </dynamics:AxLookup>
        <div style="display: none" id="divDate">
            <asp:TextBox ID="TextBoxPeriod" runat="server"></asp:TextBox>
        </div>
    </ItemTemplate>
</asp:TemplateField>
```

Example 3: Disabling the cell and hiding the lookup for a specific row

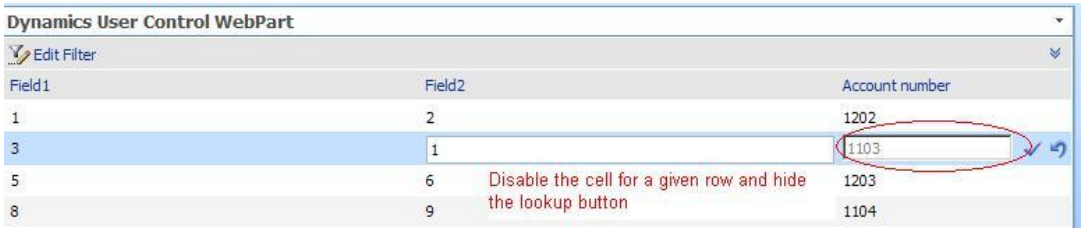


Figure 33 Disabling a cell and hiding a lookup by using template fields

```
<asp:TemplateField ConvertEmptyStringToNull="False" HeaderText="<%$ AxLabel:@SYS1996 %>"
    SortExpression="AccountNum">
        <EditItemTemplate>
            <asp:TextBox ID="TextBox1" runat="server"
                Columns="<%$ AxDataSet:DemoSet.Table1.AccountNum.DisplayLength %>"
                Enabled="<%$ AxDataSet:DemoSet.Table1.AccountNum.AllowEdit %>"
                MaxLength="<%$ AxDataSet:DemoSet.Table1.AccountNum.StringSize %>"
                Text='<%# Bind("AccountNum") %>'></asp:TextBox>
            <dynamics:AxLookup ID="AxLookup1" runat="server" DataLookupField="AccountNum"
                DataSet="DemoSet" DataSetView="Table1" TargetControlId="TextBox1"
                Visible="<%$ AxDataSet:DemoSet.Table1.AccountNum.AllowEdit %>">
            </dynamics:AxLookup>
        </EditItemTemplate>
        <ItemTemplate>
            <asp:Label ID="Label1" runat="server" Text='<%# Bind("AccountNum") %>'></asp:Label>
        </ItemTemplate>
        </asp:TemplateField>
```

The following code behind conditionally hides the lookup control and disables the cell.

```
protected void AxGridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row != null && e.Row.RowType == DataControlRowType.DataRow)
    {
        // If this is an internal project hide the lookup
        // and disable the third column which displays account number
        DataSetViewRow dataRow = (DataSetViewRow)e.Row.DataItem;
        bool isInternalProject = dataRow.GetFieldValue("Field2").ToString() == "1";

        if (isInternalProject)
        {
            Control c = e.Row.Cells[2].FindControl("AxLookup1");
            if (c!= null)
                c.Visible = false;
            e.Row.Cells[2].Enabled = false;
        }
    }
}
```

For more information about template fields, see Using Template Fields in User Controls on MSDN.

Action Pane and toolbars

AxActionPane

The AxActionPane control is used to display the Action Pane at the top of the page. The Action Pane is similar to the SharePoint Ribbon. The WebMenuName property of the AxActionPane control is used to reference the web menu that contains the menu items to display as buttons on the Action Pane. The buttons are displayed on tabs and in groups to improve discoverability. The DataSource and DataMember properties of the AxActionPane control are used to associate the data that the Action Pane buttons will act on. This control is not displayed in the Visual Studio toolbox. The Action Pane Web Part can be used as an alternative to the AxActionPane control.

To use the AxActionPane control in a web user control, you will need to add a reference to the Microsoft.Dynamics.Framework.Portal.SharePoint assembly. This can be done by adding the following to the markup of the web user control.

```
<%@ Register Assembly="Microsoft.Dynamics.Framework.Portal.SharePoint, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
Namespace="Microsoft.Dynamics.Framework.Portal.SharePoint.UI.WebControls" TagPrefix="dynamics" %>
```

The definition of the Action Pane is based on a WebMenu definition in the AOT.

- First-level WebMenu definitions become tabs.
- Second-level WebMenu definitions become button groups.
- Third-level and lower-level WebMenu definitions become drop-down or fly-out menus.
- WebMenuItems are always rendered as buttons.

Controlling the layout of buttons within a button group

- The AxActionPane control first arranges the buttons within a button group based on size, and then in the order in which they are defined.
- Buttons that have their Big property set to Yes are placed at the beginning of the button group (to the left in left-to-right ordering).
- Buttons that have their Big property set to No appear to the right in top-to-bottom column ordering.

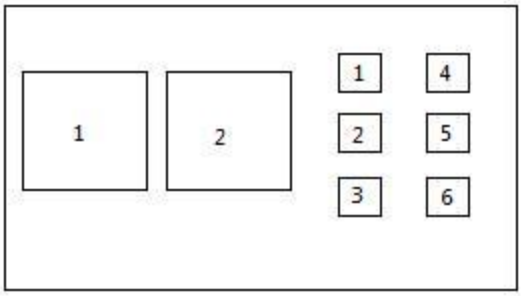


Figure 34 The button layout and ordering within a button group

Adding icons to buttons

- For list pages, both the Microsoft Dynamics AX client and Enterprise Portal get the image based on the ResourceId that is set in the NormalImage property of the Action Pane button, menu item, or web menu item.
- For task pages, you can set the NormalImage property directly on the web menu item.
- Remember that Imagelocation=File is not supported in Enterprise Portal.

Tip If the Enterprise Portal page has multiple grids, such as one primary grid and other secondary grids, only the primary grid must be bound to the AxActionPane control. The secondary grids must be bound to their respective AxToolbar controls.

Example 1: Checking an action menu item and refreshing the data source

```
void webpart_ActionMenuItemClicked(object sender, ActionMenuItemEventArgs e)
{
    if (e.MenuItem.MenuItemAOTName == "HRMEPVirtualNetworkSkillDelete")
        dsVirtualNetworkSkill.GetDataSet().DataSetViews["HRMVirtualNetworkSkill"].ExecuteQuery();
}
```

Example 2: Executing an AxActionMenuItem through code

```
AxActionMenuItem action = new AxActionMenuItem("SysFlushAOD");
    using (Proxy.Args args = new Proxy.Args(AxBaseWebPart.GetWebpart(this).Session.AxaptaAdapter))
    {
        args.parm = "Test1";
        action.Run(args);
}
```

AxToolbar

The AxToolbar control is used to display a toolbar at a certain location on the page, as an alternative to using the Action Pane at the top of the page. For example, you might choose to display a toolbar with New, Edit, and Delete actions at the top of a grid control.

The AxToolbar control internally leverages the SharePoint toolbar controls. AxToolbarButton, which is used within the AxToolbar control, is derived from SPLinkButton. It is used to render top-level buttons. Similarly AxToolBarMenu, which is used within the AxToolbar control, is derived from Microsoft.SharePoint.WebControls.Menu. This is used to render a drop-down menu via a callback when a button is clicked. The menu item properties can thus be modified before the menu items are rendered.

The Toolbar Web Part can be used as an alternative to the AxToolbar control. Generally, you use the Toolbar Web Part to control the display of toolbar menu items. But if you have a task page that contains master/detail information, such as a purchase requisition header and line items, you should use the AxToolbar ASP.NET control inside your web user control, above the detail AxGridview control, to let the user add and manage the line items.

You can bind the AxToolbar control to an AxDataSource or use it as an unbound control. When the controls are bound, the menu item context is automatically based on the currently selected item. When the controls are unbound, you have to write code to manage the toolbar context.

You can point the toolbar to a web menu in the AOT through the WebMenuName property. The AOT web menu lets you define a multilevel menu structure with the SubMenu, MenuItem, and MenuItem reference nodes. Each top-level menu item is rendered by using AxToolbarButton as a link button. Each top-level submenu is rendered by using AxToolbarMenu as a drop-down menu. If you have nested submenus, the second and further levels are displayed as submenus.

Like the Action Pane the definition of the toolbar is based on a WebMenu definition in the AOT.

- First-level WebMenu definitions are rendered by using AxToolbarButton as a link button.
- Second-level WebMenu definitions are rendered by using AxToolbarButton as a drop-down menu.
- Third-level and lower-level WebMenu definitions are rendered as fly-out menus.
- WebMenuItems are rendered as buttons.

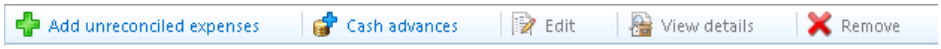


Figure 35 An AxToolbar control

Events

SetMenuItemProperties This event occurs before the toolbar is rendered. It can be used to change the behavior of the menus and menu items. For example, you can show, hide, or customize specific menu items.

```
void Webpart_SetMenuItemProperties(object sender, SetMenuItemPropertiesEventArgs e)
{
    // Remove the menu item context (query string will be empty)
    if (e.MenuItem.MenuItemAOTName == "tutorial_CustomerCreate")
         ((AxUrlMenuItem)e.MenuItem).MenuItemContext = null;

    // Set a client script for a top level button
    if (e.MenuItem.MenuItemAOTName == "SysFlushAOD")
        e.MenuItem.ClientOnClickScript = "alert('hello refresh aod')";

    // Hide the whole drop down
    if (e.MenuItem.MenuItemAOTName == "EPSalesTableList" && e.IsDefaultMenuItem)
        e.MenuItem.Hidden = true;

    // Change menu item text and help text
    if (e.MenuItem.MenuItemAOTName == "tutorial_CustomerEdit")
    {
        e.MenuItem.DisplayName = "Edit My Customer";
        e.MenuItem.HelpText = "My Customer Help Text";
    }
}
```

You can also use the SetMenuItemProperties event to set or remove context.

```
void Webpart_SetMenuItemProperties(object sender, SetMenuItemPropertiesEventArgs e)
{
    // Do not pass the currently selected customer record context, since this menu is for
    // creating new (query string should be empty)
    if (e.MenuItem.MenuItemAOTName == "EPCustTableCreate")
         ((AxUrlMenuItem)e.MenuItem).MenuItemContext = null;
}
```

ActionMenuItemClicking This event occurs after a toolbar button is clicked but before the action corresponding to the button is executed. The following example prevents the action corresponding to the button from being executed by setting RunMenuItem to false.

```
void webpart_ActionMenuItemClicking(object sender, ActionMenuItemClickingEventArgs e)
    {
        if (e.MenuItem.MenuItemAOTName.ToLower() == "EPCustTableDelete")
        {
            e.RunMenuItem = false;
        }
    }
```

ActionMenuItemClicked This event occurs after a toolbar button is clicked. The following example deletes the selected row in the grid when the delete button is clicked.

```
void webpart_ActionMenuItemClicked(object sender, ActionMenuItemEventArgs e)
    {
   if (e.MenuItem.MenuItemAOTName.ToLower() == "EPCustTableDelete")
        {
                int selectedIndex = this.AxGridView1.SelectedIndex;
                if (selectedIndex != -1)
                {
                    this.AxGridView1.DeleteRow(selectedIndex);
                }
        }
    }
```

Tip If you want to enable or disable a toolbar menu item when a field is modified, set AutoPostBack to true on the bound field and UpdateOnPostBack to true on the grid. When the field is modified, the RowUpdating and RowUpdated events will fire on the AxDataSourceView (which is the view that the grid is bound to in the AxDataSource control). You can write code in these events to enable and disable the toolbar menu item.

Adding icons to buttons

- As for the Action Pane, Enterprise Portal gets the image for a toolbar button based on the ResourceId that is set in the NormalImage property of the web menu item.
- Remember that Imagelocation=File is not supported in Enterprise Portal.
- The toolbar always shows small images that are 16x16 pixels.

Example: Programmatically adding items to an AxToolbar control. This example also shows you how you can cache the generated toolbar to improve performance.

```
protected void Page_Load(object sender, EventArgs e)
{
    this.TrvExpTransGridToolbar.WebMenuGeneratorConstructor = this.GetExpenseTransToolbarGenerator;
}

private IWebMenuGenerator<WebMenuGeneratorEventArgs> GetExpenseTransToolbarGenerator()
{
    if (this.expenseTransToolbarMenu == null)
    {
        // Check the state cache
        IAxStateCache stateCache = AxStateCacheFactory.CreateStateCache(this.Page.Session);

        if (stateCache != null)
        {
            object cachedState = stateCache.LoadState(this.GetExpenseTransToolbarMenuCacheKey());

            if (cachedState != null)
            {
                // Got it from the cache - deserialize it
                this.expenseTransToolbarMenu =
                    WebMenuItemMetadataHierarchicalContainer.Deserialize((byte[])cachedState);
            }
            else
            {
                // Not found in the cache - create it
                this.expenseTransToolbarMenu = this.GetExpenseTransToolbarMenu();

                // Save it in the cache
                stateCache.SaveState(this.GetExpenseTransToolbarMenuCacheKey(),
                    this.expenseTransToolbarMenu.Serialize());
            }
        }
        else
        {
            // Caching not enabled - simply create the toolbar
            this.expenseTransToolbarMenu = this.GetExpenseTransToolbarMenu();
        }
    }

    // Create the generator for the web menu container
    return new  WebMenuItemMetadataHierarchicalContainerGenerator(this.expenseTransToolbarMenu);
}

private string GetExpenseTransToolbarMenuCacheKey()
{
    long projRecId = 0L;
    long worker = 0L;

    // Get per row state that impacts the expense category menu
    if (this.CurrentRow != null)
```

```
    {
        projRecId = (long)this.CurrentRow["ProjTable"];
        worker = (long)this.CurrentRow["CreatingWorker"];
    }

        return string.Format("ExpenseTransToolbarMenuCacheKey{0}{1}", projRecId, worker);
}

private WebMenuItemMetadataHierarchicalContainer GetExpenseTransToolbarMenu()
{
    long projRecId = 0L;
    long worker = 0L;

    // Get per row state that impacts the expense category menu
    if (this.CurrentRow != null)
    {
        projRecId = (long)this.CurrentRow["ProjTable"];
        worker = (long)this.CurrentRow["CreatingWorker"];
    }

    // Create the container that holds the expense transaction toolbar menu items
    WebMenuItemMetadataHierarchicalContainer expenseTransToolbarMenu =
        new WebMenuItemMetadataHierarchicalContainer("ExpenseTransToolbarMenu");

    // Create the container that will hold the expense category sub menu
    WebMenuItemMetadataHierarchicalContainer expenseCategorySubMenu =
    (WebMenuItemMetadataHierarchicalContainer)ApplicationProxy.EPTrv.createExpenseCategoryMenu(
        projRecId, worker);

    // Add it as the first tool bar button
    expenseTransToolbarMenu.AddSubMenu(expenseCategorySubMenu);

    // Get the remaining web menu items from the web menu defined in the AOT
    WebMenuMetadata trvExpTransGridToolbarMenu = MetadataCache.GetWebMenuMetadata(
        this.AxSession, "TrvExpTransGridToolbar");

    foreach (IWebTreeNode<WebMenuMetadata> currentItem in trvExpTransGridToolbarMenu.Children)
    {
        // Create the AxWebMenuItem to get the menu item metadata
        AxWebMenuItem webMenuItem = MenuItemFactory.CreateMenuItem(this.AxSession, currentItem);

        if (webMenuItem != null)
        {
            // Add the menu item
            expenseTransToolbarMenu.AddWebMenuItem(webMenuItem.MenuItemMetadata);
        }
    }

    return expenseTransToolbarMenu;
}
```

EPTrv.createExpenseCategoryMenu is an X++ class, as shown in the following code. In the preceding code, it is called by using proxies.

```
public static
    Microsoft.Dynamics.Ax.Services.Platform.Client.WebMenuItemMetadataHierarchicalContainer
        createExpenseCategoryMenu(RefRecId projRecId = 0, RefRecId worker = 0)
{
    Microsoft.Dynamics.Ax.Services.Platform.Client.WebMenuItemMetadataHierarchicalContainer
        categoryMenu;

    Microsoft.Dynamics.Ax.Services.Platform.Client.WebMenuItemMetadataHierarchicalContainer
        otherSubMenu;

    Microsoft.Dynamics.Ax.Services.Platform.Client.ActionWebMenuItemMetadata
        categoryActionMenuItem;

    Query categoryQuery;
    QueryRun queryRun;
    TrvCostType trvCostType;

    categoryMenu = new
        Microsoft.Dynamics.Ax.Services.Platform.Client.WebMenuItemMetadataHierarchicalContainer(
            'CategoryMenu', "@SYS135720", '');

    // create other submenu to have other categories
    otherSubMenu = new
        Microsoft.Dynamics.Ax.Services.Platform.Client.WebMenuItemMetadataHierarchicalContainer(
        'OtherCategoryMenu', "@SYS69996", "@SYS69996");

    categoryQuery = EPTrv::categoryLookup(projRecId, worker);
    queryRun = new QueryRun(categoryQuery);

    while (queryRun.next())
    {
        trvCostType = queryRun.get(tableNum(TrvCostType));

        // Construct the menu item for the expense category
        categoryActionMenuItem = new
            Microsoft.Dynamics.Ax.Services.Platform.Client.ActionWebMenuItemMetadata(
                'AddNew_' + trvCostType.CostType,
                trvCostType.CostType,
                trvCostType.CostTxt,
                '');

        if (trvCostType.IsCommon)
        {
            categoryMenu.AddWebMenuItem(categoryActionMenuItem);
        }
        else
        {
            otherSubMenu.AddWebMenuItem(categoryActionMenuItem);
        }
    }

    categoryMenu.AddSubMenu(otherSubMenu);

    return categoryMenu;
}
```

AxLookup

AxLookup is used in data entry pages to help the user select a valid value for a field that references keys from other tables. In Enterprise Portal, lookups are metadata-driven by default, and they are automatically enabled for fields, based on the relationship that is defined in metadata in the AOT.

An example is the customer group lookup on the customer creation page. The extended data type (EDT) and table relationship metadata in the AOT define a relationship between the customer table and the customer group table; therefore, a lookup is rendered so that the user can select a customer group in the customer group field when creating a customer record. You don't need to write any code to enable this behavior—it happens automatically.

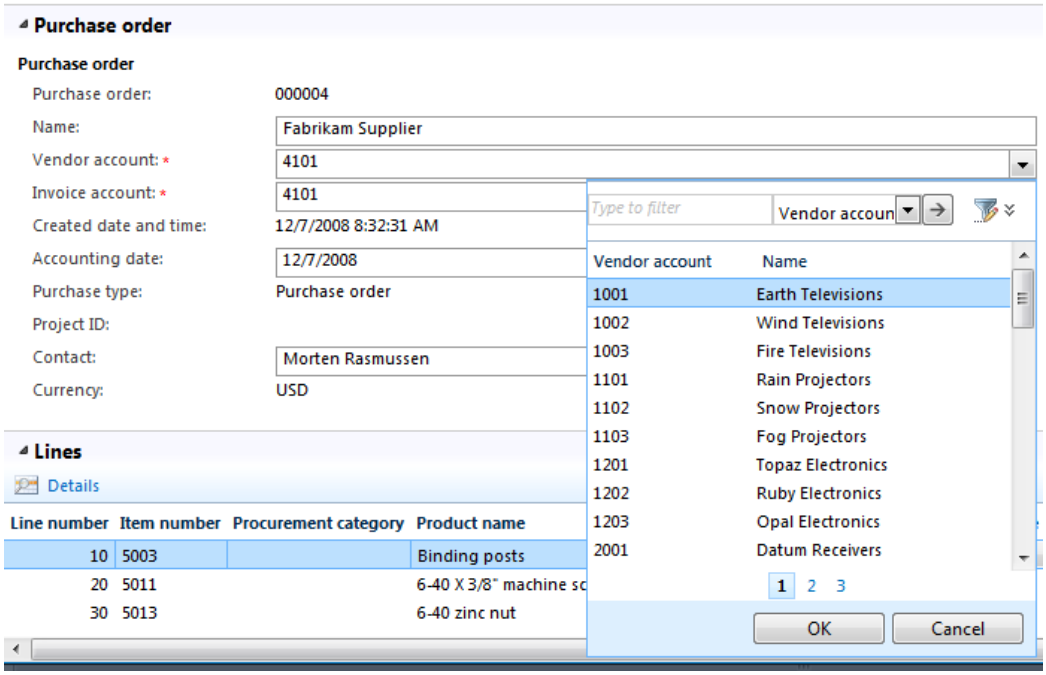


Figure 36 An AxLookup control for selecting a vendor account

Developers can select the type of lookup that they want (DataSetField, Extended Data Type, and so on) and then set the corresponding properties (DataSet, DataSetView, DataLookupField, or ExtendedDataType). Developers also need to set the TargetControlId property to point to the control that will receive the value from the AxLookup control.

In some application scenarios, the automatic behavior isn't sufficient, and you might be required to customize the lookup. The lookup infrastructure of Enterprise Portal is designed to offer flexibility and customization options in both X++ and C#, so that developers can tailor the lookup user interface and the data retrieval logic to their needs.

In the AOT Data Set node, you can override the dataSetLookup method of the field in the data source to control the lookup behavior. For example, if you want to filter the values displayed for a ZIP/postal code field, based on what has been entered for a country/region, state, or county, you can override dataSetLookup as shown in the following X++ code.

```
void dataSetLookup(SysDataSetLookup sysDataSetLookup)
{
    if (custTable.CountryRegionId)
    {
        sysDataSetLookup.parmQuery().dataSourceNo(1).addRange(
            fieldnum(AddressZipCode,CountryRegionId)).value(queryValue(custTable.CountryRegionId));
    }

    if (custTable.State)
    {
        sysDataSetLookup.parmQuery().dataSourceNo(1).addRange(
            fieldnum(AddressZipCode,State)).value(queryValue(custTable.State));
    }

    if (custTable.County)
    {
        sysDataSetLookup.parmQuery().dataSourceNo(1).addRange(
            fieldnum(AddressZipCode,County)).value(queryValue(custTable.County));
    }
}
```

You can also customize the lookup in C# in the web user control by writing code in the Lookup event of bound fields, or by using the AxLookup control for fields that don't have data binding. To use AxLookup to provide lookup values for any ASP.NET control that isn't data bound, you should set the TargetControlID property of the AxLookup control to the ASP.NET control to which the lookup value is to be returned. Alternatively, you can base AxLookup on the EDT, data set, custom data set, or custom User Control by specifying the LookupType property. You can also control which fields are displayed in the lookup and which ones are returned. You can do this either through the markup or through code. You can write code to override the Lookup event and control the lookup behavior, as shown in the following code.

```
protected void AxLookup1_Lookup(object sender, AxLookupEventArgs e)
    {
        AxLookup lookup = (AxLookup)sender;

        // Specify the lookup fields
        lookup.Fields.Add(AxBoundFieldFactory.Create(this.AxSession,
            lookup.LookupDataSetViewMetadata.ViewFields["CustGroup"]));

        lookup.Fields.Add(AxBoundFieldFactory.Create(this.AxSession,
            lookup.LookupDataSetViewMetadata.ViewFields["Name"]));
}
```

For more information about lookups, see Lookups on MSDN.

Multi-selection

The AxLookup control supports multi-selection. This helps in scenarios where the user needs to choose multiple records from the lookup. When you click OK, the lookup will create a list of the selected field values, separated by semicolons (or a custom separator that you have specified).

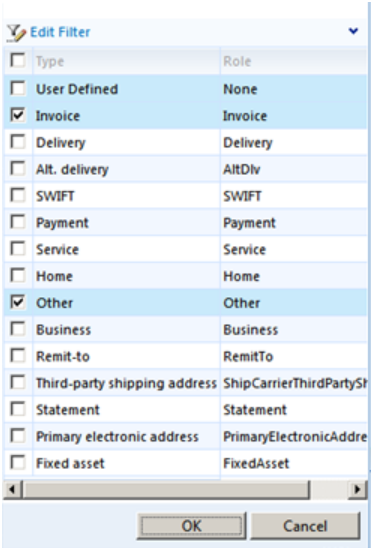


Figure 37 Multi-selection in AxLookup

AllowMarking This is a property on the AxLookup control. By default, it is set to false. When it is set to true, the lookup will enable multi-selection.

Separator This parameter is used to specify a custom separator that is used when multiple selected values are returned. By default, a semicolon (;) is used.

SetMarkedRows This method is used to preset the selected/marked rows in the lookup. It accepts the set of ViewDataKeys and marks the matching records in the lookup.

```
public void SetMarkedRows(IEnumerable<IAxViewRowKey> markedRecordsViewRowKeys)
```

OKClicked This event will be fired when the OK button in the lookup is clicked. It will provide access to the grid DataSetView, so that developers can programmatically access the marked records in the lookup.

Examples

Markup

```
<asp:TextBox ReadOnly="true" CssClass="AxInputField" ID="LocationRoleType"
    runat="server"></asp:TextBox>
<asp:Literal ID="LocationRoleTypeText" Visible="false" runat="server"></asp:Literal>
<dynamics:AxLookup AutoPostBack="true" ID="LocationRoleLookup" runat="server"
    OnLookup="LocationRoleLookupMethod" TargetControlId="LocationRoleType">
</dynamics:AxLookup>
```

Role lookup method

```
/// <summary>
/// The lookup override for the location role field
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void LocationRoleLookupMethod(object sender, AxLookupEventArgs e)
{
    AxLookup lookup = (AxLookup)sender;
    lookup.AllowMarking = true;
    lookup.AllowPaging = false;

    // Create the lookup dataset - we will do a lookup in the CustGroup table
    using (Proxy.SysDataSetBuilder sysDataSetBuilder =
        Proxy.SysDataSetBuilder.constructLookupDataSet(this.AxSession.AxaptaAdapter,
            TableMetadata.TableNum(this.AxSession, "LogisticsLocationRole")))
    {
        // Set the run time generated data set as the lookup data set
        lookup.LookupDataSet = new Microsoft.Dynamics.AX.Framework.Portal.Data.DataSet(
            this.AxSession, sysDataSetBuilder.toDataSet());
    }

    // Get the roles converted into keys
    IEnumerable<IAxViewRowKey> viewKeys = getViewDataKeys(e.LookupControl.LookupDataSetViewMetadata);

    lookup.SetMarkedRows(viewKeys);

    // Get the ID of RoleType Base Enum
    int roleEnum = EnumMetadata.EnumNum(this.AxSession, "LogisticsLocationRoleType");
    using (Proxy.DictEnum dictEnum = new Proxy.DictEnum(this.AxSession.AxaptaAdapter, roleEnum))
    {
        // This label corresponds to "One time" address
        string roleType2 = dictEnum.index2Label((int)LogisticsLocationRoleType.OneTime);


        // We are adding the XML for the filter conditions
        // We are adding the Type and Name filters. Adding 2 type filters is not working
        // The One-Time name is same as One-Time Type label
        lookup.LookupDataSet.DataSetViews[0].SystemFilter.AddXml(
            @"<filter name='Roles'>
            <condition attribute='IsPostalAddress' status='open' value='1' operator='eq' />
            <condition attribute='Name' status='open' value='" + roleType2 + @"' operator='ne' />
            </filter>");
    }
}
```

Method for creating view data keys

```
/// <summary>
/// Get the view datakeys based on the provided recIds
/// </summary>
/// <param name="viewMetaData"></param>
/// <returns></returns>
private List<IAxViewRowKey> getViewDataKeys(DataSetViewMetadata viewMetaData)
{
    TableMetadata tableMetaData = MetadataCache.GetTableMetadata(
        TableMetadata.TableNum("LogisticsLocationRole"));
    String[] roles = Roles.Split(';');
    List<IAxViewRowKey> viewDataKeys = new List<IAxViewRowKey>();

    // For each recid
    foreach (string role in roles)
    {
        Int64 recId = 0;
        // Sanitize
        if (Int64.TryParse(role, out recId))
        {
            // Create dictionary
            Dictionary<string, Int64> dict = new Dictionary<string, Int64>();
            //dict.Add((tableMetaData.Fields.GetByName("RecId")).FieldId, recId);
            dict.Add("RecId", recId);

            // Create indexes
            List<IndexMetadata> index = new List<IndexMetadata>(1);
            index.Add(tableMetaData.DefaultUniqueIndex);

            // Get viewdatakey
            viewDataKeys.Add((IAxViewRowKey)AxViewDataKey.CreateFromDictionary(
                viewMetaData, dict, index.ToArray()));
        }
    }

    return viewDataKeys;
}
```

The OKClicked event

```
/// <summary>
/// The ok clicked event
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void lookup_OkClicked(object sender, AxLookupEventArgs e)
{
    Roles = String.Empty;
    int roleCount = 0;

    this.IsFlagChanged = true;

    // Get all the chosen recids
    foreach (DataSetViewRow row in e.LookupControl.LookupDataSetView.GetMarkedRowsSet())
    {
        Int64 role = (Int64)row.GetFieldValue("RecId");

        roleCount++;

        Roles += role.ToString();
        if (roleCount < e.LookupControl.LookupDataSetView.GetMarkedRowIndexes().Count)
        {
            Roles += ";";
        }
    }
}
```

PreLoad

Lookups can be marked as PreLoad. When the LookupStyle property is set to PreLoad, the data for the first page of the lookup is fetched when the page is initially rendered. When the user clicks the lookup button, the lookup will be displayed immediately on the client side, without going back to the server. If the lookup is generally small (less than 50 items), marking it as a client-side lookup is recommended.

```
<dynamics:AxLookup runat="server" ID="CategoryLookup" CssClass="AxLookupButtonBF"
    HoverCssClass="AxLookupButtonHoverBF" TargetControlId="Category" AutoPostBack="true"
    Visible="true" OnLookup="CategoryLookup_Lookup" LookupStyle="PreLoad" >
</dynamics:AxLookup>
```

You can also do the same thing for bound fields that are rendered as lookups.

```
<dynamics:AxBoundField DataSet="TrvExpTrans" DataSetView="TrvExpTrans" DataField="PayMethod"
OnLookup="PayMethod_Lookup" LookupStyle="PreLoad"/>
```

LookupCacheScope

To improve the performance of lookups, you can also set the LookupCacheScope property to Global or DataBindingContainer.

Global Causes the lookup to be cached at the company level. Use this if you expect the lookup values to remain the same from one page to another.

DataBindingContainer Causes the lookup to be cached at the page level. Use this if you expect the lookup values to change from one page to another.

You can also set this programmatically, as shown in the following code sample.

```
private void AttachLookupsToBoundFields()
{
    foreach (DataControlField dcf in this.Group_Transaction.DataControlFieldCollection)
    {
        AxBoundField boundField = dcf as AxBoundField;
        if (boundField != null)
        {
            switch (boundField.DataField)
            {
                case "ExchangeCode":
                    boundField.LookupStyle = LookupStyle.PreLoad;
                    boundField.LookupCacheScope = CacheScope.Global;
                    break;
                case "PayMethod":
                    boundField.Lookup += new EventHandler<AxLookupEventArgs>(PayMethod_Lookup);
                    boundField.LookupStyle = LookupStyle.PreLoad;

                    // since CostType is not changing on this page, we can cache it for the form
                    boundField.LookupCacheScope = CacheScope.DataBindingContainer;
                    break;
            }
        }
    }
}
```

AxDatePicker

This control provides an easy way for users to select a date from a calendar. Developers can set the Extended-DataType property if required. Developers also need to set the Target property to point to the control that receives the value from the AxDatePicker control.

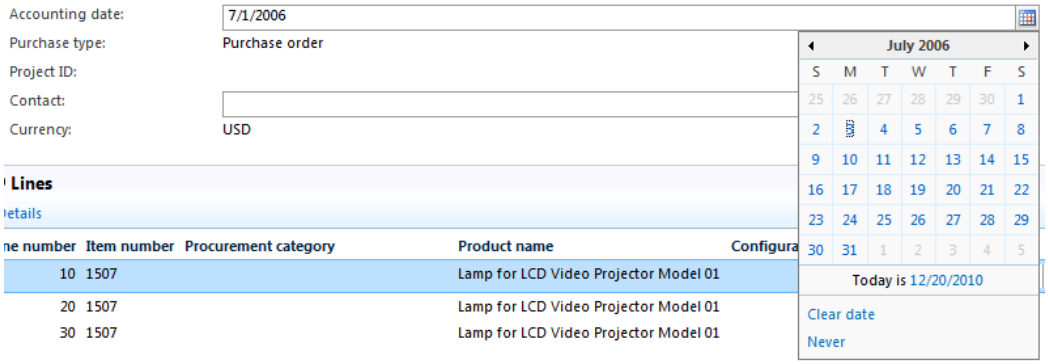


Figure 38 An AxDatePicker control (drop-down calendar for the Accounting date field)

AxDateTimeHelper

This control applies the user's preferred time zone to the current UTC DateTime value for use in your code behind.

```
DateTime sampleDateTime = AxDateTimeHelper.GetUserNow(this.AxSession);
```

FactBox controls

AxPartContentArea

This control is used to add FactBoxes to a details page. It acts as a container for the AxFormPart, AxInfoPart, and CueGroupPartControl controls.

AxFormPart

This control is used to display a form part on the page. The MenuItemName property must be set to point to the menu item for the form part. It must be placed inside an AxPartContentArea control.

AxInfoPart

This control is used to display an info part on the page. The MenuItemName property must be set to point to the menu item for the info part. It must be placed inside an AxPartContentArea control.

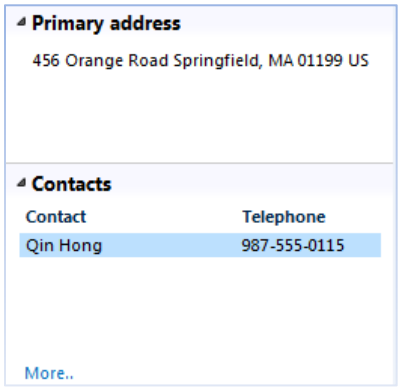


Figure 39 AxInfoPart controls

CueGroupPartControl

This control is used to display a cue group on the page. The MenuItemName property must be set to point to the menu item for the cue group. It must be placed inside an AxPartContentArea control.

Figure 40 A CueGroupPartControl control

AxContentPanel

The ASP.NET UpdatePanel control enables sections of the webpage to be refreshed, so that the entire page does not have to be refreshed with a postback. This makes the page more interactive. The AxContentPanel control derives from the UpdatePanel control and extends its functionality to provide Microsoft Dynamics AX–specific constructs, such as passing record context and applying it to the underlying child controls.

AxControlPanel is useful if you have a child user control that needs to be displayed based on the context provided by the data source in the parent user control. It is also useful if you have two independent data sources that do not have a direct table-level relation, but one needs to change based on the context from the first data source after executing some business logic.

Example: You have a page with separate header and detail data sources, and you need to apply the context of the newly created header record to the details section.

```
<dynamics:AxDataSource ID="DemoExpenseHeaderDS" runat="server"
    DataSetName="DemoExpenseHeader" ProviderView="DemoExpenseTable">
</dynamics:AxDataSource>
…
…
Header part of markup
…
…
<dynamics:AxSection ID="AxSection4" runat="server" Caption="Expense Lines"
    Font-Bold="True" Font-Size="Large" ForeColor="#336600">
    <dynamics:AxContentPanel ID="AxContentPanel_ExpenseLine" runat="server">
        <ContentTemplate>
            <dynamics:AxDataSource ID="DemoExpenseLineDS" runat="server"
                DataSetName="DemoExpenseLine" Role="Consumer">
            </dynamics:AxDataSource>
            <dynamics:AxGridView ID="DemoExpenseLineGrid" runat="server" AllowDelete="True"
                AllowEdit="True" DataKeyNames="RecId"
                DataMember="demoexpenseline" DataSourceID="DemoExpenseLineDS"
                ShowFilter="False">
                <Columns>
                    <dynamics:AxDropDownBoundField DataField="ExpenseCategory"
                        DataSet="DemoExpense" DataSetView="demoexpenseline"
                        SortExpression="ExpenseCategory">
                    </dynamics:AxDropDownBoundField>
…
…
                </Columns>
            </dynamics:AxGridView>
        </ContentTemplate>
    </dynamics:AxContentPanel>
…
```

Provide the context from the header to the expense lines (that is, the details section) in the code behind

```
protected void Page_Init(object sender, EventArgs e)
{
    IAxContext contextInterface = AxContextHelper.FindIAxContext(this);
    if (contextInterface != null)
        contextInterface.CurrentContextChanged += new
            EventHandler<AxCurrentContextChangedEventArgs>(CurrentContextChanged);
}

protected void CurrentContextChanged(object sender, AxCurrentContextChangedEventArgs e)
{
    if (e.CurrentContext != null)
    {
        this.AxContentPanel_ExpenseLine.ExternalContext = e.CurrentContext.RootTableContext;
    }
}
```

AxPopup

AxPopup controls are used to open a page in a pop-up browser window. When a pop-up page is closed, AxPopup controls pass data from the pop-up page back to the parent page and trigger an OnPopupClosed server event on the parent. This functionality is encapsulated in two controls: AxPopupParentControl, which is used on the parent page, and AxPopupChildControl, which is used on the pop-up page. These both derive from AxPopupBaseControl. These controls are AJAX-compatible, and therefore they can be created conditionally as part of a partial update.

Data can be passed from the pop-up page back to the parent page by using AxPopupField objects. These are exposed through the Fields property of the AxPopupBaseControl control, from which both AxPopupParentControl and AxPopupChildControl are derived.

AxPopupParentControl and AxPopupChildControl have fields with the same names. When the pop-up page closes, the value of each field of the AxPopupChildControl control is assigned (via client-side script) to the corresponding field in the AxPopupParentControl control.

AxPopupField can optionally be associated with another control, such as TextBox or any other control, by assigning its TargetId property to the ID property of the target control. This is useful, for example, when the pop-up page has a TextBox control. In order to pass the user input to the parent page when the pop-up page is closed, and to do it entirely on the client and therefore avoid the round trip, you need to associate a field with the TextBox control.

Example

The AxPopupField control is placed in the user control displaying the list.

```
<dynamics:AxPopupParentControl ID="PopupConvertCustomer" runat="server" PopupHeight ="180"
    PopupWidth="400"  >
    <dynamics:AxPopupField name="hiddenCustomerAccountNo"  />
</dynamics:AxPopupParentControl>
```

In the code behind, the toolbar action opens the pop-up page by using the GetOpenPopUpEventReference method.

```
protected void AddCustomerAccountNoScript(SetMenuItemPropertiesEventArgs e, string custAccount)
{
    AxUrlMenuItem menuItem = new AxUrlMenuItem(CUSTOMER_ACCOUNT_DIALOG);
    DataSetViewRow row = this.GetCurrentDataSetViewRow();
    if (row != null)
    {
        AxTableContext context = AxTableContext.Create(
            row.GetTableDataKey(row.DataSetView.Metadata.RootDataSource, null));

        menuItem.MenuItemContext = context;

        //Adding the CustAccount QueryString variable
        if (custAccount != string.Empty)
        {
            menuItem.ExtraParams.Add("CustAccount", custAccount);
        }
        menuItem.RemoveNavigation = true;

        //Calling the javascript function to set the properties of opening the customer account
        //on clicking the menu items.
        e.MenuItem.ClientOnClickScript =
            this.PopupConvertCustomer.GetOpenPopupEventReference(menuItem);
    }
}
```

The markup for the child control (pop-up) uses an AxPopupField field with the same name as the parent control (that is, hiddenCustomerAccountNo) to pass the value back and forth. The AxPopupField field in the child control gets the value from the text box by using the TargetControlID property.

```
<div>
    <br />
    <br />
    <table style="width: 100%">
        <tr>
            <td class="PopoverFormText">
                <asp:Label ID="lblCustAccount" runat="server" Text="<%$ axlabel:@SYS7149 %>">
                </asp:Label>
            </td>
            <td class="PopoverFormText" >
                <asp:TextBox ID="txtCustAccount" runat="server" MaxLength="20" ></asp:TextBox>
                <dynamics:AxPopupChildControl ID="popupChild" runat="server">
                    <dynamics:AxPopupField name="hiddenCustomerAccountNo"
                        TargetControlId="txtCustAccount" />
                </dynamics:AxPopupChildControl>
            </td>
        </tr>
        <tr><td colspan="3"><br /> <hr class="hr" />
            </td></tr>
        <tr>
            <td align="right" colspan="2">
            <asp:Button  id = "OkButton" CssClass="okCancelButton" runat ="server"
                Text="<%$ axlabel:@SYS5473 %>" onclick="OkButton_Click" />
            <input id="CancelButton" class="okCancelButton" runat ="server" type="button"
                value="<%$ axlabel:@SYS50163 %>" onclick="window.close();" />
            </td>
            <td style="width: 10%"></td>
        </tr>
    </table>
</div>
```

The code behind validates the input and closes itself, giving control back to the parent page.

```
//Use to validate the CustAccount no entered.
protected void OkButton_Click(object sender, EventArgs e)
{
    try
    {
        if (this.txtCustAccount.Text.Equals(string.Empty))
        {
            //Displaying error message: Account number is not specified
            DisplayInfolog(InfoType.Error, "@SYS24085");
            return;
        }

        //Validating the Customer Account no. entered
        if (!ApplicationProxy.SmmOpportunityStatusUpdate.checkCustomerAccount(
            this.AxSession.AxaptaAdapter, this.txtCustAccount.Text))
        {
            return;
        }

        //Calling the script for closing the dialogbox
        this.popupChild.ClosePopup(true, true);
    }
    catch (Exception ex)
    {
        AxExceptionCategory exceptionCategory;

        // This returns true if the exception can be handled here
        if (!AxControlExceptionHandler.TryHandleException(this, ex, out exceptionCategory))
        {
            // The exception was fatal - in this case we re-throw.
            throw;
        }
    }
}
```

For more information about pop-up pages, see http://msdn.microsoft.com/en-us/library/cc592938.aspx.

AxModalPrompt

The AxModalPrompt control is used to prompt the user to confirm certain actions or make selections in a small dialog box. The user cannot access the parent page until the prompt is dismissed. The default size of the dialog box for the prompt is auto. You can override this by using the DialogSize property to specify a width and height for the prompt.

```
private AxModalPrompt<MessagePromptInputData, PromptResult> validationPrompt;

protected void Page_Load(object sender, EventArgs e)
{
    validationPrompt = new AxModalPrompt<MessagePromptInputData, PromptResult>();
    validationPrompt.SetRuntimeHostedUserControlProvider<MessagePrompt>(this.CreateMessagePrompt);
    validationPrompt.PromptDismissed += new
        EventHandler<PromptDismissedEventArgs<PromptResult>>(this.ValidationPromptDismissed);
    this.Controls.Add(validationPrompt);
}

private MessagePrompt CreateMessagePrompt()
{
    return new MessagePrompt();
}

private void ValidationPromptDismissed(object sender, PromptDismissedEventArgs<PromptResult> e)
{
    if (e.ClickedButton == PromptClickedButton.Ok)
        Label1.Text = "OK";
    else
        Label1.Text = "Cancel";
}

protected void Button1_Click(object sender, EventArgs e)
{
    validationPrompt.Show(new MessagePromptInputData("Title1", "Message1",
        PromptAutoButtons.OkCancel), null);
}
```

AxReportViewer

The AxReportViewer control is used to display SSRS reports on the page. The Select a report property must be set to point to the menu item for the report.

AxEnhancedPreview

The AxEnhancedPreview control references a URL. When the user hovers over the content within the AxEnhancedPreview control, it displays a preview of the webpage that the URL points to in a small pop-up window. This behavior is also used in the CueGroup FactBox if the cue is mapped to an info part for preview through the PreviewPartReference property.

Example: In the following code sample, when you hover over the label “Hover over here,” the control will display a preview of http://www.microsoft.com in a little pop-up window.

```
<dynamics:AxEnhancedPreview ID="AxEnhancedPreview1" runat="server"  Enabled='true'
    Url='http://www.microsoft.com' Height="260" Width="500" ShowBullet="True">
    <asp:Label ID="Label1" runat="server" Text="Hover over here"></asp:Label>
</dynamics:AxEnhancedPreview>
```

For more information about framework controls, see User Control Components on MSDN.

List and details pages

In Enterprise Portal, we enforce a set of consistent page types and layouts throughout the application. The two most common page types are list pages and details pages. Let's see how developers can easily and quickly create simple list and details pages.

Model-driven list pages

A list page in Enterprise Portal displays a list of records. In Microsoft Dynamics AX 2012, a new model-driven way of creating list pages was introduced. In Microsoft Dynamics AX 2009, developers had to create a Form to display on the client and a webpage to display in Enterprise Portal. With model-driven list pages, you model the list page once, and it can appear both on the Microsoft Dynamics AX client and in Enterprise Portal. The form displayed on the Microsoft Dynamics AX client and the webpage displayed in Enterprise Portal share code and metadata. Any changes to the Form are automatically reflected both on the Microsoft Dynamics AX client and in Enterprise Portal. This leads to a number of advantages, such as reduced development effort, a unified codebase, and easier maintenance.

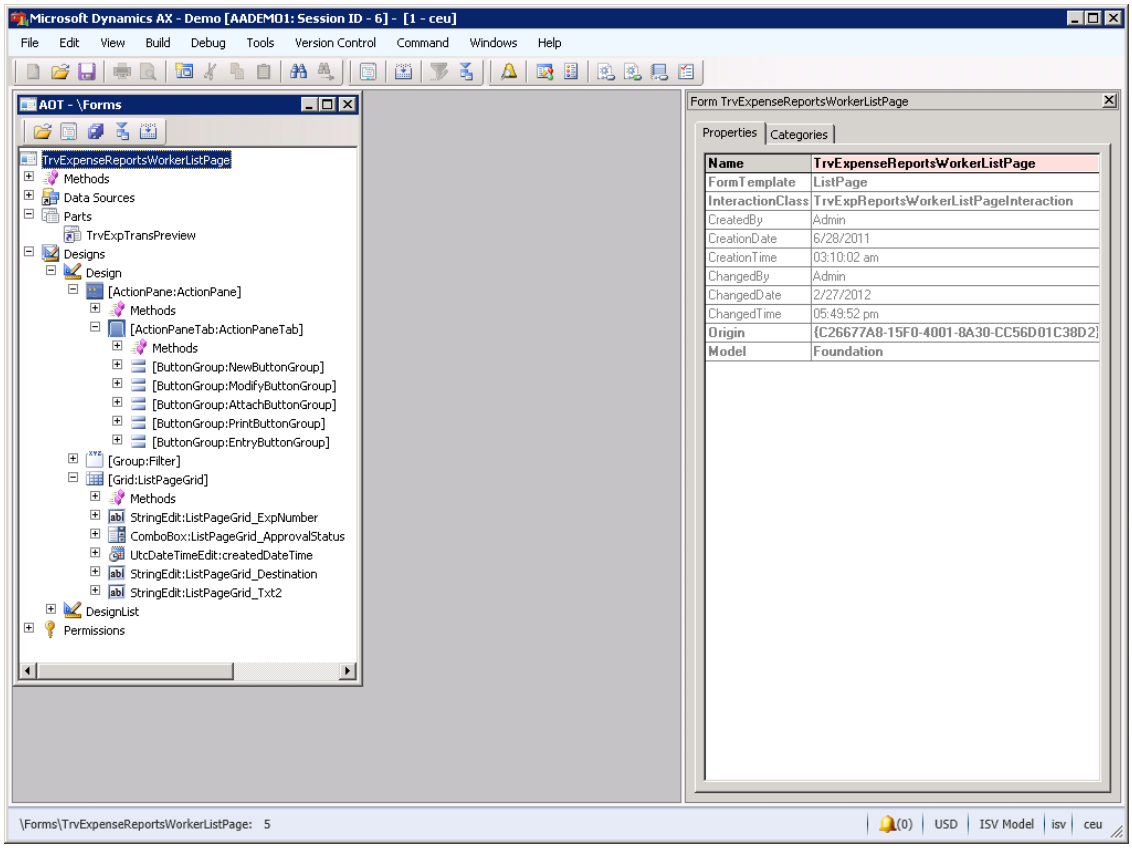


Figure 41 A snapshot of model-driven list page development

The following list describes the sequence of high-level steps that you can follow to create a model-driven list page:

1. Start the Development Workspace.
2. Create a new Form in the AOT, and set the FormTemplate property to ListPage. This will automatically add some design elements, such as the filter, grid, and Action Pane.
3. Set the query on the form to get the data to be displayed in the form.
4. Set the DataSource on the grid to the required data view.
5. Add the fields to display in the grid.
6. Create and add an Action Pane and info parts, if required. Ideally, you should create one info part to be displayed in the Preview Pane (below the grid), and one or more info parts, form parts, and cue groups to be displayed in the FactBox area (to the right of the grid). The Preview Pane should display extended information about the selected record, and the FactBoxes should display related information. To link these parts to the list page, you will need to create the corresponding display menu items.
7. Create a display menu item that points to the form. Right-click the menu item, and then click Deploy to EP.
8. When prompted, select the module that you want to deploy the page to.

   This will automatically create a SharePoint Web Part page for the list page for Enterprise Portal. It will also create a URL web menu item and import the corresponding page definition into the AOT.
9. Set the HyperLinkMenuItem property on the first field in the grid to a display menu item corresponding to a details page. This will render links in the first column that can be used to open up the record by using a linked details page. (**Note** See the next section for information about how to create a details page).

For more information about list page development, see List Page Reference on MSDN.

To achieve more control over how your model-driven list page behaves, you can specify a custom interaction class by using the InteractionClass property on the form. This is discussed in more detail in later sections.

Details pages

A details page in Enterprise Portal displays detailed information about a specific selected record.

The following list describes the high-level steps that you can follow to create a details page:

1. Start Visual Studio, and use the EP Web Application Project template (found under the Microsoft Dynamics AX category) to create a new project.
2. Add a new item to the project by using the EP User Control with Form template (found under the Microsoft Dynamics AX category). This will also automatically add the control to the AOT.
3. Switch to design view, select the AxDataSource control, and set the DataSet name.
4. Select the AxForm control, and ensure that DataSourceID is set to the AxDataSource.
5. Set the DataMember and DataKeyNames properties on the form as appropriate.
6. If required, change the default mode of the form to Edit or Insert (it is ReadOnly by default).
7. To auto generate the Save and Close buttons:
    a. In ReadOnly mode: Set AutoGenerateCancelButton to true.
    b. In Edit mode: Set AutoGenerateEditButton to true.
    c. In Insert mode: Set AutoGenerateInsertButton to true.
    d. Select an AxGroup control, and ensure that the FormID property is set.
8. Click the Edit Fields link, and add the required fields to the AxGroup control.
9. Compile the EP Web Application by using the Build menu. Ensure that there are no errors. This will also automatically deploy the control to the SharePoint directory.
10. Start the Development Workspace, and navigate to \Web\Web Content\Managed.

11. Right-click the Managed item that maps to the web user control that you have created, and click Deploy to EP.

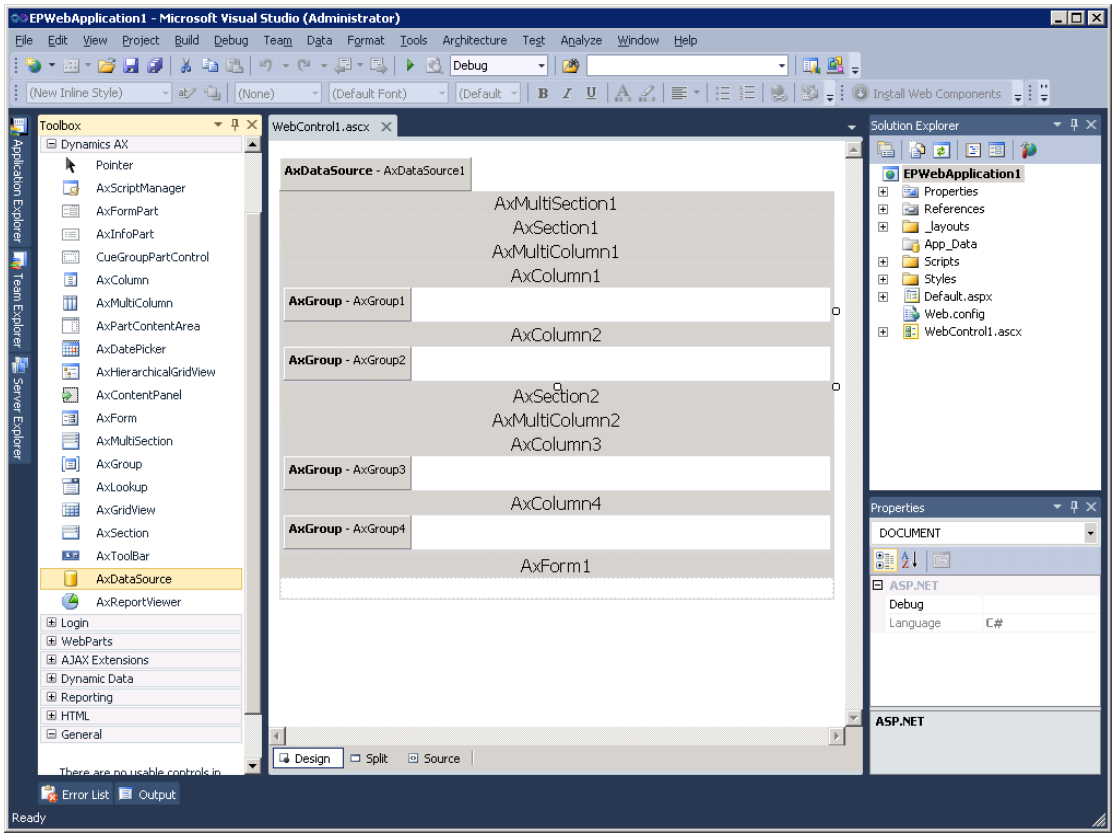


Figure 42 A snapshot of details page development

12. When prompted, select the module to deploy the page to.

    This will automatically create a SharePoint Web Part page for Enterprise Portal and put your web user control on the page by using the User Control Web Part. It will also create a URL web menu item and import the corresponding Page Definition into the AOT.

13. Select the URL web menu item that you created for the page, and set the WindowMode property to Modal. This will cause the details page to open in a modal window.

14. Create a new Display Menu Item, and set the WebMenuItemName property to the URL Web Menu Item that is linked to the details page.

15. Use this Display Menu Item to link to the details page from the list page grid, as described in the "Model-drive list pages" section.

Enterprise Portal has also made it very easy to implement a variety of interaction patterns by using modal windows. This is discussed in more detail in later sections.

Advanced list page development

List page interaction classes

To achieve more control over how your model-driven list page behaves, you can specify a custom interaction class by using the InteractionClass property on the form. The name of your class should end with ListPageInteraction, and it can inherit either the SysListPageInteractionBase class, which is easy to use, or the ListPageInteraction class, which is more flexible. The following are some of the methods that the SysListPageInteractionBase class provides that developers can override and add custom code to:

- **initialized** Called after the list page has been initialized.
- **initializing** Called when the list page is initializing.
- **selectionChanged** Called when the list page selection changes.
- **setButtonEnabled** Enables or disables buttons. This method is called from the selectionChanged method.
- **setButtonVisibility** Displays or hides buttons. This method is called when the form opens.
- **setCaption** Changes the form caption. This method is called when the form opens.
- **setGridFieldVisibility** Shows or hides grid fields. This method is called when the form opens.
- **setListPageType** Used to identify the type of list page when you work with secondary list pages.
- **setModeledQueryName** Replaces the query. This method is called when the form opens.

Example: You can override the setButtonEnabled button, and add code to enable or disable buttons on the Action Pane.

```
protected void setButtonEnabled()
{
    TrvExpTable trvExpTable = this.listPage().activeRecord(identifierstr(TrvExpTable));
    TrvHcmWorkerRecId parmWorker = this.getWorker();

    boolean isEditable = trvExpTable.isEditable();
    boolean isDeletable = isEditable && trvExpTable.ApprovalStatus != TrvAppStatus::Returned;
    boolean userHasCurrentDelegateAccess =
        EPTrv::userHasCurrentDelegateAccessFor(parmWorker);

    super();

    this.listPage().actionPaneControlEnabled(
        formcontrolstr(TrvExpenseReportsWorkerListPage, EditExpenseReport),
        isEditable && userHasCurrentDelegateAccess);

    this.listPage().actionPaneControlEnabled(
        formcontrolstr(TrvExpenseReportsWorkerListPage, ViewExpenseReport), !isEditable);

    this.listPage().actionPaneControlEnabled(
        formcontrolstr(TrvExpenseReportsWorkerListPage, DeleteExpenseReportConfirmation),
            isDeletable);
}
```

For more information about list page interaction classes, see http://msdn.microsoft.com/en-us/library/syslistpageinteractionbase.aspx.

Secondary list pages

Secondary list pages are used to display a filtered subset of the data displayed on the primary list page. For example, if the primary list page displays a list of vehicles, a secondary list page might display the list of rented vehicles. Given an existing primary list page, you can complete the following steps to create a corresponding secondary list page:

1. Create a new query:
    a. Add the primary list page's query under Composite Query.
    b. Add the required range.

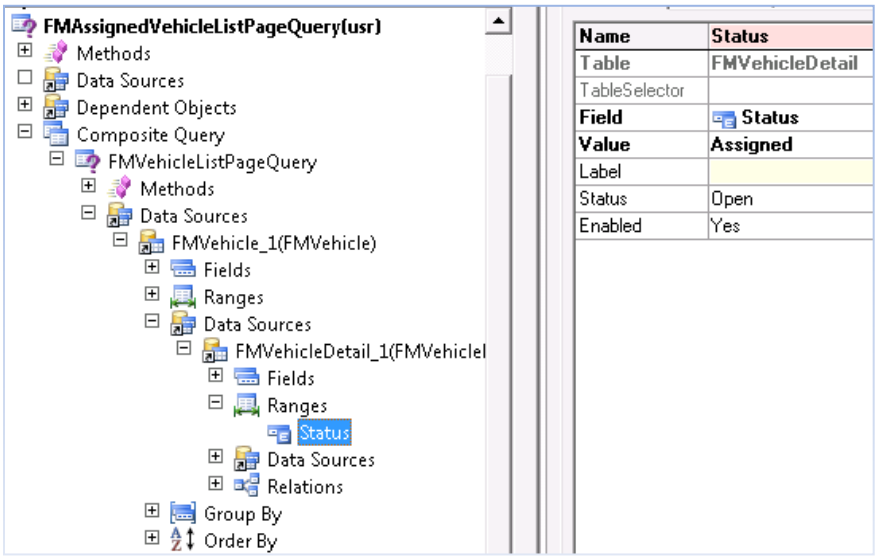


Figure 43 A modeled query for a secondary list page

2. Create a new Display Menu Item:

    a. Associate the menu item with the primary list page.

    b. Set the new query on the menu item.

    c. Set the EnumTypeParameter property on the menu item enumeration that contains the various list page types. For example, for vehicles, it can be set to All, Available, or Rented.

    d. Set the EnumParameter property on the menu item to the type of the list page.

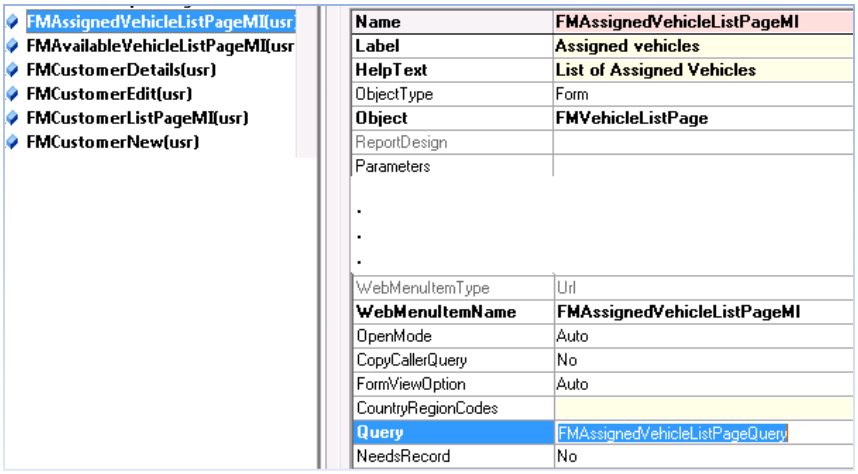


Figure 44 The display menu items for primary and secondary list pages

3. In the interaction class, determine and set the list page type by using the setListPageType function. The EnumParameter value set on the menu item is passed to that function as _listPageArgs.

```
protected void setListPageType(ListPageArgs _listPageArgs)
{
    if (_listPageArgs && _listPageArgs.enumTypeParameter() == enumnum(SalesTableListPage))
    {
        salesTableListPage = _listPageArgs.enumParameter();
    }
    else
    {
        salesTableListPage = SalesTableListPage::Main;
    }
}
```

4. In the interaction class, make columns, Action Pane controls, and list page columns visible or hidden as needed. You can use the setButtonVisiblity method.

```
protected void setButtonVisibility()
{
    switch (this.getListPageType())
    {
        case SalesTableListPage::ShippedNotInvoiced :
        //Invoice tab
            this.listPage().actionPaneControlVisible(formcontrolstr(SalesTableListPageNew,
                RelatedDocumentsInvoice), false);
            this.listPage().listPageFieldVisible(formcontrolstr(SalesTableListPageNew,
                listpageField1), true);
        break;

        case SalesTableListPage::Open :
        // Sell tab
            this.listPage().actionPaneControlVisible(formcontrolstr(SalesTableListPageNew,
                GroupNew), false);
        break;
    }

    this.listPage().actionPaneControlVisible(formcontrolstr(SalesTableListPageNew,
        manageQualityGeneral), UseQualityManagement);
}
```

5. Add the menu item to Quick Launch.

Multi-selection

To act on multiple selected rows, complete the following steps. This works for both Enterprise Portal and the Microsoft Dynamics AX client.

1. Create an X++ class, and use the _args.multiSelectionContext class to get the marked rows.
2. Create an action menu item that points to this class, and set the MultiSelect property of the menu item to Yes.
3. Create a MenuItemButton on the Action Pane, and set the MenuItemName property to the action menu item that you created.

Example: Iterating through the selected records

```
static public void main(Args _args)
{
    Common multiSelectionRecord;
    MultiSelectionContext multiSelectionContext;

    multiSelectionContext = _args.multiSelectionContext();

    if (multiSelectionContext)
    {
        multiSelectionRecord = multiSelectionContext.getFirst();
        while (multiSelectionRecord)
        {
            info(int642str(multiSelectionRecord.RecId));
            multiSelectionRecord = multiSelectionContext.getNext();
        }
    }
}
```

General guidelines

- Even if you are creating the list page only for Enterprise Portal, you must still create a client menu item. The Web Part always points to the client menu item.
- Action Pane:
    - For Action Pane elements that update or change the currently selected row, set the AutoRefreshData property to true to automatically refresh the list after the action is completed.
    - For Action Pane elements that must be disabled when no record is found, set the NeedsRecord property to Yes.
    - Use the CommandButton:Delete method to enable delete functionality both on the Microsoft Dynamics AX 2012 client and in Enterprise Portal.
    - Set the NormalImage property to the ResourceId, and it will work both on the client and in Enterprise Portal. ImageLocation file is not supported in Enterprise Portal.
    - If your Action Pane action is specific to Enterprise Portal, you must still create a client menu item. The Action Pane menu item button always points to a client menu item.

    - You can show or hide Action Pane buttons by using the initialized event. Showing or hiding buttons in selection changes is not supported. In selection changes, you should only enable or disable buttons.
    - The DropDialog button is not supported in Enterprise Portal.
    - CommandButtons: Only the Export to Excel and Delete commands are supported. Other commands are not supported in Enterprise Portal.
    - Buttons that need to have some kind of code for control events (client only): Change the DisplayTarget property of the control to Client. This will let you add event codes to your controls. The controls will be rendered only on the Microsoft Dynamics AX client.
- Custom filters:
    - Be sure that the FilterVisible property of the Group class is set to Yes.
    - To display the possible values in the Filter drop-down menu, set the ExtendDataType property for the lookup.
    - To enable customer filters both on the client and in Enterprise Portal, set the FilterDataSource, FilterField, and FilterExpression properties of the Edit controls under the grid. The default value for FieldExpression is %1. You can use any operator or wildcard character—for example, >%1 or *%1*.
    - If the custom filter is client-specific, and you need default values, population of the drop-down menu, and so on, set the DisplayTarget property of the FilterGroupControl control to Client, and add controls or override methods. To set default values, use setInitialFilterControlValue method of SysEPCustomFilter (which controls the filter on the list page).
- Image columns are not supported on model-driven list pages for Enterprise Portal.
- If you are using views for your list page query, make sure that the query has a unique index for Ordering or Paging views.
- Document handling: You should be able to use Enterprise Portal document handling functionality from the model-driven list pages for Enterprise Portal. However, this is not delivered through the client's Attach Document command. For the client, you must add a CommandButton, CommandButton:Attachments, which has Command set to Document handling. This button opens up the list of documents that are attached with the selected record in the list. Set DisplayTarget to Client. To the same ButtonGroup, add another MenuItemButton, point it to the MenuItem DocuViewAction, which has WebMenuItem set to EPDouListFromInfo, and set DisplayTarget to EP.
- You can control the page size of all list pages throughout an installation from System Administration > Setup > Enterprise Portal > Enterprise Portal Parameters > General > Number of rows to display in list pages. After making a change, make sure that the cache is cleared in Enterprise Portal; otherwise, this setting will not take effect.

Preview pane title

1. Create a display method to return a concatenated string (Title1 : Title2).

```
display FMPreviewTitle PreviewTitle()
{
    return this.VehicleID + " : " + this.Description;
}
```

2. Add a Field to the PreviewPane part, and set the Style property to TitleField.

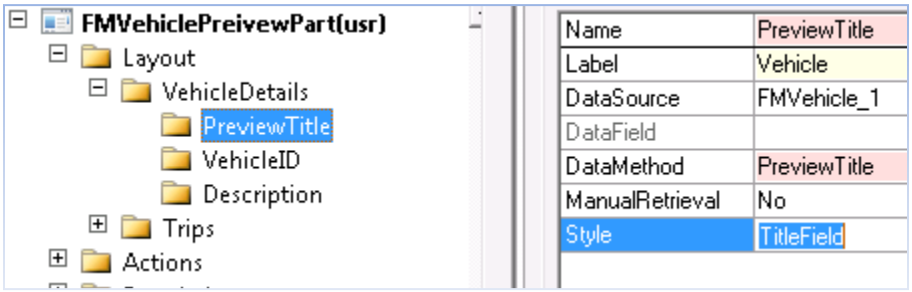


Figure 45 Adding a field that is bound to the display method

Converting Microsoft Dynamics AX 2009 client list pages

1. When you convert an existing list page into a model-driven list page, all existing X++ codes will be wiped out. Your control and data source logic will have to be moved to the interaction class. Therefore, you must duplicate or backup your form node, so that you can reference the old code.
2. If you have an existing list page that has secondary list pages, update their menu items. The old secondary list page menu items would refer to classes that call the SysListPageHelper::runFormWithModeledQuery method. Delete these classes, and set the main list page form as the target for these menu items. Update your interaction class to refer to the correct query when it is called from a secondary list page menu item.
3. If you are moving Enterprise Portal data set code to a list page, and if you are using this.query in the data set, use the _query parameter in the initializequery method.

```
// qbDS = this.query().dataSourceTable(this.table());
// becomes
qbDS = _query.dataSourceTable(tablenum(PurchTableHistory));
```

4. If you are using element.args().dataset(), you can use listpageargs().record().tableid instead.

```
// if (element.args().dataset() == tablenum(PurchReqLine))
// becomes
if (this.listpageargs().record().tableid == tablenum(PurchReqLine))
{
    this.query().dataSourceTable(tablenum(VendPurchOrderJour)).clearDynalinks();

    queryDataSourceLink = this.query().dataSourceTable(tablenum(VendPurchOrderJour));
    queryDataSourceLink.relations(true);
    queryDataSourceLink.addDynalink(fieldnum(VendPurchOrderJour, PurchId),
                                    element.args().record(),
                                    fieldnum(PurchReqLine, PurchId));
}
```

5. If you are using formButtonManager to show or hide Action Pane buttons, change the code to use the this.listpage().actionPaneControlVisible method instead.

```
this.listPage().actionPaneControlVisible(formcontrolstr(
    SalesTableListPageNew, RelatedDocumentsInvoice), false);
```

Converting Microsoft Dynamics AX 2009 secondary list pages

1. Locate the menu item for the secondary list page. It will most likely point to a class.
2. Find and open the class that the menu item points to. It should have code that uses the SysListPageHelper::runFormWithModeledQuery method to run a form that passes in a query. See the following example. Note down the form and query names being used.

```
static void listPageOpen(Args args)
{
    FormRun formRun = SysListPageHelper::runFormWithModeledQuery(
        formstr(SalesTableListPage),


        querystr(SalesTableListPageOpen), "@SYS117040", args);
…
}
```

3. Update the menu item as follows:
    a. Change the ObjectType property from Class to Form.
    b. Set the Object property to the name of the form from the previous step.
    c. Set the Query to the name of the modeled query from the previous step.
4. Remove unnecessary code that uses the runFormWithModeledQuery method. This code is unnecessary, because the query referenced by the menu item gets passed to the form automatically.
5. If there is additional code in the method (for example, code to set visibility), it should be moved to the init method on the form or the list page interaction class.

Report list pages

1. Create a web menu item for each of the reports.
2. You will have to associate the page with the generic report view page (SRSReportViewer.aspx) and set the MenuItemName parameter to the actual report (Output menu item name).

   In other words, set the URL to http:// server/sites/DynamicsAx/Enterprise%20Portal/SRSReportViewer.aspx?MenuItemName=<YourReportMenuItem>.

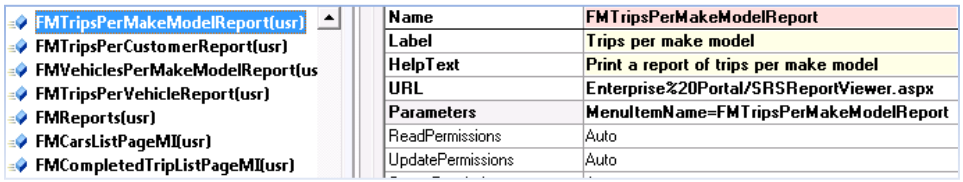


Figure 46 A web menu item for a report list page

3. Create a Web Menu to contain all the reports.

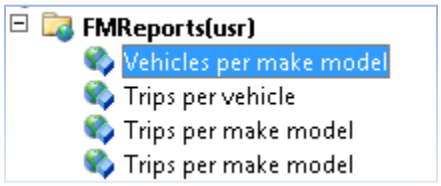


Figure 47 A Web Menu that contains all reports

4. Create a page to show the list of reports.
5. Add a User Control Web Part, and point it to the WebReportWebMenuItemList managed content item.
6. Create a Web Menu Item for the report list page.
7. Add it to Quick Launch, and set the parameters property to webmenu=<the name of the web menu that contains the list of reports>.

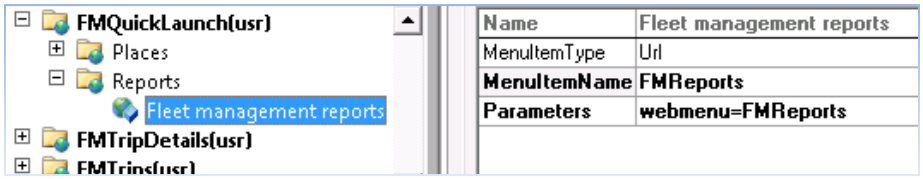


Figure 48 Adding the link to the report list page to Quick Launch

Advanced details page development

Enterprise Portal now includes much richer integration with Visual Studio for development. The following are some key features:

- Visual Studio Projects is now a section of the AOT, and you can now save Enterprise Portal projects to the AOT.
- The development experience is aligned with managed code for the development of reports and generic business logic managed code.
- Web Controls are automatically deployed from within Visual Studio when you use an Enterprise Portal Web Application project.
- Proxies are handled as project items and can exist in multiple projects (as opposed to a single proxy definition file).
- Proxies are automatically regenerated on build, which ensures type consistency.
- All MorphX source control back ends are now integrated into Visual Studio, which provides SCC operations on Enterprise Portal items from Visual Studio.

Creating a new project

The following steps explain how you can create a web application project to help jump-start Enterprise Portal development by using Visual Studio.

1. Start Visual Studio.
2. On the File menu, click New Project.
3. In the New Project dialog box, under Installed Templates, select Microsoft Dynamics AX.
4. In the list of templates, select EP Web Application.
5. Provide a name for the application and solution.
6. Add the project to Microsoft Dynamics AX by clicking File > Add <project name> to Dynamics AX.

Editing an existing project

1. Start Visual Studio.
2. Navigate to the Microsoft Dynamics AX 2012 Application Explorer (View > Application Explorer) and to Visual Studio Projects > C # Projects.
3. Locate the project that you need to edit.
4. Right-click the project name, and then click Edit. This will open the project in Visual Studio.
5. When you have finished editing, click the Save button in Visual Studio.

   Note Clicking Save will save the project to the AOT. If the project is an Enterprise Portal project, the web controls are deployed to the SharePoint folder. If the project is a proxy project, it is referenced in Visual Studio, and the proxy files are generated in the development website's App_code\Proxies folder.

Creating a new user control

1. Create a new project, or open an existing one.
2. Add a new item to the project by right-clicking the project in the Solution Explorer pane, pointing to Add, and then clicking New Item.
3. In the Add New Item dialog box, under Installed Templates, select Microsoft Dynamics AX.
4. Select one of the available EP user control templates.
5. Provide a name for the web control.
6. In the Visual Studio installed templates section, select EP User Control.
7. Save the project.

   The web control and web managed content will be added to the AOT. The web control is also deployed to <Drive>:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ep.
8. You can now use the new user control on your Web Part page with the User Control Web Part.

Editing an existing user control

1. Create a new project, or open an existing one.
2. If you have added the web control to the project before, it will be automatically loaded along with the project, and it will be available for editing.
3. If you want to add an existing web control to the project, click View > Application Explorer to open the Application Explorer window.
4. Expand the AOT nodes in Application Explorer, and navigate to Web > Web Files > Web Controls.
5. Select the web control to add to Visual Studio.
6. Right-click the web control node, and then click Add to Project.

   Note If you have multiple projects in your Visual Studio solution, before clicking Add to Project, make sure that the correct project has been selected in Visual Studio Solution Explorer on the right. Click the project name to keep it in the selected state before you click Add to Project.
7. You can now start editing the web control.

Removing a web control

- To remove a web control from your web project but retain it in the AOT, right-click the web control in Solution Explorer, and then click Exclude From Project.
- To remove the web control from the web project and the AOT (in other words, to delete it), right-click the web control in Solution Explorer or in the Microsoft Dynamics AX 2012 Application Explorer, and then click Delete.

General guidelines

- Create one user control, and switch the mode based on the web content parameter. Typically, one control is created for Info, and another one is created for both New and Edit. If your scenario permits, you can create one web control for all the three. If it's used for both New and Edit, set the default to Edit.

```
/// <summary>
/// Returns the mode from the query string
/// 0 = ReadOnly; 1 = Edit; 2 = Insert
/// </summary>
private EPFormAction FormMode
{
    get { return (EPFormAction)Convert.ToInt16(this.Page.Request.QueryString.Get("mode")); }
}

/// <summary>
/// Prepares the form in the proper mode, sets the page title
/// </summary>
private void SetupMode()
{
    switch (this.FormMode)
    {
        case EPFormAction.EditMode:
            this.ActivityForm.DefaultMode = DetailsViewMode.Edit;
            this.ActivityForm.AutoGenerateEditButton = true;
            break;
        case EPFormAction.CreateMode:
            this.ActivityForm.DefaultMode = DetailsViewMode.Insert;
            this.ActivityForm.AutoGenerateInsertButton = true;
            break;
        default:
            this.ActivityForm.DefaultMode = DetailsViewMode.ReadOnly;
            break;
    }
}

/// <summary>
/// Page OnInit event handler
/// </summary>
protected override void OnInit(EventArgs e)
{
    this.SetupMode();
    this.ToggleControlsVisibility();
    base.OnInit(e);
}
```

- Create three web content items that point to one web control (or two, if Info is different from New and Edit) and three URL web menu items (for three separate pages).
- Use the same data set for all three operations.
- Set the modal window properties (set WindowMode to Modal, and set WindowSize as required).
- Ideally, the name of the user control for viewing must end with Info.
- Ideally, the name of the user control for new and edit must end with AddEdit.
- If new and edit are done by using different user controls, the name of the user control used for

new must ideally end with Create, and the name of the user control used for edit must ideally end with Edit.

- The same naming guidelines apply to web managed content and web menu items.
- For the view page, reuse the model-driven info parts from the FactBox area on the list page. To display the FactBox area and the info part in a web control, use the AxPartContentArea control along with the AxInfoPart control.
- Make sure that the user control is connected to the other Web Parts on the page, such as the Title Web Part, the Action Pane/Toolbar Web Part, and the Web Parts in the FactBox area.
- If you want to display FactBoxes in the user control, use the Header, Footer, 3 Columns layout. Set the width of the second Web Part to 30%. If the user control does not have any FactBoxes, use the Header, Left Column, Body layout for the Web Part page.
- Always set the correct label and help text on the managed web content item.
- The WebConfigurationKey property for the managed web content item is generally set to EP, and the ConfigurationKey property is set to the application-specific key.
- For wizard-style pages, use the asp:Wizard control provided by the .NET Framework.

Auto Action Pane buttons

The Auto Action Pane feature automatically adds the buttons for common actions, such as Save and Close, and Close, to the Action Pane.

- If AxForm is in Edit mode and AutoGenerateEditButton = True:
    - Enterprise Portal will add an Edit tab with two buttons: Save and close and Close.
    - If the Action Pane already exists, the two buttons are added as the first buttons.
- If AxForm is in Insert mode and AutoGenerateInsertButton = True:
    - Enterprise Portal will add an Insert tab with two buttons: Save and close and Close.
    - If the Action Pane already exists, the two buttons are added as the first buttons.
- If AxForm is in View mode and AutoGenerateCancelButton = True:
    - Enterprise Portal will add a View tab with one button: Close.
    - If the Action Pane already exists, the Close button is added as the second button.

Embedding user controls

If you have reusable code or UI elements, you can create them as one user control and reuse them in many other user controls by embedding the user control by using the Register directive in the markup.

```
<%@ Register Src="DimensionDefaultingEPController.ascx" TagName="DimensionDefaultingEPController"
    TagPrefix="Dim" %>
<Dim:DimensionDefaultingEPController ID="DimensionDefaultingController" runat="server" />
```

When you are embedding controls, make sure that only one data source control has its Role property set to Provider or ProviderConsumer. Also, under the Permissions node in the AOT, make sure that the embedded control is added under the Associated Web Controls node.

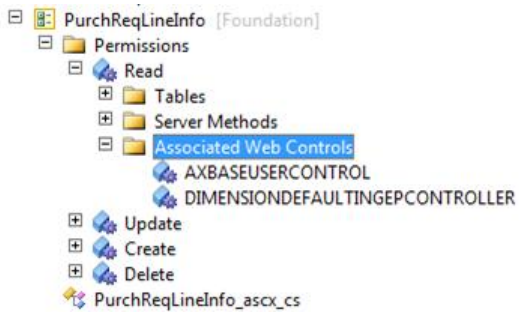


Figure 49 Adding associated web controls to the Permissions node

Modal window properties

By default, all details pages in Enterprise Portal are opened in modal windows. You can use the following properties on the Web URL Menu Item to control the appearance and behavior of modal windows.

WindowMode

- **Inline** Opens the URL in the same browser window.
- **Modal** Open the URL in a new window, if it is invoked from a modal window.
- **NewModal** Opens the URL in a modal window.
- **NewWindow** Opens the URL inline in the existing modal window, if it is invoked from a modal window.

WindowSize

- **Large** 930 x 600
- **Medium** 800 x 600
- **Small** 550 x 412
- **Smallest** 331 x 160
- **Maximum** Depends on the resolution

Note Sizes are defined in Web\Web Files\Static Files\EPAXDialog and are deployed to \layouts\ep\scripts\AXDialog.js.

WindowParameters Custom name-value pairs delimited by commas.

HideActionPane

- **Yes** Hides the Action Pane.
- **No** Renders the Action Pane if the page has one.

CloseDialogBehavior

- **Auto** The default, preferred value. Depending on where it is invoked and the corresponding Action Pane property, this property refreshes the appropriate data source either in the parent form or on the entire page.
- **RefreshDataSource** When the modal window is closed, this refreshes the read-only data source in the parent form (typically the data source that is bound to the toolbar of the line grid). This preserves the current selection in the grid and performs a Research() operation on the data source.
- **RefreshPage** This is useful for modal windows that are invoked from the line grid toolbar.
- **Submit** When the modal window is closed, this refreshes the parent page.
- **None** This just closes the modal window, without doing anything to the parent form.

Modal windows – Interaction patterns

Let's take a look at the various common interaction patterns in Enterprise Portal and how you can achieve them. We will also describe how to update your Microsoft Dynamics AX 2009 user controls to take advantage of these patterns.

List > View > Edit

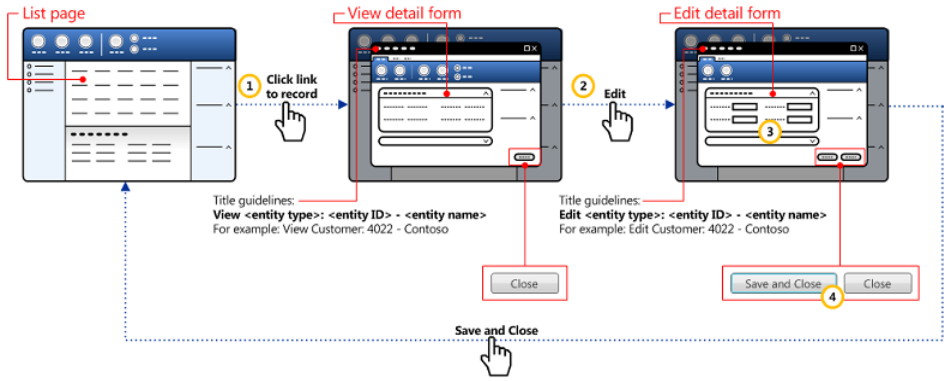


Figure 50 The List > View > Edit pattern

Interaction

1. Click the link or button on the list page to open the View details page.

   The View details page opens in a modal window.

2. Click the button to open the Edit details page.

   The View details modal window closes, and the Edit details page opens in a modal window.

3. Make changes, as needed.

4. Close the modal window.

   The list page is refreshed with the updated data.

Configuration

- For the Web URL menu items corresponding to the View and Edit details pages, set the WindowMode property to Modal.
- For Microsoft Dynamics AX 2009 controls, replace the Response.Redirect code in the code behind with DialogHelper.Close. This will perform a smart redirect, redirecting in the case of a normal browser window or closing the page in the case of a modal window. This will help keep existing redirect behavior if such a URL is opened in a normal browser window by using a bookmark.

Note The list page is refreshed only when it comes from an Edit or New page.

```
/// <summary>
/// Closes the modal window for Add/Edit or
/// redirects the user to the Customer list page if not Modal window
/// </summary>
private void RedirectToListPage()
{
    AxUrlMenuItem customerUrlMenuItem = new AxUrlMenuItem("CustomersList");

    // Response.Redirect(customerUrlMenuItem.Url.OriginalString);
    DialogHelper.Close(customerUrlMenuItem);
}
```

List > Edit

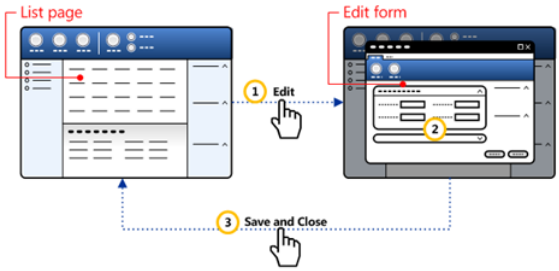


Figure 51 The List > Edit pattern

Interaction

1. Click the link or button on the list page to open the Edit details page.

   The Edit details page opens in a modal window.

2. Make any desired changes.
3. Close the modal window.

   The list page is refreshed with the updated data.

Configuration

- The steps to achieve this are the same as for the List > View > Edit pattern described earlier. The only difference is that there is no intermediate View details page.
- For Microsoft Dynamics AX 2009 controls, replace the Response.Redirect code in the code behind with DialogHelper.Close, as we did for the List > View >Edit pattern.

List > Edit > Sub Edit

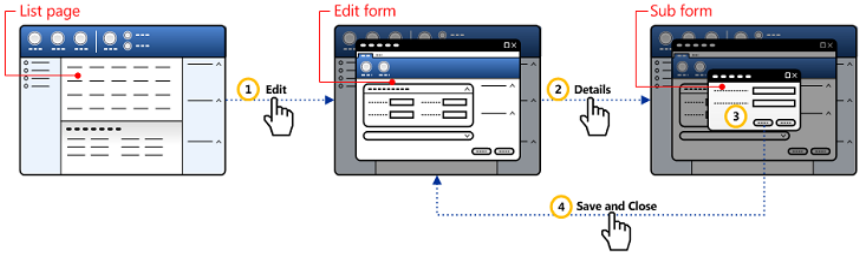


Figure 52 The List > Edit > Sub Edit pattern

Interaction

1. Click the link or button on the list page to open the Edit details page.

   The Edit details page opens in a modal window.

2. Click the button to open the Sub Edit page.

   The Sub Edit page opens in a smaller modal window. The Edit details page remains open in the background.

3. Make any desired changes.

4. Close the modal window for the Sub Edit page.

   The Edit details page is refreshed with the updated data.

Configuration

- For the Web URL menu item corresponding to Edit details page, set the WindowMode property to Modal.
- For the Web URL menu item corresponding to the Sub Edit page, set the WindowMode property to NewModal and the WindowSize property to Small.

Note Clicking Close on the Edit page will not cancel the changes made on the Sub Edit page. These are saved as soon as you click Save and Close on the Sub Edit page.

List > View > Sub View > Sub Edit

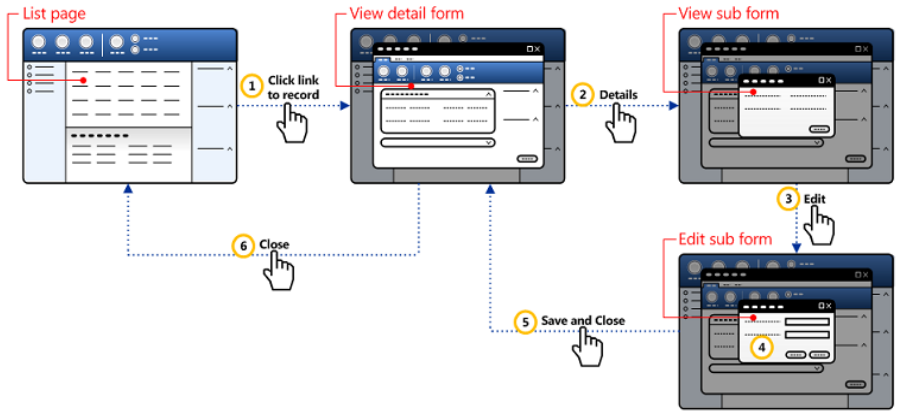


Figure 53 The List > View > Sub View > Sub Edit pattern

Interaction

1. Click the link or button on the list page to open the View details page.

   The View details page opens in a modal window.

2. Click the button to open the Sub View page.

   The Sub View page opens in a smaller modal window. The View details page remains open in the background.

3. Click the button to open the Sub Edit page.

   The Sub View modal window closes, and the Sub Edit page opens in a modal window.

4. Make any desired changes.

5. Close the modal window for the Sub Edit page.

   The View details page is refreshed with the updated data.

6. Close the modal window for the View details page.

   The list page is refreshed with the updated data.

Configuration

- For the Web URL menu item corresponding to View details page, set the WindowMode property to Modal.
- For the Web URL menu item corresponding to the Sub View page, set the WindowMode property to NewModal and the WindowSize property to Small.
- For the Web URL menu item corresponding to the Sub Edit page, set the WindowMode property to Modal and the WindowSize property to Small.
- For Microsoft Dynamics AX 2009 controls, replace the Response.Redirect code in the code behind with DialogHelper.Close, as we did for the List > View >Edit pattern.

List > New > Edit (Two-phase create)

Figure 54 The List > New > Edit (Two-phase create) pattern

Interaction

1. Click the link or button on the list page to open the Phase 1 Header Creation page.

   The Phase 1 Header Creation page opens in a modal window.

2. Enter the required details.

3. Click the button to create the header and open the Phase 2 Edit details page.

   The modal window for the Phase 1 Header Creation page closes, and the Phase 2 Edit details page opens in a modal window.

   You can now add the required details, and then close the modal window for the Phase 2 Edit details page.

   The list page is refreshed with the updated data

Configuration

- For the Web URL menu items corresponding to the Phase 1 Header Creation page and the Phase 2 Edit details page, set the WindowMode property to Modal.
- For the Web URL menu item corresponding to the Phase 1 Header Creation page, you should also set the WindowSize property to Small or Medium, and the ShowActionPane property to No.
- Override the default CloseDialogBehavior value at the point where Phase 1 closes, and pass the menu item for Phase 2, which is set to Modal.

Note Phase 1 and Phase 2 are separate pages and use separate user controls. Each commits its changes on its own.

Example

```
protected void AxForm_VendRequestCompany_ItemInserted(object sender, DetailsViewInsertedEventArgs e)
{
    //check if update is successful, then redirect to appropriate page
    if (e.Exception == null && e.AffectedRows == 1)
    {
        using (IAxaptaRecordAdapter record_Overview = this.CurrentRow.GetRecord())
        {
            AxUrlMenuItem urlMenuItem = new AxUrlMenuItem("VendRequestCompanyRequestEdit");
            urlMenuItem.MenuItemContext = AxTableContext.Create(
                AxTableDataKey.Create(record_Overview, null));
            urlMenuItem.MenuItemType = Proxy.WebMenuItemType.Url;
            DialogHelper.Close(urlMenuItem);
        }
    }
}
```

Submit a form to workflow

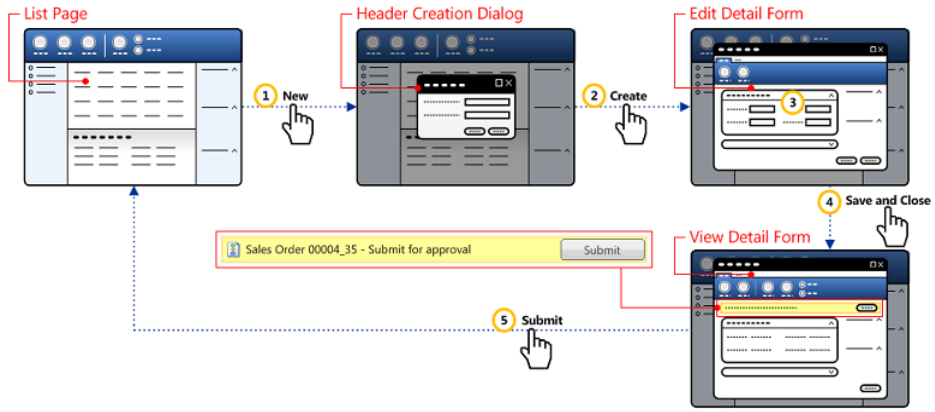


Figure 55 The Submit a form to workflow pattern

Interaction

1. Click the link or button on the list page to open the Phase 1 Header Creation page.

   The Phase 1 Header Creation page opens in a modal window.

2. Enter the required details, and then click the button to create the header and open the Phase 2 Edit details page.

   The modal window for the Phase 1 Header Creation page closes, and the Phase 2 Edit details page opens in a modal window.

3. Add the required details.

4. Click the Save and close button.

   The modal window for the Phase 2 Edit details page closes, and the View Details page opens in a modal window.

5. Click the Submit button in the Workflow bar to initiate the workflow process.

   The list page is refreshed with the updated data.

Configuration

- For the Web URL menu items corresponding to the Phase 1 Header Creation page, the Phase 2 Edit details page, and the View details page, set the WindowMode property to Modal.
- For the Web URL menu item corresponding to the Phase 1 Header Creation page, you should also set the WindowSize property to Small or Medium, and the ShowActionPane property to No.
- Always include the Workflow bar on the View details page and not on the Edit details page.

Modal windows – Programming

The framework provides various APIs for programmatically opening and closing modal windows. This section provides some examples.

Example 1: Closing the modal window by using the default behavior for the parent page (that is, CloseDialogBehavior = Auto)

```
DialogHelper.Close();
```

Example 2: Closing the modal window and opening a new URL menu item. The new menu may be opened in a modal window, inline, or in a new window, depending on its properties.

```
void modalPrompt_ModalDialogClosed(object sender, AXModalDialogClosedEventArgs e)
{
    AxUrlMenuItem axurlEPCSSCheckOut = new AxUrlMenuItem("EPCSSProductGroupPrintGuest");
    DialogHelper.Close(axurlEPCSSCheckOut);
}
```

Example 3: Opening a modal window by using the URL menu item, and also providing context

```
AxUrlMenuItem urlMenuItem = new AxUrlMenuItem("CaseAssociationChange");
urlMenuItem.MenuItemContext = AxTableContext.Create(
    AxTableDataKey.Create(this.AxSession, record, null));
urlMenuItem.MenuItemType = Proxy.WebMenuItemType.Url;

DialogHelper.Open(urlMenuItem, this);
```

Example 4: Opening a modal window by using the URL menu item and dialog args

```
AxUrlMenuItem a = new AxUrlMenuItem("TrvExpTransLineDialog");

a.MenuItemContext = this.ExpenseHeaderTableContext;
AXDialogArgs darg = new AXDialogArgs();
darg.WindowParameters = "{width:350, height:325}";
darg.URL = a.Url.ToString() + "&cat=mileage"; darg.Data = "mileage";

DialogHelper.Open(darg, this);
```

Example 5: Passing data from the parent page to the modal window

```
void TripsToolbar_ActionMenuItemClicked(object sender, ActionMenuItemEventArgs e)
{
    if (e.MenuItem.MenuItemAOTName.ToLower() == "fmtripdetails")
    {
        AxUrlMenuItem urlFMTripDetails = new AxUrlMenuItem("FMTripDetails");

        AXDialogArgs args = new AXDialogArgs();
        args.Data = "Some data";
        args.URL = urlFMTripDetails.Url.ToString();
        args.Size = DialogSize.Medium;

        DialogHelper.Open(args, this);
    }
}
```

Example 6: Getting the data passed from the modal window in the parent page

```
protected void Page_Load(object sender, EventArgs e)
{
    this.WebPart.ModalDialogClosed += new
        EventHandler<AXModalDialogClosedEventArgs>(WebPart_ModalDialogClosed);
}

void WebPartModalDialogClosed(object sender, AXModalDialogClosedEventArgs e)
{
    string strReturnData = e.DialogArgs.Data;
}
```

Example 7: Passing data from the modal window to the parent page

```
DialogHelper.Close(CloseDialogBehavior.Auto, "Returned from the modal");
```

Example 8: Getting the data passed from the parent page in the modal window

```
protected void Page_Load(object sender, EventArgs e)
{
    AXDialogArgs args = DialogHelper.DialogArgs;
    string sData = args.Data.ToString();
}
```

Note You can also use query parameters to pass string data between the parent page and the modal window.

Example 9: Getting the Base Web Part and Managed Content Item

```
//Web part private
AxBaseWebPart _baseWebpart = null;

//Webpart
AxBaseWebPart BaseWebpart
{
    get
    {
        //return the basewebpart if it is not null
        if (this._baseWebpart != null)
            return this._baseWebpart;

        //get the webpart if basewebpart is null
        this._baseWebpart = AxBaseWebPart.GetWebpart(this);
        return this._baseWebpart;
    }
}

//Web part managed content item for the page
Proxy.WebManagedContentItem ContentItem
{
    get
    {
        //get the managed content item from the webpart
        return this.BaseWebpart == null ? null : this.BaseWebpart.WebManagedContentItem;
    }
}
```

Example 10: Calling a static X++ method

```
Object o = AxBaseWebPart.GetWebpart(this).Session.AxaptaAdapter.CallStaticClassMethod(
    "Class1", "Method1");
```

CommonControls.ControlHelper

The ControlHelper class in the Microsoft.Dynamics.Framework.Portal.CommonControls namespace contains some helper methods that can be used in web controls.

CreateMenuItemWithTableContext Creates an AxUrlMenuItem object, with the given data as the table context.

```
AxUrlMenuItem menuItem = ControlHelper.CreateMenuItemWithTableContext(
    CommonConst.TableName.CatExternalCatalog,
    CommonConst.FieldName.RecId,
    externalCatalogRecId,
    CommonConst.MenuItemName.CatProcurement);
```

DecryptWKEY Gets the decrypted value keyed by the WKEY, which is normally the record ID in context, from the query string.

GetRowData Gets data from the given data source's current data set view record.

GetCurrentSessionCompany Gets the current session company in uppercase.

General development

Sessions

All the Web Parts on a webpage share the same session in Microsoft Dynamics AX. After the page is served, the session is disposed of. To optimize performance, you can control the time frame for the disposal of the session. Through settings in the web.config file, you can specify the session time-out, as well as the maximum number of concurrent sessions that are cached.

For example, to set the maximum number of cached concurrent sessions to 300 and the session time-out to 45 seconds, add the <Microsoft.Dynamics> section after </system.web>, as shown in the following code. Remember that an increase in any of these values comes at the cost of more memory consumption.

```
<Microsoft.Dynamics>
    <Session MaxSessions="300" Timeout="15" />
</Microsoft.Dynamics>
```

Note The HttpRunTime.ExecutionTimeout property indicates the maximum number of seconds that a request is allowed to execute before being automatically shut down by ASP.NET. If you want, you can alter this property to improve performance.

```
<httpRuntime executionTimeout="6000" maxRequestLength="51200" />
```

Many of the methods that you use in the Enterprise Portal framework to add code to user controls require access to the Session object. You also need to pass the Session object when using proxy classes. You can access the Session object through the Web Part that hosts the user control, as shown in the following code.

```
AxBaseWebPart webpart = AxBaseWebPart.GetWebpart(this);
return webpart == null ? null : webpart.Session;
```

Windows Server AppFabric

By default, Enterprise Portal uses the ASP.NET session state to store the DataSet's state between postbacks. For more details about the ASP.NET session state, see ASP.NET Session State on MSDN.

You can also configure and use Windows Server AppFabric distributed caching with Enterprise Portal to further improve performance in server farm environments (network load balancing scenarios). Windows Server AppFabric caching features use a cluster of servers that communicate with each other to form a single, unified application cache system. Because it is a distributed cache system, all cache operations are abstracted to a single point of reference, which is referred to as the cache cluster. Your applications can then work with a single logical unit of cache in the cluster, regardless of how many computers make up the cache cluster.

After you have installed and configured Windows Server AppFabric, you can specify the cache name and region that Enterprise Portal uses in the web.config file. For example, to set the cache name as MyCache and the region as MyRegion, add the <Microsoft.Dynamics> section after </system.web> in the web.config file for Enterprise Portal, as shown in the following example.

```
<Microsoft.Dynamics>
    <AppFabricCaching CacheName="MyCache" Region="MyRegion" />
</Microsoft.Dynamics>
```

For details about Windows Server AppFabric, see Windows Server AppFabric Caching Physical Architecture Diagram on MSDN.

Context

Context is a data structure that enables data related to the current environment and to user actions that are taking place to be shared with different parts of a web application. Context lets you know what is happening in one control, so that you can react to that information via another control or a Web Part, or pass the information to a new page. Generally, information about the current record that the user is working on forms the context. For example, when the user selects a row in a grid view, other controls might need to react by getting information about the newly selected row. To pass context, the corresponding data sources and Web Parts must be connected. One data source or Web Part acts as a publisher, and the other data sources or Web Parts act as subscribers.

The following steps explain how to set up the connection between two data source in a user control:

1. On the AxDataSource control that the publishing control binds to, set the Role property to Provider.
2. For the publishing control (for example, an AxGridView control) set the DataKeyNames property so that it contains the key fields that uniquely identify the record being published.
3. On the AxDataSource control that a subscribing control binds to, set Role to Consumer.
4. For a subscribing control (for example, an AxForm control) set the DataKeyNames property so that it contains the key fields that uniquely identify the record being subscribed to. Also set the DataMember property, typically to <View>_Current.

The following steps explain how to set up the connection between two Web Parts on a page:

1. Open the Web Part page in the browser, in edit mode.
2. On the Web Part that contains the publishing control, set Role to Provider.
3. On the Web Part that contains a subscribing control, set Role to Consumer.
4. On the Web Part that contains a subscribing control, click Connections > Get AxContextList From, and then select the Web Part that contains the publishing control.

   Alternatively, on the Web Part that contains a publishing control, you can click Connections > Send AxContextList To, and then select the Web Part that contains the subscribing control.

Note The tables should have a relationship defined at the table level, or extended data types must be set up for the key fields that are used for the AxContextList. These extended data types must have defined relations that link the field to its corresponding table.

AxContext is an abstract class that encapsulates the concept of context. AxTableContext and AxViewContext derive and implement AxContext. AxTableContext is for table-based context, and AxViewContext is for data set view context. A view can contain more than one table; therefore, it contains an AxTableContext object for each table in the view in the TableContextList collection. The RootTableContext property returns the TableContext of the root table in that data set view. AxViewDataKey uniquely identifies AxViewContext, and it contains the TableDataKeys collection. AxTableDataKey uniquely identifies AxTableContext.

An event is raised whenever the context changes. If the context is changed within the web user control, the CurrentContextChanged event is raised. If the context changes from other Web Parts that are connected to it, the ExternalContextChanged event is raised. You can write code in these events from the AxBaseWebPart in your web user control, and use the CurrentContextProviderView or ExternalContextProviderView and ExternalRecord properties to get the record associated with the context. You can also fire these events programmatically from your application logic by calling FireCurrentContextChanged or FireExternalContextChanged, so that all other connected controls can react to the change that you made through your code.

To get the context, you can use the AxContextHelper.FindIAxContext method.

```
IAxContext context = AxContextHelper.FindIAxContext(this);
```

The following sample code is used to fire the CurrentContextChanged event.

```
void CurrentContextProviderView_ListChanged(object sender,
    System.ComponentModel.ListChangedEventArgs e)
{
    AxBaseWebPart webpart = this.WebPart;

    // The current row (which is the current context) has changed - update the consumer webparts.
    // Fire the current context change event to refresh (re-execute the query) the consumer webparts
    webpart.FireCurrentContextChanged();
}
```

The following sample code is used to get the record context from the connected Web Part. First subscribe to the ExternalContextChanged event in the consumer web user control, as shown here.

```
protected void Page_Load(object sender, EventArgs e)
{
    // Add Event handler for the ExternalContextChange event.
    // Whenever the selecting the grid of the provider webpart changes, this event will get fired.
    (AxBaseWebPart.GetWebpart(this)).ExternalContextChanged += new
        EventHandler<Microsoft.Dynamics.Framework.Portal.UI.AxExternalContextChangedEventArgs>(
            AxContextConsumer_ExternalContextChanged);
}
```

Next, get the record passed through the external context, as shown here.

```
void AxContextConsumer_ExternalContextChanged(object sender,
    Microsoft.Dynamics.Framework.Portal.UI.AxExternalContextChangedEventArgs e)
{
    // Get the AxTableContext from the ExternalContext passed through Web part connection
    // Construct the record object and get to the value of the fields
    IAxaptaRecordAdapter currentRecord = (AxBaseWebPart.GetWebpart(this)).ExternalRecord;
    {
        if (currentRecord != null)
        {
            lblCustomer.Text = (string)currentRecord.GetField("Name");
        }
    }
}
```

In some cases, you might want to refresh the consumer Web Part from the provider Web Part, even if the context is not changing.

For example, UserControl1 is a grid of header rows (for example, a Rental header) and acts as the provider. This user control also contains an action button. UserControl2 is a grid of line rows (for example, Rental lines) and acts as the consumer. In UserControl 1, when the action button is clicked, you are calling an X++ method that inserts a record into the Rental lines table through business logic. You want to refresh UserControl2, even though the context has not changed.

In this case, in the button click event, you can explicitly fire CurrentContextChanged so that the consumer Web Part will be reloaded. Remember to connect the user controls and Web Parts as outlined earlier.

```
Microsoft.Dynamics.Framework.Portal.UI.IAxContext contextInterface =
    Microsoft.Dynamics.Framework.Portal.UI.AxContextHelper.FindIAxContext(this);

contextInterface.FireCurrentContextChanged();
```

For more information about context, see http://msdn.microsoft.com/en-us/library/ee330234.aspx.

Data

The Enterprise Portal ASP.NET controls access and manipulate data through data binding to AxDataSource. You can also access the data directly through the APIs. The Microsoft.Dynamics.AX.Framework.Portal.Data namespace contains several classes that work together to retrieve data.

Example 1: Getting the current row

```
private DataSetViewRow CurrentRow
{
    get
    {
        return
            this.AXDataSource1.GetDataSet().DataSetViews[this.AXGridView1.DataMember].GetCurrent();
    }
}
```

Example 2: Getting the current record

```
internal IAxaptaRecordAdapter GetCurrentDataSourceRecord()
{
    IAxaptaRecordAdapter record = null;

    if (this.DataSourceView.DataSetView != null &&
        this.DataSourceView.DataSetView.GetCurrent() != null)
    {
        //setting the record as the current record in the datasource
        record = this.DataSourceView.DataSetView.GetCurrent().GetRecord();
    }

    return record;
}
```

Example 3: Getting a field value

```
DataSetViewRow dataRow = this.CurrentRow;
Int I = (int)(dataRow.GetFieldValue("MappedToDispute**"));
```

Note The IAxaptaRecordAdapter interface provides is another method that you can use, GetField. However, GetFieldValue, which is defined on the DataSetViewRow, looks at the database cache and will generally perform better.

Example 4: Calling a method on the current record

```
private bool IsLineHasActiveWorkflowTrackingStatus()
{
    IAxaptaRecordAdapter recAdapter = this.CurrentRecord;
    return recAdapter != null && (bool)recAdapter.Call("hasActiveWorkflowStatus");
}
```

Example 5: Passing the record context to a menu item

```
using (IAxaptaRecordAdapter record = this.CurrentRow.GetRecord())
{
    AxTableContext context = AxTableContext.Create(AxTableDataKey.Create(AxSession, record, null));
    ((AxUrlMenuItem) e.MenuItem).MenuItemContext = context;
}
```

For more information about data, see http://msdn.microsoft.com/en-us/library/cc624200.aspx.

Metadata

The Enterprise Portal framework provides a rich set of APIs for accessing the metadata from the AOT in managed code. The Microsoft.Dynamics.AX.Framework.Services.Client namespace contains several classes that work together to retrieve metadata from the AOT. Enterprise Portal controls use the metadata to retrieve formatting, validation, security, and other information from the AOT, and then automatically apply the information in the user interface. Developers can also use these APIs to retrieve the metadata in their user interface logic. MetadataCache is the main entry point for accessing metadata and provides static methods.

Add the following using statement to the code behind.

```
using Microsoft.Dynamics.AX.Framework.Services.Client;
```

Call the methods in the MetadataCache object. The following code sample gets the web menu item metadata.

```
ApplicationProxy.CatCartLine record = ApplicationProxy.CatCartLine.find(recId);
ISession session = AxBaseWebPart.GetWebpart(this).Session;
WebLink webLink = new WebLink(session as ILinkContext);

URLWebMenuItemMetadata menuItemMetadata =
    MetadataCache.GetURLWebMenuItemMetadata(CommonConst.MenuItemName.CatCategoryRequest);

// Set menu item
webLink.SetWebMenuItemInfo(
new WebUrlLinkMenuItemInfo(
    menuItemMetadata.Name,
    menuItemMetadata.GetLabel(session.SessionInfo),
    menuItemMetadata.GetHelpText(session.SessionInfo),
    menuItemMetadata.URL));

// Set data properties
webLink.RecordAsMILRecord = record;

// Return URL
return webLink.Url(false, false);
```

The following code sample gets the table metadata.

```
tableMetaData = MetadataCache.GetTableMetadata(TableMetadata.TableNum("SMAAgreementTable"));
```

The following code sample gets the enum metadata.

```
EnumMetadata approvalEnum = MetadataCache.GetEnumMetadata(EnumMetadata.EnumNum("HRMGoalApproval"));

foreach (EnumEntryMetadata entry in approvalEnum.EnumEntries)
{
    item = new ListItem(entry.GetLabel(this.AxSession.SessionInfo), entry.Value.ToString());
    ddlApproval.Items.Add(item);
}
```

Changing metadata at run time

You can change field properties through X++ or managed code, and the UI will react to the changes. For example, you can change the mandatory value of a field in X++ code; or, in your code behind, you can get the data source field metadata and set the mandatory value. The bound field will automatically respond to the change. We support changing data source and field properties, as well as adding data sources at run time.

To change the AllowEdit property on the salesTable data source, override a data set method, and use the following code.

```
salesTable_ds.allowEdit(true);
```

To change the Mandatory property on the CustAccount field, use the following code.

```
salesTable _ds.object(fieldNum(SalesTable, CustAccount)).mandatory(true);
```

If you are overriding a method of the data source for which you want to change a property, you can also use this instead of the dataSourceName_ds.

```
// If this code is being written in one of the override methods on the SalesTable data source
// You can use "this" instead of salesTable_ds
this.allowEdit(true);
this.object(fieldNum(SalesTable, CustAccount)).mandatory(true);
```

To change the properties from managed code, use the following code.

```
// Set AllowEdit property on the data source
dataSetView.Metadata.DataSources["SalesTable"].AllowEdit = false;

// Set Mandatory property on the CustAccount field
dataSetView.Metadata.DataSources["SalesTable"].DataSourceFields["CustAccount"].Mandatory = false;
```

You can also listen to metadata changed events and write your own handler.

```
this.CaseAddEditDS.Metadata.DataSetViews[0].DataSources[0].DataSourceFields["Process"].
    AllowEdit = false;



this.DS_TrvExpTrans.GetDataSet().DataSetViews[0].Metadata.DataSources["TrvExpTrans"].
    DataSourceFields["Location"].Mandatory = true;
```

For more information about metadata, see Metadata on MSDN.

Pages

Page definition properties

PageTitle Set to a label, generally <Action><Entity>—for example, New Customer.

UseContext Generally, set to true for details pages that require a record context in the query string.

ParentPage For task pages, set to a list page. This property is used by the breadcrumb control in Enterprise Portal.

Events

OnInit The OnInit event performs the initialization and setup steps required to create a Page instance. In this stage of the page's life cycle, declared server controls on the page are initialized to their default state; however, the view state of each control is not yet populated. One control on the page cannot access other server controls on the page during the Page_Init phase, regardless of whether the other controls are child or parent controls. Other server controls are not guaranteed to be created and ready for access. The OnInit event is called after the OnPreInit method and before the OnInitComplete method.

```
protected override void OnInit(EventArgs e)
{
    this.SetupMode();
    this.ToggleControlsVisibility();
    base.OnInit(e);
}
```

Init The Init event is raised after all controls have been initialized. The Init event of individual controls occurs before the Init event of the page. Use this event to read or initialize control properties.

```
protected void Page_Init(object sender, EventArgs e)
{
    if (this.FormMode == EPFormAction.CreateMode)
    {
        this.ActivityDS.CreatingDataSetRun += new
            EventHandler<CreatingDataSetRunEventArgs>(ActivityDS_CreatingDataSetRun);
    }
}



void Page_Init(object sender, EventArgs e)
{
    gridEPSalesLineEdit.RowCancelingEdit += new
        GridViewCancelEditEventHandler(gridEPSalesLineEdit_RowCancelingEdit);

    formEPSalesTableEdit.ItemUpdating += new
        DetailsViewUpdateEventHandler(formEPSalesTableEdit_ItemUpdating);

    BaseWebpart.ModalDialogClosed += new
        EventHandler<AXModalDialogClosedEventArgs>(BaseWebpart_ModalDialogClosed);
}
```

PreRender The PreRender event is raised after the Page object has created all controls that are required to render the page, including child controls of composite controls. To do this, the Page object calls EnsureChild-Controls for each control and for the page.

The Page object raises the PreRender event on itself, and then recursively raises it on each child control. The PreRender event of individual controls occurs after the PreRender event of the page. Use this event to make final changes to the contents of the page or its controls before the rendering stage begins.

```
protected void Page_PreRender(object sender, EventArgs e)
{
    if (performDataBindingForTransactionDetailsForm)
    {
        this.Form_TransactionDetails.DataBind();
    }

    this.reportLineEditable = (this.CurrentRow == null) ? true :
         (bool)this.CurrentRow.GetFieldValue("isEditable**");

    this.GridView_TrvExpTrans.AllowEdit = this.IsGridAllowEdit();

    if (!this.IsPostBack)
    {
        int selIndex = this.SelectedExpenseLine;

        if (this.GridView_TrvExpTrans.Rows.Count > selIndex)
        {
            this.GridView_TrvExpTrans.SelectedIndex = selIndex;
            this.UpdateTransactionDetailsFields();
            this.UpdateDetailsFormsMode(DetailsViewMode.ReadOnly);

            if (this.reportEditable)
                this.GridView_TrvExpTrans.EditIndex = -1;
        }
    }
    else
    {
        if (this.CurrentRow != null)
        {
            this.SelectedExpenseLine = this.CurrentRow.RowIndex;
        }
    }

    if (!this.Page.IsPostBack)
    {
        this.InitializeAdminCustomFieldsOnGrid();
    }

    if (performDataBinding)
    {
        this.AxForm_ExpenseHeader.DataBind();
    }

    this.PreRenderHeaderDimensions();
}
```

Load The Page object calls the OnLoad method on itself, and then recursively calls it on each child control, until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page. Use the OnLoad method to set properties in controls and establish database connections.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (this.ActiveSigningLimits.GetDataSet().DataSetViews[this.gvActiveLimits.DataMember].Count
        <= 0)
    {
        gvActiveLimits.Visible = false;
    }
}
```

Page titles

To specify a page title, you must be using the Page Title Web Part on the page. Connect the Page Title Web Part to your user control if the user control will be providing the title to the page. If the Page Title Web Part is not connected, the title will come from the page definition metadata (PageTitle, UseContext) and the record context through the query string.

The Page Title Web Part will pick up two different items from the user control:

- **Caption** The initial text that will always be displayed.
- **Context** Information that is specific to a record.

Four properties from ITitleProvider control what is displayed in the page title:

- **Caption** Sets the caption text directly. By default, it uses the TitleField1 and TitleField2 properties defined on the table, and is set to TitleField1Label: TitleField1Value, TitleField2Value. If TitleField1 or TitleField2 is a surrogate foreign key (SFK), it will be replaced with its AutoIdentification field group.
- **ShowContext** Determines whether the page title displays any available context.
- **MenuItem** A URL that encompasses the entire context.
- **View** The DataSetView from which to retrieve the current context (record).

```
Protected void Page_Load(object sender, EventArgs e)
{
    ITitleProvider titleProvider = AxBaseWebPart.GetWebPart(this) as ITitleProvider;

    // If Caption is not provided, nothing will be shown. Basically, it can't be null or empty
    titleProvider.Caption = "Caption";

    // Determines if the context will link to any pages
    titleProvider.MenuItem = new AxUrlMenuItem("UrlMenuItem");

    // Determines if the context will be shown
    titleProvider.ShowContext = false; // true by default

    // Determines which data set view will provide the context
    titleProvider.View = this.AxDataSource1.GetDataSet().DataSetViews[0];
}
```

For model-driven list pages, the page title is set by either the code in the interaction class (this.listPage().caption('Title')) or the corresponding Display Menu Item's label.

For more information about how to connect Help content to pages, see How to: Connect Help for Pages on MSDN.

Custom filters

Creating custom filters in model-driven list pages

You can create custom filters to help users quickly apply frequently used filter conditions.

Figure 56 A custom filter

To create a custom filter for a model-driven list page, follow these steps:

1. Create an enum and an EDT, with All as one of the options. For example, for Vehicle Status, the enum can be called FMVehicleStatusFilter, and it can have the values All, Assigned, Available.
2. Add a new method to the SysQueryRangeUtil class that returns an empty string for All and a proper value for the other options.

```
public static str fmVehicleStatusFilter(anytype _fmVehicleStatusFilterEDT)
{
    FMVehicleStatusFilterEDT fmVehicleStatusFilterEDT = -fmVehicleStatusFilterEDT;

    switch(fmVehicleStatusFilterEDT)
    {
        case FMVehicleStatusFilter::Assigned:
            return SysQuery::value(FMVehicleStatusFilter::Assigned);
        case FMVehicleStatusFilter::Assigned:
            return SysQuery::value(FMVehicleStatusFilter::Available);
        case FMVehicleStatusFilter:: All:
             return SysQuery::valueUnlimited();
        default:
             return SysQuery::valueUnlimited();
    }
}
```

Instead of using the switch statement, you can also take the following approach.

```
public static str tutorialStatusAll(tutorialOrderStatusAll _tutorialStatusAll)
{
    if (_tutorialStatusAll == tutorialOrderStatusAll::All)
    {
        return SysQuery::valueUnlimited();
    }
    else
    {
        return SysQuery::value(_tutorialStatusAll);
    }
}
```

3. Under the Design node for a form, create a new filter (ComboBox in this example).

4. Instead of setting the FilterExpression property to a value, set it to the function that you created earlier.

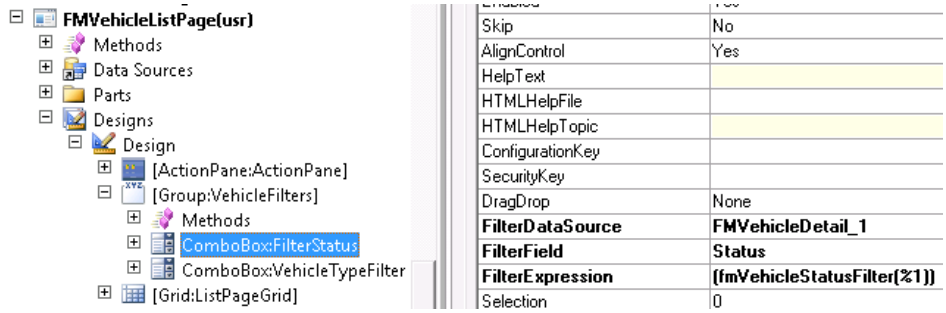


Figure 57 Setting the FilterExpression property

5. To persist the values of the custom filter per user across sessions, set the SaveFilter property of the Group:Filter control to Yes.

```
public void initializing()
{
    SysEPCustomFilter customFilter;
    super();
    caption = this.listPage().listPageArgs().parameters();
    customFilter = SysEPCustomFilter::construct(formStr(tutorialOrderForm));
    customFilter.load();
    customFilter.setInitialFilterControlValue(formControlStr(tutorialOrderForm,ComboBoxStatus),
        tutorialOrderStatusAll::All);
    customFilter.save();
}
```

Creating an advanced filter via a user control

For more advanced filtering requirements in Enterprise Portal, you can build a user control that just renders the filter, and add it to the webpage. When the selection changes, you can pass the selected value to the list page through a query string parameter named WP. In the ListPageInteraction class, you can read the value of the WP query string parameter by using this.listPage().listPageArgs().parameters() and filter the records.

Example

In the user control

```
private void RedirectWithSelectedWorker(string worker)
{
    AxUrlMenuItem urlMenuItem;
    urlMenuItem = new AxUrlMenuItem(ListPageNames.ExpenseReports);
    urlMenuItem.ExtraParams.Add("WP", worker);
    HttpContext.Current.ApplicationInstance.CompleteRequest();
    Response.Redirect(urlMenuItem.Url.OriginalString, false);
}

protected void Dropdown_EnterExpenseReportFor_SelectedIndexChanged(object sender, EventArgs e)
{
    this.Session_Worker = Dropdown_EnterExpenseReportFor.SelectedValue;
    this.RedirectWithSelectedWorker(this.Session_Worker);
}
```

In the list page interaction class

```
public void initialized()
{
    HCMWorkerRecId  worker;
    super();
    worker = this.getWorker();
    this.listPage().actionPaneControlParameters(
        formcontrolstr(TrvExpenseReportsWorkerListPage, NewExpenseReport),
            int642str(worker));
}

public void initializeQuery(Query _query)
{
    TrvHcmWorkerRecId parmWorker = this.getWorker();
    TrvListPageHelper::addWorkerRange(_query, parmWorker);
    TrvListPageHelper::addCurrentCompanyRange(_query);
    TrvListPageHelper::removeDraftDocumentsForExpiredDelegate(_query, parmWorker);
    super(_query);
}

private HcmWorkerRecId getWorker()
{
    if (this.listPage().listPageArgs().parameters())
    {
        return str2int64(this.listPage().listPageArgs().parameters());
    }
    else
    {
        return Global::currentWorker();
    }
}
```

Creating a custom filter in a non-modeled list page by modifying the data set

1. Add QueryBuildRange to the declaration.

```
public class DataSetRun extends ObjectRun
{
    QueryBuildRange qbrStateId;
}
```

2. Override the init() method of the data source, and add the range to the query.

```
public void init()
{
    super();

    qbrStateId = this.query().dataSourceTable(tablenum(AddressCounty))
        .addRange(fieldnum(AddressCounty, StateId));
}
```

3. Create a method on the data source to add the range.

```
void SetFilterOnStateId(AddressStateId _stateId)
{
    qbrStateId.value(QueryValue(_stateId));
}
```

4. Create a method on the data set that can be called from ASP.NET, and that can call the method in the data source. Also re-execute the query.

```
void SetFilterOnStateId(AddressStateId _StateId)
{
    AddressCounty_ds.SetFilterOnStateId(_StateId);
    AddressCounty_ds.executeQuery();
}
```

5. Call the method in the data set from ASP.NET—for example, from the Button_Clicked event.

```
protected void btnSelectStateID_Click(object sender, EventArgs e)
{
    // Call filter method in Ax-DataSet
    dsCaseCustFilter.GetDataSet().DataSetRun.AxaptaObjectAdapter.Call(
        "SetFilterOnStateId", txtSelectStateID.Text);
}
```

Creating a custom filter in a non-modeled list page without modifying the data set

The following sample code shows how you can create a custom filter directly from ASP.NET. This can be in any event, such as Button_Clicked.

```
protected void btnSelectStateID_Click(object sender, EventArgs e)
{
    DataSetView dsv = this.axDsIPICContract.GetDataSet().DataSetViews[0];
    dsv.UserFilter.ClearOpenFilter();
    Microsoft.Dynamics.Framework.Data.Ax.filterObject flt = new filterObject();
    flt.name = "ContractFilter";

    Microsoft.Dynamics.Framework.Data.Ax.conditionType typ = new conditionType();
    typ.__operator = Microsoft.Dynamics.Framework.Data.Ax.operatorType.eq;
    typ.status = Microsoft.Dynamics.Framework.Data.Ax.conditionStatus.open;
    typ.value = this.tbCompanyID.Text;
    typ.attribute = "CompanyId";
    flt.conditionCollection.Add(typ);

    dsv.UserFilter.Add(flt);
}
```

Custom lookup

Creating a custom lookup in managed code

The following code sample shows how you can create a custom lookup in managed code.

```
protected void Name_Lookup(object sender, AxLookupEventArgs e)
{
    AxLookup nameLookup = e.LookupControl;
    string partyName = "*";

    try
    {
        // We set the lookup range based on either the name selected or "*"
        // We get the range value from the current view field value
        partyName = this.DataSourceView.DataSetView.GetCurrent().GetFieldValue(
            "DirPartyTable!Name").ToString();

        // Create the lookup dataset - we will do a lookup in the DirPartyTable table
        using (Proxy.SysDataSetBuilder sysDataSetBuilder =
            Proxy.SysDataSetBuilder.constructLookupDataSet(this.AxSession.AxaptaAdapter,
                TableMetadata.TableNum(this.AxSession, "DirPartyTable")))
        {
            // Set the run time generated data set as the lookup data set
            nameLookup.LookupDataSet = new DataSet(this.AxSession, sysDataSetBuilder.toDataSet());
        }

        // DataSet has to be init'ed before accessing the data sources
        nameLookup.LookupDataSet.Init();

        // Filter the lookup
        using (Proxy.Query query = nameLookup.LookupDataSet.DataSetViews[0].MasterDataSource.query())
        {
            // If the partyName is blank we assign the "*" as the range
            query.dataSourceNo(1).addRange(TableDataFieldMetadata.FieldNum(
                this.AxSession, "DirPartyTable", "Name")).value =
                String.IsNullOrEmpty(partyName) ? "*" : "*" + partyName + "*";
        }

        // Specify the lookup fields used
        nameLookup.Fields.Add(AxBoundFieldFactory.Create(this.AxSession,
            nameLookup.LookupDataSetViewMetadata.ViewFields["EntityType"]));
        nameLookup.Fields.Add(AxBoundFieldFactory.Create(this.AxSession,
            nameLookup.LookupDataSetViewMetadata.ViewFields["Type**"]));
        nameLookup.Fields.Add(AxBoundFieldFactory.Create(this.AxSession,
            nameLookup.LookupDataSetViewMetadata.ViewFields["Name"]));
```

```
        // Specify the select field
        nameLookup.SelectField = "RecId";
    }
    catch (System.Exception ex)
    {
        AxExceptionCategory exceptionCategory;
        // This returns true if the exception can be handled here
        if (!AxControlExceptionHandler.TryHandleException(this, ex, out exceptionCategory))
        {
            // The exception is system fatal - in this case we re-throw.
            throw;
        }
    }
}
```

Creating a custom lookup in X++

The following code sample shows how you can create a custom lookup in X++.

```
void dataSetLookup(SysDataSetLookup sysDataSetLookup)
{
    List list;
    Query query;

    args = new Args();
    list = new List(Types::String);
    list.addEnd(fieldstr(ProjTable, ProjId));
    list.addEnd(fieldstr(ProjTable, Name));
    list.addEnd(fieldstr(ProjTable, Status));
    sysDataSetLookup.parmLookupFields(list);
    sysDataSetLookup.parmSelectField('ProjId');

    // Call query
    query = projTimeSheet_ds.getProjectIDQuery();

    // Pass the query to SysDataSetLookup so it result is rendered in the lookup page.
    sysDataSetLookup.parmQuery(query);
}
```

Getting the selected row

To get the currently selected row from a lookup, you should set the RunLookupDataSetWhenOkClicked property to true on the lookup control, and then in the OKClicked event, you can get the current row, as shown in the following code sample.

```
DataSetViewRow row = ((IAxDataBoundControl)e.LookupControl.DefaultLookupGrid).View.GetCurrent();
```

AutoPostback when OK is clicked

If you need the OKClicked event to do an AutoPostback on the calling page, set the AutoPostback property to true. (Generally, it is set to false.) Additionally, if clicking OK needs to trigger a validation event on the calling page, set the CausesValidation property to true.

```
protected void Name_Lookup(object sender, AxLookupEventArgs e)
{
    AxBoundField field = (AxBoundField)sender;
    AxLookup lookup = e.LookupControl;
    ISession session = AxBaseWebPart.GetWebpart(this).Session;
    lookup.LookupDataSet = new Microsoft.Dynamics.Framework.Data.Ax.DataSet(
        this.AxSession, "VendRequestContactParty");

    // Get the query that filters the contact party name lookup by the vendor party contact relationship
    ApplicationProxy.Query appQuery = VendRequest.webLookupContactPartyQuery(
    (long)this.CurrentRow["VendParty"]);

    lookup.LookupDataSet.DataSetViews[0].MasterDataSource.query(
        new Proxy.Query(this.AxSession.AxaptaAdapter,
        ApplicationProxyHelper.CastToObjectAdapter(this.AxSession, appQuery)));

    lookup.LookupDataSet.Init();

    lookup.Fields.Add(AxBoundFieldFactory.Create(
        lookup.LookupDataSetViewMetadata.ViewFields["FirstName"]));

    lookup.Fields.Add(AxBoundFieldFactory.Create(
        lookup.LookupDataSetViewMetadata.ViewFields["MiddleName"]));

    lookup.Fields.Add(AxBoundFieldFactory.Create(
        lookup.LookupDataSetViewMetadata.ViewFields["LastName"]));

    // Remove the table name and ! char from the select field
    lookup.SelectField = field.DataField.Substring(field.DataField.IndexOf('!') + 1);

    lookup.RunLookupDataSetWhenOkClicked = true;
    lookup.AutoPostBack = true;
    lookup.CausesValidation = false;
    lookup.OkClicked += new EventHandler<AxLookupEventArgs>(Lookup_OkClicked);
}

protected void Lookup_OkClicked(object sender, AxLookupEventArgs e)
{
    DataSetViewRow row = ((IAxDataBoundControl)e.LookupControl.DefaultLookupGrid).View.GetCurrent();

    if (row != null)
    {
        this.CurrentRow["ContactName!FirstName"] = row["FirstName"];
        this.CurrentRow["ContactName!MiddleName"] = row["MiddleName"];
        this.CurrentRow["ContactName!LastName"] = row["LastName"];
    }
}
```

Proxies

If you need to call X++ classes, table methods, or enumerations in your web user control, the Enterprise Portal framework provides an easy way to create a managed wrapper for these X++ objects and use it in your web user control. A proxy file internally wraps the Microsoft .NET Business Connector calls and provides a simple, easy-to-use, typed interface for C# applications.

Creating a new proxy project

1. Start Visual Studio, and click File > New Project.
2. In the project types window, select Visual C#.
3. In the Visual Studio installed templates section, select Class Library.
4. Click File > Add <project name> to Dynamics AX.
5. In Solution Explorer, select the project, and set the Deploy to EP property of the project to Proxies.

   Note Setting Deploy to EP to Yes will deploy the Enterprise Portal as an assembly. For a proxy project, we recommend that you set it to Proxies.
6. In Solution Explorer, right-click the project, click Properties, and then set the Default namespace property to Microsoft.Dynamics.Portal.Application.Proxy.
7. Delete the Class1.cs file that comes with the ClassLibrary project.
8. Click View > Application Explorer, right-click the object for which you need a proxy, such as tables, EDTs, enums, or classes, and then click Add to Project.
9. Save this project, and then add a reference to this project in your Enterprise Portal web control project.
10. In your web control, add a reference to the Microsoft.Dynamics.Portal.Application.Proxy namespace, and start using proxies.

Example

```
using ApplicationProxy = Microsoft.Dynamics.Portal.Application.Proxy;
…
…
if (this.AxQueryString != null && this.AxQueryString.RecordContext != null)
{
    IEnumerator<KeyValuePair<int, object>> en =
        this.AxQueryString.RecordContext.DataKey.GetEnumerator();

    int fieldId = this.AxQueryString.RecordContext.DataKey.Table.Fields.GetByName("RecId").FieldId;

    ApplicationProxy.smmActivityCategory categoryValue =
        ApplicationProxy.smmActivityCategory.Appointment;

    while (en.MoveNext())
    {
        if (en.Current.Key == fieldId)
        {
            Int64 recId = Convert.ToInt64(en.Current.Value);
            ApplicationProxy.smmActivities activityTable =
                ApplicationProxy.smmActivities.findWithRecId(recId);

            categoryValue = activityTable.Category;
            break;
        }
    }
}
```

Tip Use the ApplicationProxyHelper class to cast an application proxy that you generated with Visual Studio Tools to kernel proxies.

```
lookup.ExtendedDataType = smmActivityParentLink.getExtendedTypeId(
    ApplicationProxyHelper.CastToRecordProxy<smmActivityParentLinkTable>(linkTable));
```

Proxy deployment

When you deploy Enterprise Portal, all the elements from AOT > Visual Studio Projects > C Sharp Projects that have their Deploy to EP property set to Proxies are deployed to the file system—for example, C:\inetpub\wwwroot\wss\VirtualDirectories\80\App_Code\Proxies.

During development, when you save your Enterprise Portal Web application project in Visual Studio, all referenced Visual Studio projects, including the proxy projects that are referenced in the solution, are deployed as well. Alternatively, you can deploy the proxy files from the Manage Deployment dialog box in Enterprise Portal or use the AXupdateportal command-line utility.

General guidelines

- With this new add-in, you do not need to edit the proxy file under Web\Web files\Static files to create a proxy file so that you can use your X++ objects from the user control.
- You can create your own proxy project and add the items that you need. In this way, there won't be any source control check-in conflicts with other developers.
- We recommend that you not include more than approximately 20 items in one proxy project. Editing performance in Visual Studio degrades as the number of items increases. Instead, split the items among multiple projects if you have a large number of proxies.
- We have moved the existing proxy definition to the AOT\Visual Studio Projects\C Sharp Projects\EPApplicationProxies and EPApplicationProxies1 projects.
- For new proxies, you can add a project to the EPApplicationProxies1 project or create a new one.

Tip For better performance, and to avoid going back and forth from managed code to X++ through proxies, write managed code whenever possible.

For more information about proxies, see Proxies on MSDN.

Formatting

If you're not using data-bound controls and want your unbound ASP.NET controls to be formatted just like Enterprise Portal controls, you can leverage the AxValueFormatter class in the Enterprise Portal framework. This class implements ICustomFormatter and the IFormatProvider interface. It also defines a method that supports custom, user-defined formatting of an object's value and provides a mechanism for retrieving an object to control formatting. For the various data types, specific ValueFormatter classes derived from AxValueFormatter are implemented: AxStringValueFormatter, AxDateValueFormatter, AxDateTimeValueFormatter, AxTimeValueFormatter, AxRealValueFormatter, AxNumberValueFormatter, AxGuidValueFormatter, and AxEnumValueFormatter.

You use AxValueFormatterFactory to create AxValueFormatter objects. You can create any of the preceding formatters, or you can create a formatter that is based on an EDT in Microsoft Dynamics AX. The data type for the extended data is retrieved from the metadata object for the EDT, and the culture information comes from the context. The various rules for languages and countries, such as number formats, currency symbols, and sort orders, are aggregated into a number of standard cultures. The Enterprise Portal framework identifies the culture based on the user's language setting in Microsoft Dynamics AX and makes this information available in the context. Formatter objects have a Parse method that you can use to convert a string value back into the underlying data type. For example, the following code formats the data based on a given EDT.

```
private string ToEDTFormattedString(object data, string edtDataType)
{
    ExtendedDataTypeMetadata edtType = MetadataCache.GetExtendedDataTypeMetadata(
        this.AxSession, ExtendedDataTypeMetadata.TypeNum(this.AxSession, edtDataType));

    IAxContext context = AxContextHelper.FindIAxContext(this);

    AxValueFormatter valueFormatter = AxValueFormatterFactory.CreateFormatter(
        this.AxSession, edtType, context.CultureInfo);

    return valueFormatter.FormatValue(data);
}
```

For more information about formatting, see Formatting Unbound Data on MSDN.

Validation

You use ASP.NET validator controls to validate user input on the server. You can optionally validate the input on the client (browser) as well. The Enterprise Portal framework has ASP.NET validators that are specific to Microsoft Dynamics AX. AxBaseValidator derives from System.Web.UI.WebControls.BaseValidator, and AxValueFormatValidator derives from AxBaseValidator. Both are metadata-driven and are used intrinsically by bound fields. You can also use them in unbound scenarios.

ASP.NET validators are automatically validated when a postback that causes validation occurs. For example, an ASP.NET Button control causes validation on the client and the server when clicked. All validators that are registered on the page are validated. If any validator is found to be invalid, the page becomes invalid, and Page.IsValid returns false.

Example 1

```
<asp:TextBox ID="NoOfNights" runat="server" Style="text-align: right" CssClass="AxInputField">
</asp:TextBox>

<asp:CustomValidator ID="numberNightsValidator" runat="Server" ControlToValidate="NoOfNights"
    OnServerValidate="ValidateNumberOfNights" ValidationGroup="itemize" Display="Dynamic" />
```

```
protected void Page_PreRender(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        this.numberNightsValidator.ErrorMessage = String.Format(
            CultureInfo.InvariantCulture, Labels.GetLabel("@SYS110666"), MAX_DAYS);
    }
}
```

Example 2

```
    <asp:Label ID="lblExpirationDate" runat="server" AssociatedControlID="tbExpirationDate"
        Text="<%$Axlabel:@SYS88794%>">
    </asp:Label>
    <dynamics:AxValueFormatValidator ID="axVFVExpirationDate" runat="server"
        ControlToValidate="tbExpirationDate" DisplayErrorInInfolog="true"
        ErrorMessage="<%$Axlabel:@SYS100987%>" ExtendedDataType="TransDate" SetFocusOnError="True"
        ValidateEmptyText="True" ValidationGroup="Error" Display="Dynamic"> *
    </dynamics:AxValueFormatValidator>
```

For more information about validation, see Validating Unbound Data on MSDN.

Error handling

In Enterprise Portal, .NET Business Connector (including proxies), the metadata, and the data layer all throw exceptions when error conditions occur. The Enterprise Portal ASP.NET controls automatically handle these exceptions, taking appropriate actions and displaying the errors in the Infolog.

Exceptions in Enterprise Portal are divided into three categories. These exception categories are defined in the AxExceptionCategory enumeration.

NonFatal The exception handling code should respond appropriately and allow the request to continue normally.

AxFatal An unrecoverable error has occurred in Enterprise Portal. Enterprise Portal content will not be displayed. Content that is not related to Enterprise Portal should be displayed as expected.

SystemFatal A serious error, such as out of memory, has occurred, and the request must be aborted. Errors of this kind often cause an HTTP error code 500.

If your code does any of the following, it must handle any exceptions that may occur:

- Directly calls methods in data layers from Enterprise Portal
- Directly calls metadata methods
- Uses proxy classes to call X++ methods

The following code shows how to use AxControlExceptionHandler in the try-catch statement to handle exceptions.

```
try
{
    // Code that may encounter exceptions goes here.
}
catch (System.Exception ex)
{
    AxExceptionCategory exceptionCategory;

    // Determine whether the exception can be handled.
    if (AxControlExceptionHandler.TryHandleException(this, ex, out exceptionCategory) == false)
    {
        // The exception was fatal and cannot be handled. Rethrow it.
        throw;
    }
    if (exceptionCategory == AxExceptionCategory.NonFatal)
    {
        // Application code to properly respond to the exception goes here.
    }
}
```

AxControlExceptionHandler tries to handle Microsoft Dynamics AX exceptions based on the three exception categories described earlier in this section. It returns true if the exception is NonFatal.

Debugging

In this section, we list some tips to help you debug in Enterprise Portal.

- If you are using Remote Desktop Connection, make sure that you are using the /console switch.
- In Visual Studio, in the Options dialog box, under Debugging, make sure that you clear the Require source files to exactly match the original version check box.

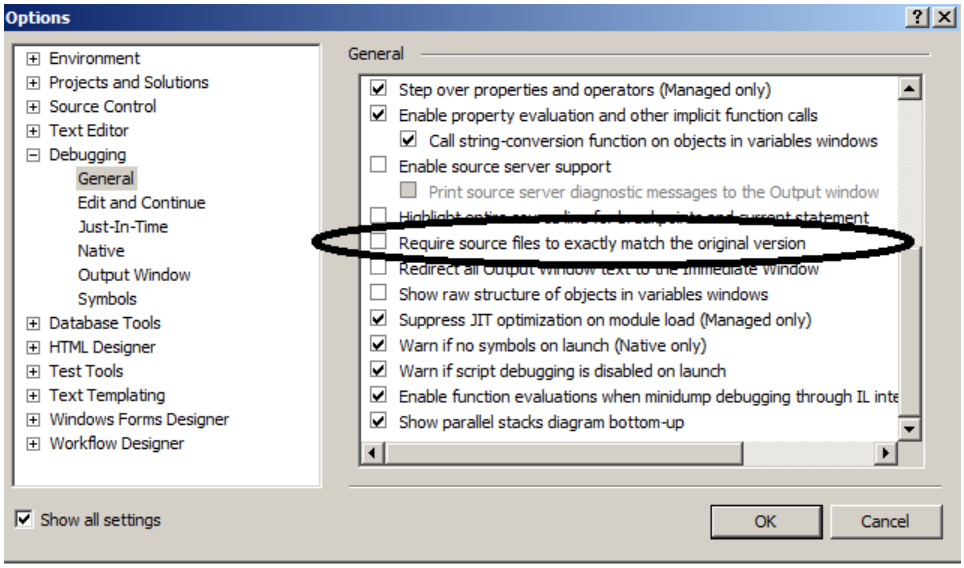


Figure 58 Debugging options in the Options dialog box

- In the SharePoint web.config file, set debug to true. Typically, the web.config file is located in C:\inetpub\wwroot\wss\VirtualDirectories\80.

```
<compilation batch="false" debug="true">
```

- When you attach the w3wp process, be sure to choose .NET 2.0 (not .NET 4.0).

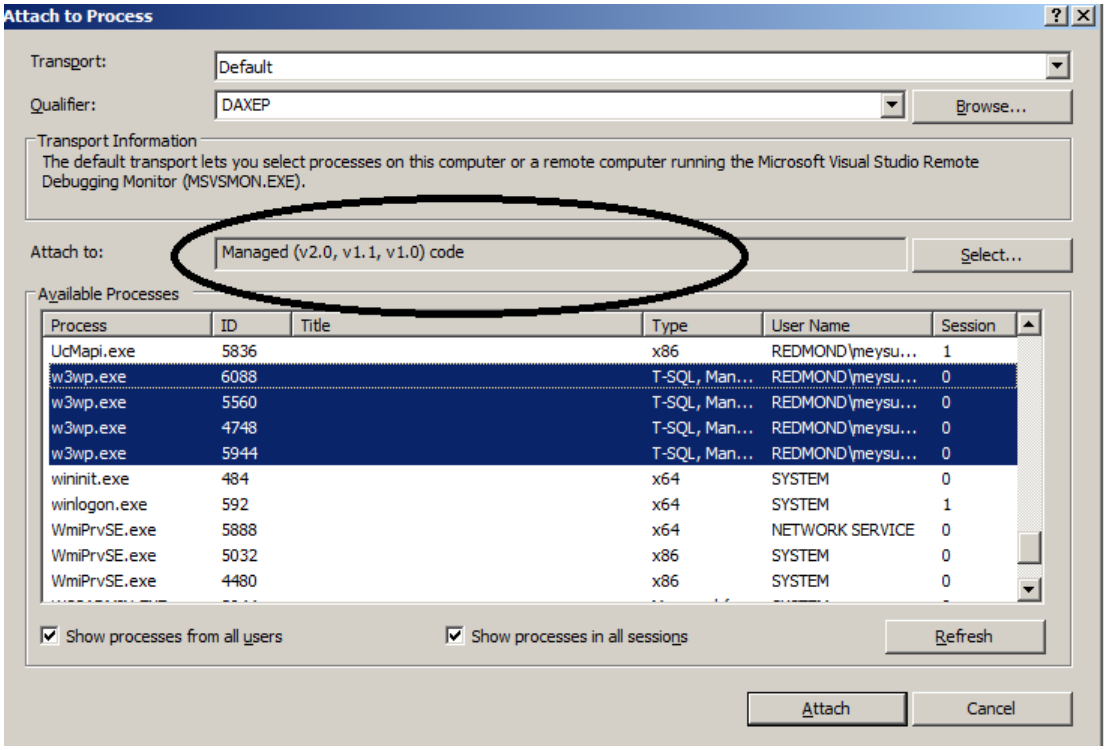


Figure 59 The Attach to Process dialog box

- In the Microsoft Dynamics AX 2012 configuration settings, create a debug profile for .NET Business Connector. On the Developer tab, select the settings to enable breakpoints.

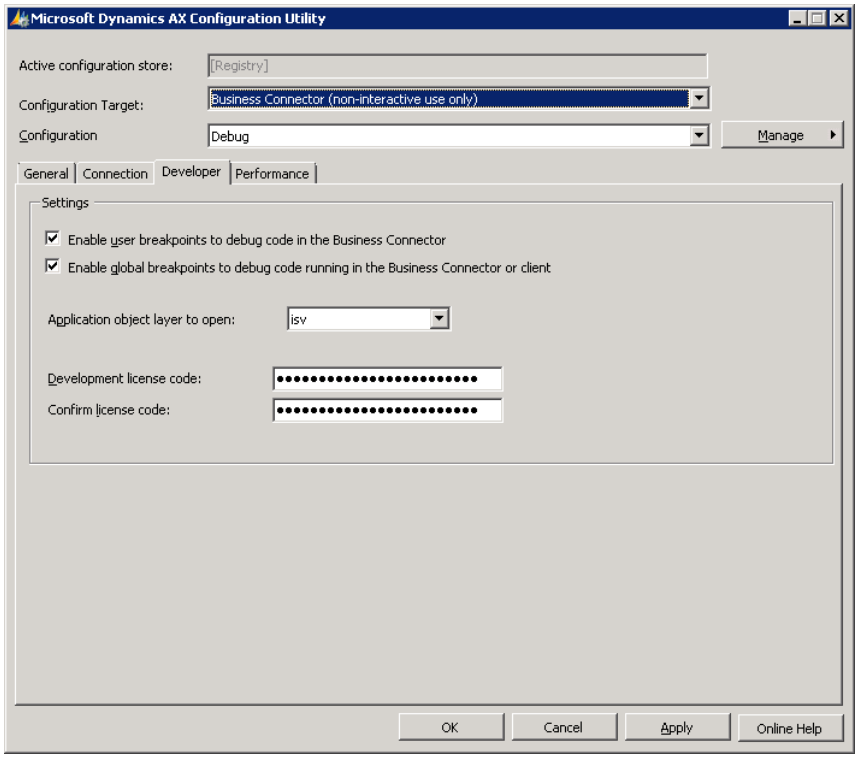


Figure 60 The Microsoft Dynamics AX Configuration Utility

- In MorphX (the Development Workspace), navigate to Tools > Options > Development > Debug. Make sure that the Debug mode is set to When Breakpoint.
- To debug X++ code, make sure that the AxDebugger (Tools >Debug) is open, because this cannot open automatically when you debug from Enterprise Portal.
- When you install Enterprise Portal along with the Microsoft Dynamics AX client, you can select a developer installation. This automatically enables many of the settings required for a debugging environment.
- Add a breakpoint in the X++ code. It's best to have the keyword breakpoint wherever you want it.
- For more details about debugging Enterprise Portal web user controls, see How to: Debug User Controls on MSDN.
- For more details about debugging X++ code in Enterprise Portal, see How to: Debug X++ Code on EP Pages on MSDN.

Infolog

To programmatically add messages to the Infolog, add the following namespace to your user control.

```
using Proxy = Microsoft.Dynamics.Framework.BusinessConnector.Proxy;
```

You can then add a message to the Infolog.

```
Proxy.Info objInfoLog = new Proxy.Info(this.AxSession.AxaptaAdapter);
objInfoLog.add(Proxy.Exception.Warning, "Hello World");
```

ViewState

The web is stateless, which means that each request for a page is treated as a new request, and no information is shared. When loaded, each ASP.NET page goes through a regular page life cycle, from initialization and page load onward. When a user interacts with the page in a way that requires the server to process control events, ASP.NET posts the values of the form to the same page, so that the events can be processed on the server. A new instance of the webpage class is created each time the page is requested from the server. When postback happens, ASP.NET uses the ViewState feature to preserve the state of the page and controls, so that any changes that are made to the page during the round trip are not lost. The Enterprise Portal framework leverages this feature, and Enterprise Portal ASP.NET controls automatically save their state to ViewState. The ASP.NET page reads the view state, and reinstates the page and control states in its regular page life cycle. Therefore, you don't need to write any code to manage the state if you're using Enterprise Portal controls. However, if you want to persist any in-memory variables, you can write code to add or remove items from StateBag in ASP.NET.

```
public int Counter
{
    get
    {
        Object counterObject = ViewState["Counter"];

        if (counterObject == null)
        {
            return 0;
        }

       return (int)counterObject;
    }

    set
    {
        ViewState["Counter"] = value;
    }
}
```

If you need to save the state of an X++ data set, you can use the pack and unpack design pattern to store the state.

The Enterprise Portal framework uses the EPStateStore table in Microsoft Dynamics AX to store the state of AxDataSourceControl. The states of all other controls are stored in the ASP.NET view state for added security. This storage is per user. Each user can read only his or her information. Writes and deletes are protected with AOS methods, and reads are protected through code access permission.

WebLink and AxUrlMenuItem

WebLink and AxUrlMenuItem can be used to generate links to content in Enterprise Portal.

Example 1: Generating a WebLink and returning the corresponding URL

```
ISession session = AxBaseWebPart.GetWebpart(this).Session;
WebLink webLink = new WebLink(session as ILinkContext);

URLWebMenuItemMetadata menuItemMetadata = MetadataCache.GetURLWebMenuItemMetadata(
    CommonConst.MenuItemName.CatCategoryRequest);

// Set menu item
webLink.SetWebMenuItemInfo(
    new WebUrlLinkMenuItemInfo(
        menuItemMetadata.Name,
        menuItemMetadata.GetLabel(session.SessionInfo),
        menuItemMetadata.GetHelpText(session.SessionInfo),
        menuItemMetadata.URL));

// Set data properties
webLink.RecordAsMILRecord = record;

// Return URL, encodeUrl = false and useExternalUrl = false
return webLink.Url(false, false);
```

Example 2: Creating an AxUrlMenuItem with context and parameters, and returning the corresponding URL

```
AxUrlMenuItem webMenuItem = new AxUrlMenuItem(DEFAULT_MENU_NAME);
webMenuItem.MenuItemContext = AxTableContext.Create(nextPK);
webMenuItem.ExtraParams.Add(SEARCH_TEXT, m_searchText);
return webMenuItem.Url.OriginalString;
```

Example 3: Creating the complete URL in code, based on a web menu item, and adding context

```
protected override void OnPreRender(EventArgs e)
{
    // Gets the current view to the record selected in the dropdown.
    DataSetView view = dsSales.GetDataSourceView(ddSales.DataMember).DataSetView;
    DataSetViewRow row = view.GetCurrent();

    // Get the URL for the webpage that referenced the current record
    // In the sample an extra method was made for this
    AxUrlMenuItem url = new AxUrlMenuItem("EPSalesTableInfo");

    // Set the record context on the URL so we can open the page for a specific record
    DataSourceMetadata metaData = this.dsSales.Metadata.DataSources[ddSales.DataMember];
    AxTableContext context = AxTableContext.Create(row.GetDefaultTableDataKey(metaData));
    url.MenuItemContext = context;

    // update teh hyperlink to point to the URL
    hpRecord.NavigateUrl = url.Url.ToString();

    base.OnPreRender(e);
}
```

HTMLUtilities

For user controls that construct HTML through code behind, the HTMLUtilities class provides methods for encoding and decoding HTML and attributes, and for cleaning up unsafe HTML.

Example 1: Encoding HTML text

```
listItem.Text = HtmlUtilities.EncodeHtml(text);
```

Example 2: Encoding HTML attributes

```
string s = string.Format("<a href={0}{1}{0}>{2}</a>", '"',
    HtmlUtilities.EncodeAttribute(GetProductDetailNavigateUrl(inventTableRecId)),
    Labels.GetLabel("@SYS190991"));
```

Example 3: Cleaning up unsafe HTML in the code behind

```
// If we have to show raw HTML, use CleanupUnsafeHtml as anti-XSS measure to remove really dangerous
// HTML tags
this._groupHeaderText = HtmlUtilities.CleanupUnsafeHtml(value);
```

Example 4: Cleaning up unsafe HTML in markup

```
<asp:Literal ID="ltlECPPresentationProductSummary" runat="server"
    Text='<%#HtmlUtilities.CleanupUnsafeHtml((string)Eval("ProductSummary"))%>' />
```

Adding JavaScript

To add a script block, you can use the ScriptManager.RegisterClientScriptBlock function provided by the .NET Framework.

```
void promptDiv_PromptDismissed(object sender, PromptDismissedEventArgs<PromptResult> e)
{
    if (e.ClickedButton == PromptClickedButton.No)
    {
        // Script to uncheck the checkbox
        StringBuilder st = new StringBuilder();
        st.Append("var checkBox = document.getElementById('" + this.IsPrimaryCheck.ClientID + "');");
        st.Append("if (checkBox) {checkBox.checked = false;}");

        // Register the script to uncheck the check box
        ScriptManager.RegisterClientScriptBlock(
            this.promptDiv, this.promptDiv.GetType(), "changeValue", st.ToString(), true);
    }

    // Mark the check
    this.IsPrimaryCheck.Checked = (e.ClickedButton == PromptClickedButton.Yes);
}
```

Images

Images can be stored in the AOT > Resources node or in Microsoft Dynamics AX 2012 tables. If the images are stored in tables, you need to add those tables to the SysEPDeployment::deployCompanyImages method. For example, you should add deployImages(tableStr(<TableName>)), where <TableName> is the name of the table that contains the images.

The images that are stored in the AOT Resources node can be referenced in the Action Pane, menu item, or web menu item by using the normal image property.

The images that are stored in tables can be referenced in the user control. The following code sample shows how you can do this.

```
string imageName = ApplicationProxy.EPWebSiteParameters.imageName(
    currentRecord, TableDataFieldMetadata.FieldNum(AxSession, "TableName", "FieldNameStoringImage"));

string imageType = ApplicationProxy.EPWebSiteParameters.imageType(currentRecord);

sbHTML.Append("<p align='center'><img src='"/_layouts/ep/images/" + imageName + "." + imageType +
    "' border='0'");
```

When Enterprise Portal is deployed or updated by using either the Manage Deployment dialog box on the Microsoft Dynamics AX client or the AxUpdatePortal utility, the images from the Resources node in the AOT, and from the tables that are specified in the SysEPDeployment.deployCompanyImages method, are published to the \Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ep\layouts\ep\images folder.

ASP.NET chart controls

This section lists the steps required to start using ASP.NET chart controls in ASP.NET. Many of the steps here are required for using the ASP.NET chart controls in any ASP.NET web application and are not specific to Enterprise Portal.

1. Download and install the ASP.NET chart controls.
2. Open the SharePoint web.config file for editing. Typically, the web.config file is located in C:\inetpub\wwroot\wss\VirtualDirectories\80.
    a. Add the following to the <httpHandlers> section.

```
<add path="ChartImg.axd" verb="GET,HEAD"
    type="System.Web.UI.DataVisualization.Charting.ChartHttpHandler, System.Web.DataVisualization,
    Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" validate="false" />
```

    b. Just after </system.web>, add the following AppSetting.

```
<appSettings>
    <add key="ChartImageHandler" value="storage=file;timeout=20;dir=c:\TempImageFiles\;" />
</appSettings>
```

    c. Just before </pages>, add the following.

```
<controls>
    <add tagPrefix="asp" namespace="System.Web.UI.DataVisualization.Charting"
        assembly="System.Web.DataVisualization, Version=3.5.0.0, Culture=neutral,
        PublicKeyToken=31bf3856ad364e35" />
</controls>
```

3. Create a folder named TempImageFiles in C:\, and give read access to the account that is being used for the Enterprise Portal application pool in IIS. Typically, this is the BC proxy account.

4. You can now add ASP.NET charts to a web control, just as for any other ASP.NET web application. You can use an AxDataSource control to provide the data to the chart control. See the following example.

```
<dynamics:AxDataSource ID="AxDataSource1" runat="server" DataSetName="FMTripMileage"
    ProviderView="FMTrip" Role="Consumer">
</dynamics:AxDataSource>
<asp:Chart ID="Chart1" runat="server" DataSourceID="AxDataSource1" DataMember="FMTrip"
    Palette="Pastel">
    <Legends>
        <asp:Legend Docking="Bottom" LegendStyle="Row" Name="Legend1">
        </asp:Legend>
    </Legends>
    <Series>
        <asp:Series Name="Mileage" XValueMember="TripID" YValueMembers="TotalMileage**"
            Legend="Legend1" ChartArea="ChartArea1">
        </asp:Series>
        <asp:Series Name="Duration" XValueMember="TripID" YValueMembers="TotalDuration**"
            Legend="Legend1" ChartArea="ChartArea1" YAxisType="Secondary">
        </asp:Series>
    </Series>
    <ChartAreas>
        <asp:ChartArea Name="ChartArea1">
            <AxisX Title="Trip ID"></AxisX>
            <AxisY Title="Miles">
                <MajorGrid LineColor="Gray" />
            </AxisY>
            <AxisY2 Title="Days">
                <MajorGrid Enabled="False" />
            </AxisY2>
            <Area3DStyle PointDepth="50" Rotation="10" />
        </asp:ChartArea>
    </ChartAreas>
</asp:Chart>
```

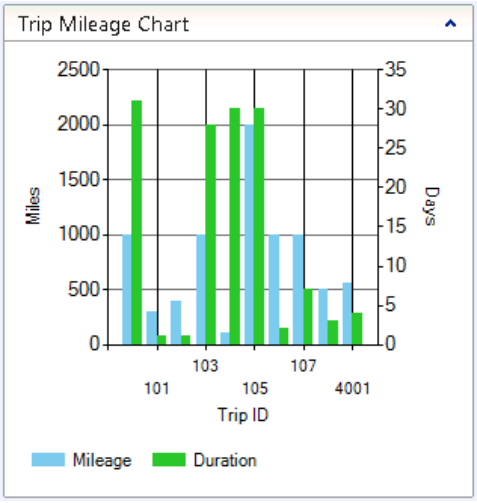


Figure 61 An ASP.NET chart control

Web modules

Web modules define the SharePoint sites and subsites in Enterprise Portal (for example, Sales, Financial, or Employee Services). These also provide properties that define Quick Launch and other aspects of each web module. You can use the following steps to create and deploy your own custom modules:

1. Create your web module as a submodule (not a sibling) of the Home module.
2. Set the QuickLaunch property to the web menu that you want to display in the Quick Launch area for the module.
3. Set the MenuItemName property to the default page for the module.
4. Set the InheritNavigation property as required. By default, this is set to true, and a module's submodules will inherit their parent's top navigation bar. For example, if the top navigation bar for the root Enterprise Portal site displays links to the Home, Sales, and Project modules, a user who navigates to the Sales module will continue to see the same top navigation bar. However, if the InheritNavigation property is set to false, the top navigation bar will change and display links to the submodules within Sales (if there are any). See the Employee Services module for an example of this.
5. Set the ShowLink property as required. By default, this is set to yes. If the property is set to no, the module name will not appear in the top navigation bar. Even if you set ShowLink to no, a user can still browse to the page by typing the URL in the browser's address bar.
6. Right-click the module, and then click Deploy Element.
7. After deployment is completed, reset IIS and AOS to make the new module appear in the top navigation bar.

Note On the Microsoft Dynamics AX client, go to System Administration > Setup > Enterprise Portal > Web sites. Make sure that the site that you want to deploy to is set as the default Enterprise Portal site.

ShowParentModule

ShowParentModule is a new property that is now available on web modules, web menus, and menu items. For pages that are shared between modules, you should set the ShowParentModule property to No on the menu items on web menus. For pages that are not shared, keep the default value (Yes) for the property.

The use of ShowParentModule is best explained through an example. Let's say that you have a page within the Sales module, and that Quick Launch of the Procurement module contains a link to this page (a Menu Item). When the user clicks the link, Sales becomes the active module, because the page belongs to the Sales module. This also causes the top navigation bar and Quick Launch to change, based on the settings of the Sales module.

If the ShowParentModule property of the menu item that links to the page in the Sales module is set to No, clicking the link will open the page from the Sales module, but it will not change the active module. Procurement will remain the active module, and the top navigation bar and Quick Launch will also remain the same. This might help make the user navigation experience better.

For more information about web modules, see Web Modules Overview on MSDN.

Cues

In Microsoft Dynamics AX 2012, cues are modeled in the AOT. Cues should point to the MenuItem of the secondary list page, which points to a query, as shown in the following logical model.

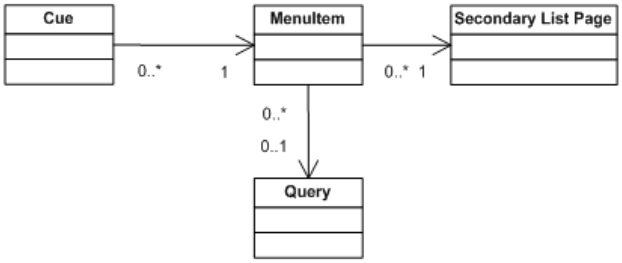


Figure 62 A logical model of Cues

You can create cues in the AOT by using the following steps. A cue uses a modeled query to retrieve the records. You can also have the cue link to a form, typically a list page.

1. Create or identify a list page or form that you want to use.
2. Identify the query that is being used by the list page.
3. Create a menu item that references the form and query:
    a. In the AOT, go to Menu Items > Display. Right-click the Display node, click New Menu Item, and then create a menu item that points to your form.
    b. Set the ObjectType property to Form.
    c. Set the Object property to the name of your form.
    d. Set the Query property to the name of your query.
    e. Save the menu item.
4. Create the cue:
    a. In the AOT\Parts\Cues node, right-click, and then click New Cue to create a new cue.
    b. Set the MenuItemName property to the menu item that you created in the preceding step.
    c. Set other cue properties, as needed, (ShowAlert, ShowSum, and so on) to change the values from the default values.
    d. Save the cue.
5. Create a cue group, and add a reference to the cue:
    a. In the AOT, go to Parts > Cue Groups. Right-click the Cue Groups node, click New Cue Group, and then create a new cue group. Cue groups do not contain cues directly; instead, they contain references that point to cues.
    b. Right-click the new Cue Group, and then click New Cue Reference to create a new cue reference.
    c. Set the Cue property to the cue that you created in the preceding step.

d. Optional: Add Cue references for each cue to display in the cue group.

e. Save the cue group.

6. Refer to the cue group from a Cues Web Part in a Role Center:

    a. On a Web Part page (typically, a Role Center page), add a Cues Web Part.

    b. Edit the Cues Web Part.

    c. Set the CueGroup Name property to the cue group that you created in the preceding step.

    d. Save the changes to the Cues Web Part, and exit edit mode.

7. Verify that the cues in the referenced cue group show up with correct counts and links.

For more information about cues, see Cues and Cue Groups on MSDN.

SharePoint integration

Enterprise Search

Enterprise Search in Microsoft Dynamics AX 2012 enables users to search for data, metadata, and the contents of documents that are attached to records. This search capability is available both in Enterprise Portal and on the Microsoft Dynamics AX client. Enterprise Search uses the Metadata service and the Query service in Microsoft Dynamics AX to gather the data and metadata from Microsoft Dynamics AX. To index and execute search queries, it uses the SharePoint Business Connectivity Services (BCS).

To enable this rich search functionality, you have to install the Enterprise Search component from Microsoft Dynamics AX Setup. If you are using SharePoint Server, you do not need to install any other prerequisites for Enterprise Search, because SharePoint Server has built-in search capabilities. However, if you are using SharePoint Foundation, you will need to install Microsoft Search Server Express as a prerequisite for Enterprise Search.

Enterprise Search uses Queries to make data searchable in Microsoft Dynamics AX. When Enterprise Search is installed, it indexes the default queries, and runs a full crawl of the data and metadata. If you want to make more data searchable, use the following steps:

1. In the AOT, find the Query that fetches the data, or create a new one.
2. Set the Searchable property on the query to Yes.
3. Compile the query, and ensure that there are no best practice errors.
4. On the Microsoft Dynamics AX client, start the Enterprise search configuration wizard (System Administration > Setup > Search > Search configuration). This will show you a list of all the queries in the AOT that have the Searchable property set to Yes.
5. By default, all queries with the Searchable property set to Yes are selected for publication to SharePoint Business Connectivity Services. You can cancel the selection of any queries that you do not want to publish.
6. You can also prevent specific fields from being indexed by using the Select fields option.
7. Select the check box to start a full crawl of the data source. Alternatively, you can use SharePoint Central Administration to manually start a full or incremental crawl.

8. Click Next and then Finish.

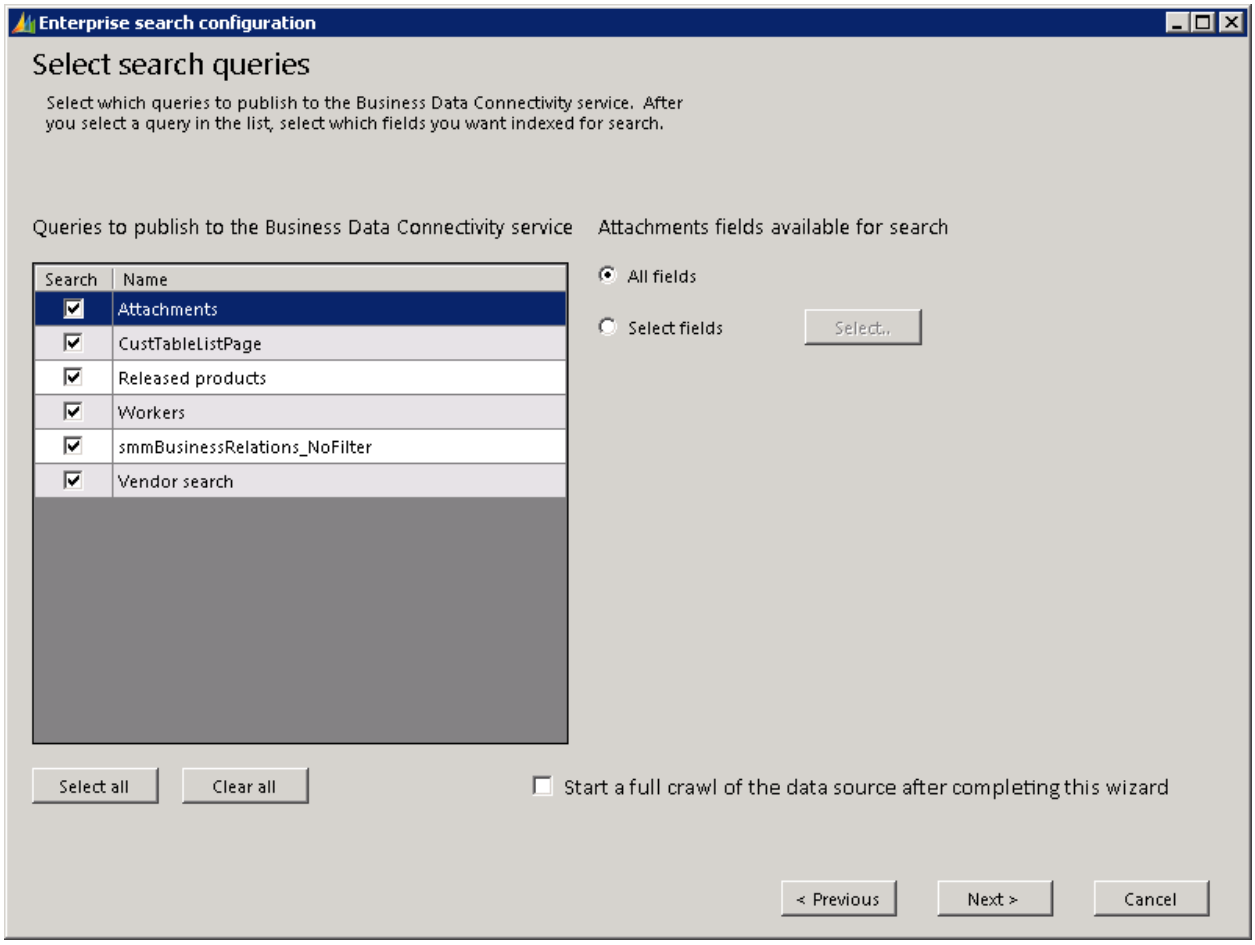


Figure 63 The Enterprise search configuration wizard

The Enterprise search configuration wizard uses the credentials of a search crawler account to index the data and publishes the queries to Business Data Connectivity Service. You can also see your published queries in SharePoint Central Administration, under Application Management > Manage service applications > Business Data Connectivity Service.

A change in metadata information, such as the web menus, is rare; therefore, by default, Enterprise Search executes a full crawl of the metadata only once, during installation. Data, on the other hand, such as sales orders, changes frequently; therefore, by default, Enterprise Search performs a full crawl of the data once during installation, and then performs an incremental crawl every day at midnight.

If you publish a new query, you can start a full crawl directly from the Enterprise search configuration wizard, as mentioned earlier. You can also start a full or incremental crawl manually by using SharePoint Central Administration. To do this, use the following steps:

1. Start SharePoint Central Administration.
2. Navigate to Application Management > Manage service applications > Search service application.

3. In the left navigation, under Crawling, click Content Sources. You will see two content sources: one for data and one for metadata.

4. Click one of these content sources, and then select the option to start a full or incremental crawl, as shown in Figure 62. Keep in mind that crawling can take a long time, depending on how much data or metadata there is to index.

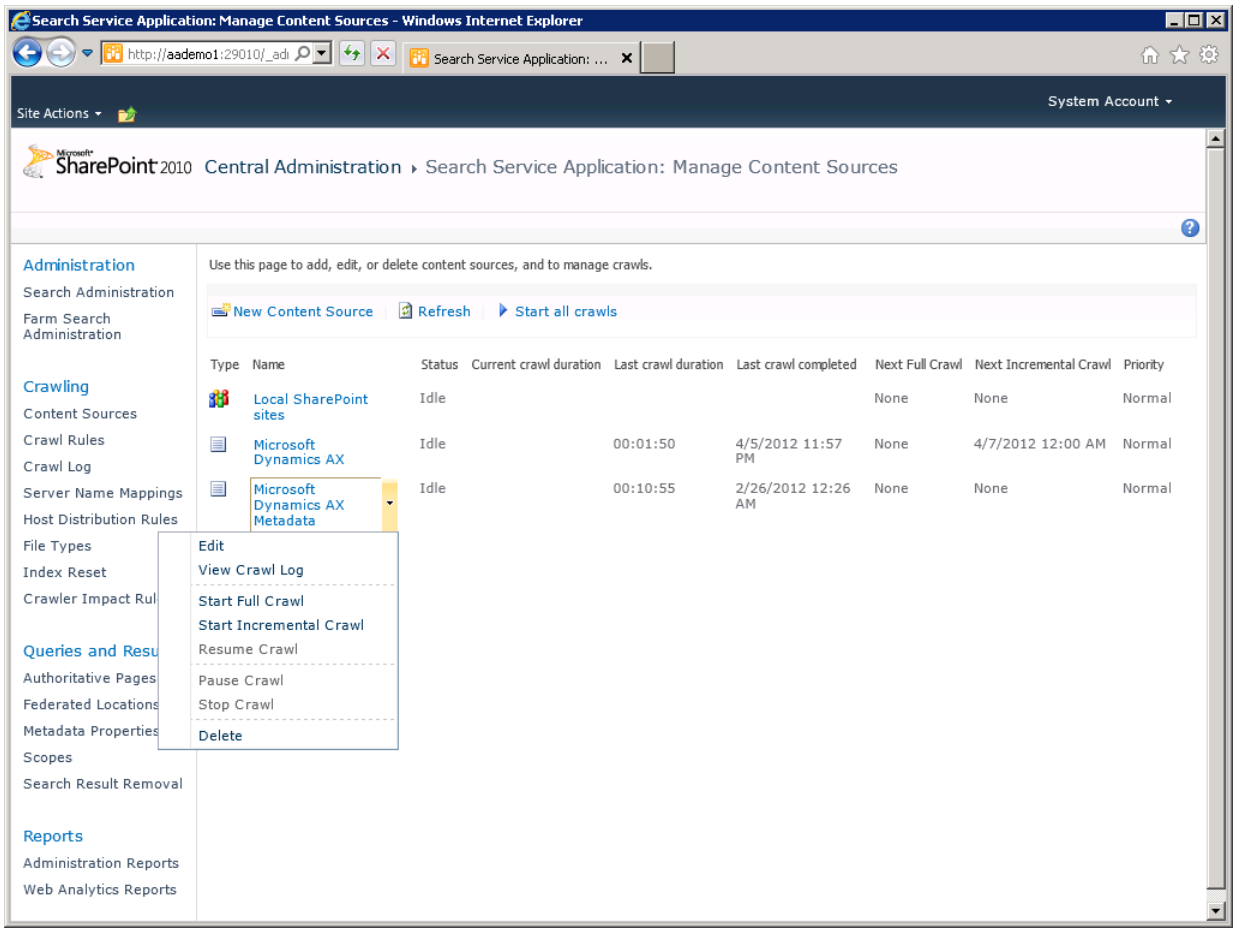


Figure 64 Starting a crawl by using SharePoint Central Administration

After the data and metadata have been crawled and indexed, they are published to SharePoint Business Connectivity Services. Users can execute searches by using the search box located in the upper-right corner of Enterprise Portal. The results are trimmed at search time, based on the user's role, language, and other settings, so that the users see only data that is available and applicable to them.

Themes

Enterprise Portal integrates with SharePoint themes. You can apply an existing SharePoint theme to the Enterprise Portal site to change its appearance, just like any other SharePoint site. Partners and customers can also create new SharePoint themes, and customize or extend the Enterprise Portal style sheets to map to the new theme.

Enterprise Portal uses five style sheets. AXEP.css is the base style sheet. AXEP_RTL.css is used for right-to-left languages and cascades on top of AXEP.css. These two files are located on the web server under <drive>:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\<lcid>\STYLES\Themable. The AXEP_CRC.css, AXEP_CRC_RTL.css, and AXEP_WebPart_Padding.css style sheets are used for Role Centers when they are rendered on the Microsoft Dynamics AX client. These files are located on the web server under <drive>:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ep\Stylesheets.

The Enterprise Portal master page references these style sheets. The AXEP.css and AXEP_RTL.css style sheets contain SharePoint theme directives, and are therefore placed in the special directory where SharePoint can locate them.

When a SharePoint theme is applied to the Enterprise Portal site, SharePoint parses the directives and makes modifications to reflect the new theme. These modifications include color and font replacements, and even recoloring of some images. It then stores the modified style sheet and images in the SharePoint content database. The master page then references this new style sheet, so that the Enterprise Portal appearance reflects the applied theme.

Tips

Here are some SharePoint tips related to Enterprise Portal:

- If a web control on a Web Part page has an error, the page most likely will not be able to be displayed. If you want to remove the corrupt Web Part from the page, you can append &contents=1 to the query string. This will display a list of all the Web Parts on the page. You can then delete the corrupt Web Part and reload the page after removing &contents=1 from the query string.
- If you navigate to http://<ServerName>/sites/DynamicsAx/Enterprise%20Portal/forms/allitems.aspx, you can see a list of all the pages in Enterprise Portal. This can come in handy when your default page is throwing an error, and you want to open a different page.
- You can programmatically set the SharePoint zone width, as shown in the following code sample.

```
protected override void OnPreRender(EventArgs e)
{
    base.OnPreRender(e);
    //Add style to get 80/20 column widths for the left/right Web part zones
    Microsoft.Dynamics.Framework.Portal.UI.StyleRegisterHelper.SetWebPartZoneWidths(
        this.Page, Unit.Percentage(80), Unit.Percentage(20));
}
```

Workflow

For more information about Workflow, see http://msdn.microsoft.com/en-us/library/ee677494.aspx.

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics