NetQuarry, Inc.

Training

500 - Metadata Examples

Metadata Cookbook

This document will detail how many common user interface tasks can be achieved by just configuring the appropriate meta data (that is the properties and attributes of a objects) without writing any code at all.

There are many ways in which configuration of object properties can achieve the same end result. The choice of which is the best way to achieve your goal is dependent on the requirement of the feature you are trying to implement. In a way, the requirement to hide a field is a perfect example of having many ways to do one thing, but each way is not necessarily best for a particular case.

Hiding Fields

Very common requirement

Method	When would you use such a method?	Pros	Cons
Set column width to 0	When the field is always likely to be hidden and never visible. For example primary key fields are always set to 0 column width by the slurper, because we expect this to contain a guid for example that a user probably shouldn't ever see.	Easiest way to hide a field. The field will never appear to the user The field is always available programmatically to the developer	The field will never appear to the user. The field is always selected in a mapper requery, even if it's never going to be visible.
Set an exclude flavor	When you want a field to be visible in the majority of circumstances and hidden in the minority of cases. Typical scenarios are that you want a field to be visible on a detail screen, but not visible on the list view.	The field is excluded from the mapper's field collection so it isn't included in the Select during a requery.	The field is actually excluded from the mapper's field collection so is not available programmatically to the developer when writing code for list views
Set an include flavor	When you want a field to be visible in the minority of circumstances and hidden in the majority of cases. Typical scenarios are that you want a field to be visible on a wizard screen, but not visible anywhere else.	The field is excluded from the mapper's field collection so it isn't included in the Select during a requery.	The field is actually excluded from the mapper's field collection so is not available programmatically to the developer when writing code for list views. The fields with include flavors specified are not included in the Typed Mapper code generation.

Hiding Fields

Very common requirement

Method	When would you use such a method?	Pros	Cons
Set a hide flavor	When you want a field to hidden from a user based on the flavor, but you also still need to be able to refer to the field programmatically and to be selected from the database.	The field is hidden for specific flavors as though it was excluded. The field is always available programmatically to the developer. The field is always selected in a mapper requery, even if it's not visible.	
Set a permission	When you want to make a field visible, or not depending on the role of the user logged in. To hide a field for a role, you should uncheck the Read and Write permission for the field. The field is hidden but still available programmatically. If you want the field to be excluded from the mapper fields collection, you uncheck all the permission checkboxes for the field.		
Use version control	When you want to hide a field from view that is related to a new feature. Until the feature is ready you don't want the field to be visible (or when a new feature is added, you want a field to be made invisible).		Need to manage the version in the versions table
Set the FilterOptions poperty attribute DefaultHideInList	You want to hide a field from the list view but you also want it to be available to a user to add back into the list view.	Reduces the size of the list view page when the fields are hidden.	

Locking Fields

Very common requirement

Method	When would you use such a method?	Pros	Cons
Set Locked attribute	When the field is always likely to be locked. For example, you have an identity field that you want let the user see, but obviously they can't edit it.	Easiest way to lock a field. The field is always locked. The field is always available programmatically to the developer	The field is always locked.
Set a lock flavor	When you want a field to be locked in certain circumstances. Typical scenarios are that you want a field to be locked on a detail screen but editable on a wizard screen.		
Set a permission	When you want to make a field locked that isn't a button, depending on the role of the user logged in. To lock a field for a role, you should uncheck the Write permission for the field.		
Set a permission	When you want to make a BUTTON locked, depending on the role of the user logged in. To lock a Button for a role, you should uncheck the Navigate permission for the button.		
Set the InitOnly field attribute.	When you want a field to be editable on creation of a new record but after the save, the same field is locked.	Manages logic of lock/unlock without code.	Always locked on an existing record as though the locked attribute was set.
Set the table property to "-", or blank it out, or set the value to the name of the mapper view	This works on the principle that the mapper cannot find a primary key field in the mapper for the table referenced in the property. The field is always locked as though the fields Locked attribute has been set.		It's better to use another method to explicitly lock a field. You can never have the value of the field saved to the database.
Uncheck the PrimaryKey field attribute(s) on the mapper.	You didn't want to make changes to a particular table.		It's better to use another method to explicitly lock a field. None of the fields related to the primary key field (by TableName) will be saved to the database.
Set the DenyUpdate and DenyInsert mapper attributes	When you want to lock all the fields on the mapper.	A quick way of locking all the fields of a mapper at once. The mapper still supports saving to the database.	

Use the same mapper to display one field as two different controls for different user roles

An Administrator logs in and navigates to a page, they want to see a field as a combo box with a list of clients. When a client logs in and navigates to the same page, the same field should be visible, but there may be a different picklist, or different control, or the control is locked, or any combination of these scenarios.

Method	When would you use such a method?
Use permissions	When it doesn't matter whether the field for a client user is a text box, or combo box, the field just has to be locked to the Client user. **How?** Uncheck the Write permission for client roles.
Use two fields that are mutually exclusive by permissions	When the fields are comboxes, but the picklists might be different depending on whether the user is an Administrator or a Client. Or, when the field for the Administrator is a different type to the field for the Client **How?** Set the permissions on the field for the administrator. For all roles, except administrator roles, uncheck all the permissions check boxes. (Highlight cells and hit space bar to toggle check/uncheck). Then right click on the field and duplicate. The duplicate field will have all permissions set to allow. For the duplicate field's permissions, for the administrator roles, uncheck the permissions check boxes. There are now two fields in the mapper with different permissions. For the field associated with the client roles, change the picklist to be the picklist appropriate for the client roles.
Use two fields that are mutually exclusive by flavor	When the fields are comboxes, but the picklists might be different depending on the flavor of the mapper in use. Flavors can be used indirectly to modify behavior based on permissions. The Permissions are converted to flavors via configuration of pages. That will be discussed next. **How?** Choose your flavor(s) on which to base your logic. Let's say the flavors are Custom1 and Custom2. Custom1 is for Administrators and Custom2 Is for Clients. Right click on the field and duplicate. On the current row, popup the include flavor widget and choose Custom1, then popup the exclude flavor widget and choose Custom2. Move to the duplicate field and popup the include flavor widget and choose Custom2, then popup the exclude flavor widget and choose Custom1. Change the picklist on this field to be the Client specific picklist. You have to make the flavor choices in this way because if there are two fields in a field collection with the same key name, the logic is to exclude both fields from the mapper and the field is not visible. Because you've set both fields with include flavors, it means that field is now excluded during the typed mapper generation. If you NEED that field in the typed mapper, because you've got a lot of code written with it, then you can duplicate the field again and for the second duplicate, set the Include flavor to 0 and the exclude flavor to Custom1 and Custom2.

Use flavors to represent different user roles

In order to convert permissions to flavors, you apply the permissions settings at a higher level than the mapper where you can set the flavor on the higher permission specific object. The next level up from a mapper is a page. The highest level is the navigator because that is how you navigate to a page. We'll describe how to perform the permission to flavor conversion at each level, but the best approach is to convert permissions to flavors at the navigator level.

Method	**When would you use such a method?**
Page level conversion	Well you really wouldn't want to do it this way unless you really had to. When you create duplicate pages, it also means you have to manage duplicate sets of layout preferences, although that can alleviated by pages being able to re-use the layout of other pages via the page's DetailLayout property **How?** Go to the Pages list and select the page you want to create for multiple roles. Remove the Read permissions on all but Administrator roles. Set the Flavor property for this page to be Custom1 Right click on the page in the list and duplicate. Change the moniker and add something that indicates the new page is for client roles, but don't change the name of the page. Unset the read permissions on the duplicate page (the client specific page) for the Administrator roles. Set the flavor property for this page to be Custom2
Navigator level conversion	This would be the preferred way to convert permissions to flavors. Now when we talk about navigators, we mean any one of the three types, Navbars, Subforms, Commands. All three types have a NavTarget that navigates to a page and it is these Page NavTargets we want to manipulate. This example we'll consider Navbar NavTargets. **How?** On the NavTarget that navigates to the page, remove the Read permission on all but Administrator roles. Set the Flavor property for this NavTarget to be Custom1 Right click on the NavTarget in the list and duplicate. Change the name of the duplicated nav target to make the purpose clear. Unset the read permissions on the duplicate NavTarget (the client specific NavTarget) for the Administrator roles. Set the flavor property for this NavTarget to be Custom2

Filter list views based on roles

It's frequently required to have a list view of the same data, but filtered differently depending on which user role is logged in. Typically Administrators have an unrestricted view of data. Clients are only allowed to see records related to their company.

Method	When would you use such a method?
Role specific pages.	Well you really wouldn't want to do it this way unless you really had to. When you create duplicate pages, it also means you have to manage duplicate sets of layout preferences, although that can alleviated by pages being able to re-use the layout of other pages via the page's DetailLayout property **How?** On each page you specify for each set of user roles, set the appropriate Read permissions On each page, set the appropriate filter clause in the Filter property.
Navigator specific pages	This would be the preferred way to convert permissions to flavors. Now when we talk about navigators, we mean any one of the three types, Navbars, Subforms, Commands. All three types have a NavTarget that navigates to a page and it is these Page NavTargets we want to manipulate. For the majority of cases, you will be making these changes on NavTargets of NavBars. **How?** On each nav target you specify for each set of users roles, set the appropriate Read permissions. On each NavTarget, set the appropriate filter clause in the Filter property. Make sure each role specific target is named appropriately to assist with discovery.

Show or Hide action buttons in list view based on the status of the record

On a timesheet list, for example, you might want to see action buttons that allow a user to Approve, or Unapprove the timesheet depending on whether the timesheet is already approved or not.

Method	When would you use such a method?
Button fields in a group. Button visibility based on values of a field on the mapper	On a timesheet list, for example, you might want to see action buttons that allow a user to Approve, or Unapprove the timesheet depending on whether the timesheet is already approved or not. **How?** In a mapper, add a field of type GroupBox, called grp_actions. Since the field is not in the underlying view, you make set the Field attribute ExcludeFromSelect. We've said we want the action buttons to appear in the list view, so we have to tell the GroupBox field to appear in the list view. It normally doesn't. Set the CellTypeAttribute ShowInList. Now add two fields to the mapper of type Button called btn_approve, btn_unapprove Set the captions to Approve and Unapprove. For these buttons we want the Approve button to only appear when the timesheet is unapproved (status_id = 0) and we want the Unapprove button to only appear when the timesheet is approved (status_id = 1). We have to connect the visibility of the buttons with the value of status_id field. On each button field, set the AliasName property to status_id to tell the mapper to select the value from this field (rather than btn_approve which is not in the view). On each button field, set the CellTypeAttribute ValueHidden to true. This tells the button to show or hide if the field values are true or false. Then on each button, you have to tell it which value of status_id represents a true value (that is, visible) and a false value (that is, hidden). btn_approve is visible when the status_id is 0. So set the TrueValues property to 0 and the FalseValues property to 1. on btn_unapprove field, set the TrueValues property to 1 and the FalseValues property to 0. Finally, associate the button fields to the group box by setting each field's Group property to grp_actions (the key name of the GroupBox).

Display a rich popup screen when a user hovers over a control

Method	
The field with the hover is directed to a page implementing a Summary page element referring to a nicely formatted template.	**How?** First create a little piece of html that acts as the template for the hover detail. The template will have fields references to a mapper that provides the data to summarize. The template is saved into the Template folder. Add a record to the Templates list for the template you've just created. Make sure you set the FileName property of the Template to the name of your template fragment. Now, create a new page based on the TabbedSubformTemplate. There's no need to specify a mapper, nor a base page. In the Page Elements subform, add a new Summary slot, using the MapperSummary component. In the properties for the Summary page element, set the Template property to the template you've just created. Set the Mapper property to the mapper containing the appropriate summary information. On the mapper of the page where you want the summary to appear, go to the field where you want the hover. On that field, set the SummaryKey property. This property will be set to a key name of a field on the mapper that provides the filter key to the summary mapper. On the same field, set the SummaryMOP property to the page you just created. On the same field, set the ShowHoverSummary attribute on the FieldBehaviorAttributes property.
The field with the hover is directed to a page implementing a Summary page element with no template	**How?** Create a new page based on the TabbedSubformTemplate. There's no need to specify a mapper, nor a base page. In the Page Elements subform, add a new Summary slot, using the MapperSummary component. In the properties for the Summary page element, set the Template property to the template you've just created. Set the Mapper property to the mapper containing the appropriate summary information. If you want to restrict the set of fields that are displayed in the summary, then set the FieldList property to a semi-colon delimited list of fields in the mapper that you want to summarize. On the mapper of the page where you want the summary to appear, go to the field where you want the hover. On that field, set the SummaryKey property. This property will be set to a key name of a field on the mapper that provides the filter key to the summary mapper. On the same field, set the SummaryMOP property to the page you just created. On the same field, set the ShowHoverSummary attribute on the FieldBehaviorAttributes property.

Make a subform show and hide different tabs depending on the parent record

Method	
Use the EnableRule property of the Subform target.	For a given page and set of subforms, you may not want all subforms to be displayed if the parent has a certain status. With a list view as the parent, and as you move from row to row, you want to show and hide different subform tabs because the status of each record in the parent list view is different.

How?
First, you have to make sure that the field(s) that determine whether a subform tab is visible, or not is/are not excluded and is/are visible on the parent detail, or list. Go to the subform targets for the page, and for each of the targets you want to hide or show, based on the parent status, set the EnableRule property. The EnableRule property is a piece of JavaScript script that contains field references to the fields in the parent detail or list which resolves to either a true, or false value.

Set up a console style page

Create a page that provides multiple views into different areas of an application. Summaries of record counts with links to filtered lists and/or graphs, etc.

Method	
Create a page that uses the console template and add page elements for information you want to display.	**How?** Create a new page and use the ConsoleTemplate as the template. Add Page elements to the Page. For page elements, there always has to be a Main slot defined and this "slot" will always be located in the top left hand corner of the console page. The Main slot can refer to any of the supported console components, like MiniDetail, MiniList, MiniNav. You can add a Side slot that refers to a NavBar so you get the tree view on the left. Further page elements can be added using the Slot Name of ConsolePage. In the properties of the Page itself, you set the Columns property to specify the number of columns to use. The ColumnWidths property lets you define the absolute widths of your columns in pixels (semi colon delimited). If the entries in the Console provide a count of rows, you can specify a maximum row count to display to improve the performance of the rendering of those pages. You limit be the row count by setting the RecordCountLimit property. On each page element, you can define which column the element will appear by setting the Column property. The column indexes are zero based.
Use a Navigator	To add a navigator style control to the console. **How?** Add a new slot to the page elements using the ConsolPage slot name and the MiniNav component. Set the Navigator property to indicate what navigator is going to be used to populate the MiniNav slot.
Use a List View	To add a list view style control to the console. **How?** Add a new slot to the page elements using the ConsolPage slot name and the MiniList component. For the list view, you have to set the Mapper property. Optionally you can set the Filter and Sort properties to show the data you want. When the mapper is created, the MiniList flavor is applied, so you can show/hide fields based on this specific flavor.
Use a Detail View	To add a detail style control to the console. **How?** To display a MiniDetail you have to define a small template file to format and layout how the detail will be rendered. This is required because the MiniDetail refers only to a mapper, and not to an existing page. First create a little piece of html that acts as the template for the Mini detail. The template will have fields references to a mapper that provides the data to summarize. The template is saved into the Template folder. Add a record to the Templates list for the template you've just created. Add a new slot to the page elements using the ConsolePage slot name and the MiniDetail component. On the properties of the MiniDetail page element, set the Template property to the template you just created. And set the Mapper property Optionally you can set the Filter and Sort properties to show the data you want. When the mapper is created, the MiniList flavor is applied, so you can show/hide fields based on this specific flavor.

Set up a console style page

Create a page that provides multiple views into different areas of an application. Summaries of record counts with links to filtered lists and/or graphs, etc.

Method	
Use a Graph	Graphs can be displayed from data in a mapper. The graph rendering is provided through Google graphs functionality. **How?** Add a new slot to the page elements using the ConsolePage slot name and the Graph component. The following properties are required for the graphing to work. Mapper CategoryAxis (the key name of the field in the mapper that represents the categories – the x-axis values for simple Line chart). ValueAxis (a semi colon delimited list of fields in the mapper that represent catgegory values – the y-axis value for simple Line chart) ChartType – the type of chart to display. (Line, Pie, 3-D Pie, Vertical Bar, Horizontal Bar). The remaining graphing properties are optional.
Static Content	**How?** Add a new slot to the page elements using the ConsolePage slot name and the UserCtrlHolder. Set the UserControl property to the location of the static content.
Relate the data of the console slots to the Main slot.	This was the original intention of the console page where you will see a main set of information and then display other pieces of information related to that record. This is similar in concept to the page/subform but in this case all the related data is displayed with the main data. **How?** You typically want your Main slot to be a MiniDetail and that requires creation of a template file to format the layout of the detail (see above). For The remaining ConsolePage slots, you relate those to the Main slot through the properties ViewKey and ParentViewKey. When you set the ViewKey property, you are telling the slot that it is related to the Main slot element. The Viewkey is the field on related ConsolePage slot to which the data is filtered If the field on the the related ConsolePage element is not related to the primary key of the Main slot, then you can modify that keyname from which to get a filter criteria, by setting the ParentViewKeySource property.

Add multiple records into a subform at one time (using MultiAdd)

Method	
Set up the subform to support multi-add and configure the subform mapper with a single Find field.	The key to supporting multi find is that the mapper you are adding to must have a single Find field configured on the mapper through which the Multi-Add operates. When a subform is set up for multiadd, the results returned by the find are automatically filtered to exclude results already in the mapper.

How?

Let's say you want to associate a set of people who can act as an approver for another person.
You have a mapper of approvers. The mapper has an approver_id, approver_name and user_id. The user_id field is the foreign key field that relates a user to their approvers. approver_id is the field containing the user_id of each approver. approver_name is the field containing the name of the approver. You want to multiple select approvers from a list of users.

On the subform target you want to add multi find support, set the MultiAdd attribute.
Go to the mapper of the subform target.

So, in keeping with the general convention of Find fields, you would specify the Find field on the related name field approver_name.

You then have to set up the Find field so that you tell the find where to find related users. You do this by setting the FindMOP property to a MOP that uses a mapper you can use for searching. In this example we have a MOP users!detail that is based on the user mapper.

You have to tell the Find how to copy values back to the mapper, by setting the FindFields property. There you set a semi-colon delimited list of field pairs <dest1>=<source1>;<dest2>=<source2>; etc.

Dest – are the fields on the related mapper
Source – are the fields on the source (find) mapper.

approver_id=user_id;approver_name=display_name

Each scenario is different, so you may need to map more or less fields than this to have a correctly functioning multi-add. The critical information, you DO need to provide is how the PK of the find (source) mapper is mapped to the destination (related) mapper.

You may not want certain related records appearing in the search list. You can do this by adding a FindFilter property to filter out unwanted records.

Now, you have to tell the approver mapper which field is the foreign key that associates these approvers to the parent. The parent here is the mapper of the MAIN slot. To specify which field on the related approver mapper should default from the parent, you set the DefaultFromParent attribute on the user_id field of the approver mapper. The core then makes the assumption that the equivalent field on the parent mapper is also called user_id. If it is different, then you can set the ParentField property on the related mapper (where you just set the DefaultFromParent attribute) to tell it the name of the equivalent field on the parent mapper.

It is now possible to test the multi add feature. As you test, you will notice two things. When you add items from the find, the screen behind the find window is refreshed with your selections and the find results are refreshed to automatically exclude the items just chosen. This automatic exclusion filter is added by the platform and is derived from the meta data you have provided to set up the feature

e.g. the find results query for this example would look something like

SELECT * FROM users WHERE user_id NOT IN (SELECT DISTINCT approver_id FROM approvers WHERE user_id='xyz')

Or, more generally

SELECT * FROM find_mapper WHERE source_key NOT IN (SELECT DISTINCT dest_key FROM related_mapper WHERE parent_key=parent_key_value)

Add multiple records into a console list (MiniList) at one time (using MultiAdd)

Method	
Set up the console page element (typically when the element is MiniList) to support multi-add and configure the console mini list element mapper with a single Find field.	This is very similar to the situation of adding multiple records to the subform list. In fact it's probably easiest if you imagine the console list slot element as a subform list of the main console object.

How?

Let's say you want to associate a set of people who can act as an approver for another person. You have a mapper of approvers. The mapper has an approver_id, approver_name and user_id. The user_id field is the foreign key field that relates a user to their approvers. approver_id is the field containing the user_id of each approver. approver_name is the field containing the name of the approver. You want to multiple select approvers from a list of users.

On the console MiniList page element you want to add multi find support, set the MultiAdd attribute. Go to the mapper of the MiniList element.

So, in keeping with the general convention of Find fields, you would specify the Find field on the related name field approver_name.

You then have to set up the Find field so that you tell the find where to find related users. You do this by setting the FindMOP property to a MOP that uses a mapper you can use for searching. In this example we have a MOP users!detail that is based on the user mapper.

You have to tell the Find how to copy values back to the mapper, by setting the FindFields property. There you set a semi-colon delimited list of field pairs <dest1>=<source1>;<dest2>=<source2>; etc.

Dest – are the fields on the related mapper
Source – are the fields on the source (find) mapper.

approver_id=user_id;approver_name=display_name

Each scenario is different, so you may need to map more or less fields than this to have a correctly functioning multi-add. The critical information, you DO need to provide is how the PK of the find (source) mapper is mapped to the destination (related) mapper.

You may not want certain related records appearing in the search list. You can do this by adding a FindFilter property to filter out unwanted records.

Now, you have to tell the approver mapper which field is the foreign key that associates these approvers to the parent. The parent here is the mapper of the MAIN slot. To specify which field on the related approver mapper should default from the parent, you set the DefaultFromParent attribute on the user_id field of the approver mapper. The core then makes the assumption that the equivalent field on the parent mapper is also called user_id. If it is different, then you can set the ParentField property on the related mapper (where you just set the DefaultFromParent attribute) to tell it the name of the equivalent field on the parent mapper.

It is now possible to test the multi add feature. As you test, you will notice two things. When you add items from the find, the screen behind the find window is refreshed with your selections and the find results are refreshed to automatically exclude the items just chosen. This automatic exclusion filter is added by the platform and is derived from the meta data you have provided to set up the feature

e.g. the find results query for this example would look something like

SELECT * FROM users WHERE user_id NOT IN (SELECT DISTINCT approver_id FROM approvers WHERE user_id='xyz')

Or, more generally

SELECT * FROM find_mapper WHERE source_key NOT IN (SELECT DISTINCT dest_key FROM related_mapper WHERE parent_key=parent_key_value)

Make a field available based on version control.

Method	
Use Versions settings and object specific version attributes.	Versioning is only available on Field objects. This is typically how the addition of new features (and the fields associated with those features) is controlled so that the new features don't appear until the feature is ready. The versioning specified here can also be referred to in code to version control other objects related to new features.

How?
Go to the Versions list and add a new Version. For the version you add, you have to provide a version number. When you add a field to a mapper that you want to be controlled by version, you set the Version attribute to the name of your version. Then, for that field, you specify either a minimum or maximum version (or both). If you specify a minimum version then the version number of your version has to be equal to or greater than the version in the MinVersion attribute, for that field to be included in the mapper. If the version is less, then the field is excluded from the mapper. If you specify a maximum version then the version number of your version has to be less than or equal to the version in the MaxVersion attribute, for that field to be included in the mapper. If the version is greater, then the field is excluded from the mapper.

Create a list view and provide predefined links to filter the list

On a list view there are static links above the list that allow a user to quickly filter the list view

Method	
Associate a navbar with a page.	**How?** Create a navbar (if one does not exist). Then for the new navbar, add targets to the NavBar that navigate to the same list view page. This is important. For each NavTarget, set an appropriate Filter property and also a Caption that is meaningful for the filter criteria. The Caption should be as short as possible. The first NavTarget should have a filter that matches the filtering applied when the page is navigated to, from the Tree. That's because the first filter link is highlighted and that implies to the user that the highlighted link and the filtered records refer to each other. Now, go to the list view page you've specified as the target for NavTargets. If the page in question has a base page defined, the you must removed the BasePage attribute for that page. In Page elements subform for the Page, specify the page elements that make up the base page elements. E.g., Side slot refers to NavBar and uses the main navigation tree for the Navigator property. The MainSlot uses a dsctrl (for list view rendering). Then for the Navigator property of that slot, set it to the name of the Navbar you created, or are re-using.

Make a printable version of a page

Method	
Create a new page based on an existing page that uses a different template.	On a standard page layout, the parent information (list or detail) is visible and typically only one subform. When you print such a page, only those visible parts are printed. There is a way to create a slightly different view of the same page (using a different page template) where all the subform items are displayed in a vertical layout so you can print all the related items.

How?

Go to the page you want to print.
Right click on the page and then duplicate.
On the duplicated page (identified by the Moniker having the name “copy of..”, change the Name to “Print” and also the Moniker to something appropriate.
Change the Template to ExpandedSubformTemplate.
When the page was duplicated, the layout was also copied. If the layout of the original page is modified subsequently, then the print page would also have to be modified. The solution to this is to refer the layout of the print page to the original detail page.
E.g., if the original page is people!detail and the print page is people!print, then on the people!print page, set the DetailLayout property to people!detail.
Set the PrintMode Page attribute. This forces the Print flavor to be applied to the page and all subforms and forces all the content to be rendered into a single IFrame.
Now go to the mapper.
Add a button to the mapper, btn_print that is exclude from select. Set the Excluded flavor to Print
Set the Caption Property to Print View, the LinkKey to people_id, the LinkMOP to people!print
Duplicate the btn_print field
On the second btn_print field, set the Exclude flavor to 0 and the Include Flavor to Print.
Set the OnClick property to “window.print();”
Set the CssClass property to “np”. “np” means the button will not be on the print output
Remove the LinkKey and LinkMOP properties
Change the Caption to “Print”

Make a page override the view of its underlying mapper

Method	
Alter an existing page to use a different view than the view in the underlying mapper	There are instances where you have a detail page on which you want to display a couple of fields, say, that are not normally visible to any other user role. Ordinarily you would set visibility permissions on a field to hide these from other user roles. However, the fields you need to show for this one particular case mean that you would have to join another table into the existing view. By adding another table in the view, there is likely to be a performance penalty. You don't want the view slowed down for one corner case of displaying a field on one version of a detail screen. What you want therefore is to have a different view when you display the detail page for this particular user role. The performance penalty to display a single record from this view is minimal. **How?** Assuming you have a view that is almost identical to the original but with an additional field added from the new join. On the page you want to display the extra field, go to the permissions and unset the read permission for the role that will use the new view. Now duplicate the page and set the read permissions for the new role and unset read permissions for other roles. Rename the moniker of the duplicated page. Set the View property of the new page to the new view. At runtime this will override the view of the mapper. On the new page, set a custom flavor on the Flavor property. Say Custom1. Go to the mapper of the page. You will need to manually add the fields to the mapper definition. On these added fields, set the Include Flavor to Custom1. This is because these fields can only be selected when we are using the alternative view.

Make a list/detail navigate to a different page for detail, delete or new action

Method	
Change the navigation target on the page properties.	When you create a page, the application assumes that the "detail" view of the data for that page is called <module>!detail. That's ok for simple situations and invariably, most modules with have a list view called <module>!list and a detail page called <module>!detail. When you click on the list view to go to a detail, unless instructed, the system attempts to navigate to the <module>!detail page. Likewise if attempting to create a new record if there is no page called <module>!new. If you delete a detail record, the default action is to navigate to a blank page. You can override this behavior in page properties.

How?
On the list view page, go to the properties and set the "DetailTarget" Property to the page you have set up as the detail for the list. You may also set the "NewTarget" property to change the page that is opened when the user clicks on the New button from the list view. On the detail view page, go to the properties and set the "NewTarget" property to change the page that is opened when the user clicks on the New button from the detail view. Setting the DetailTarget property on a detail page is little bit meaningless here. You may also set the "DeleteTarget" property to provide a more user friendly page to navigate to after deleting the record from the detail screen. You can navigate to any page and even pass navigation parameters to open a specific record.

Summarize data in a list view showing aggregated values.

Method	
Set the built in aggregation properties on mappers and pages.	The application provides a way to set up simple aggregation output on list views. The aggregation options match those provided by the DBMS as the ag.NET as the aggregation is You can turn on aggregation at the mapper level and/or page level and any fields that natively support aggregation (that is numeric fields) will be aggregated with a Sum aggregate. **How?** On the mapper, set the Display attribute on AggregateDisplay property.
Control when aggregation used at page level	Whenever a list view uses the mapper, it will not show aggregation by default unless the page allows aggregation. **How?** On the mapper, set the AllowPageToDisplay attribute on AggregateDisplay property. On a page (alist view page) where you want aggregation to appear, set the DisplayAggregates attribute on the DisplayAttributes property.

Group data in a list view

Method	
Use grouping properties on a mapper	The mapper supports a very simple approach to grouping in that it can only perform a group on one column in the list **How?** On the mapper you want to have grouping, go to the properties and in the DefaultGrouping property, specify the keyname of a field on your mapper.
Group on more than one column	As already mentioned, the mapper is unable to group on more than one column at a time, so you have to work around this problem by forcing the grouping of multiple fields into one field **How?** First identify the fields you want to group. In the underlying view of the mapper, create an additional field that concatenates the values of the fields you want to group. .e.g, ISNULL(company,'') + ' ' + ISNULL(department,'') AS group_by Use the slurper to add the new field from the view into the mapper. On the mapper, set the DefaultGrouping property to group_by (the name of the column you just added). If you do not want the group by field to appear in the list then set the column width to 0 or set the hide flavor on the field to ListView. The list will still be grouped on the field.

Putting information about the record in the Caption of a detail

Method	
Use a field expression in the caption property	You can display data from the mapper in the caption area of the page that is not static content. Note that this feature only works with detail pages, not list pages. **How?** On the page you want the customized caption, enter a caption that contains one or more field references from the underlying mapper. You can also specify vocabulary strings using the !fnVocab() syntax.

On certain pages, the new, save, delete buttons need to be hidden.

Method	
Use the ToolbarDisplayAttributes property on the page	The toolbar attributes property on the page allows you to control whether the buttons are visible to the user. The button visibility is controlled by the mapper level by DenyInsert, DenyUpdate and DenyDelete attributes. When you want to control the visibility from the page level, you must not set the mapper attributes. The page cannot override the mapper settings of DenyInsert, etc, to allow insert. It can only override mapper to deny insert if the mapper allows insert. **How?** On the page, go to the ToolbarDisplayAttributes property and set one or more of the settings To prevent multi-add on a subform(add) – Set DenyAdd To prevent delete(delete) – Set DenyDelete To prevent insert(new) – Set DenyInsert To prevent update(save) – Set Deny Update

Set up a mapper to perform logical deletes, instead of physical deletes.

Method	
Set up the logical delete property on a mapper	Supporting logical delete is extremely easy to configure. The idea is to associate one field on the mapper (only one field) that represents the "delete" status of a record. The field should be an integer field containing either zeroes (deleted) or ones (not deleted) **How?** On the mapper that should support logical delete, Set the LogicalDelete mapper attribute. Then set the LogicalDelete mapper property to one of the fields in the mapper.

Set up a mapper to perform row revisioning.

Method	
Set up the row revisioning properties on a mapper	You would use row revisioning when you want to set up a mapper to detect when a row has been modified and saved while another user has modified and saved the same record. If the row is saved after another user has modified the record. The field containing the revision information is an integer field that is incremented every time the record is saved. When the record is saved a second time, the user is presented with an error message indicating they should refresh and re-apply their changes. **How?** On the mapper you want row revisioning, set the RowRevision property to an integer field on the mapper.

Use a Tree Control Selection to Filter a Detail

Method	
Tree control set up as a filterer	You can use a tree control to filter a detail control. The key here is that the detail control is not initially filtered to a single record. Instead the filter on the detail is one that you would normally use to filter for a list view. When the page first opens, you see the first record from the result set on the detail. When you click on one of the items in the tree view, it takes the id of the selected item and filters the underlying mapper with the selected id. Typically you're going to perform this type of task on a subform detail.

How?
On the mapper you want to have the tree, Select the field that contains the key you want to have the additional filter from the tree. Set that field to be the Tree cell type. Mark the cell attributes for the field as Filterer. Optionally you can specify one of the available .Net Image sets to display on tree. You also have to provide a picklist to populate the tree. You can just use one of the meta data picklists from the selection, but probably you want to construct a picklist in code and set the picklist content dynamically. The content of the picklist will match the content of the mapper list. On the detail page, make sure your filtering is set up to return multiple records from your base filter. As though it was list view page.

Modifying the default toolbar buttons on a page

The buttons for your toolbar are by default in the main Images folder. However, it is likely that you have application specific buttons in a folder wwwroot\apps\{{appname}}\images\buttons. To check you would go to the Application in the studio and check the ButtonImagePath property in the Appearance category of properties. In that folder will be button images for the Save, Refresh, Next, Previous, etc buttons. For all pages, these buttons use the same image.

However you can change what images are used for these buttons by setting the ButtonImageOverride property on a page. Of course if you want to change the button image for all pages, simply change the actual button image in the ButtonImagePath.

The ButtonImageOverride property expects and set of key, value pairs separated by a semi-colon (a tag string). The key represent the original name of the button. The value represents the relative path to the other button image to replace the default image.

For example. Setting the ButtonImageOverride property to the following

SaveLg.gif=NewSaveLg.gif;DelLg.gif=NewDelLg.gif;RefreshLg.gif=newimgs/RefreshLg.gif

Will change the image of the Save Button, Delete Button, Refresh Button on a page. The refresh button property is set to a subfolder of Button images wwwroot\apps\{{appname}}\images\buttons\newimgs

Image Name (key)	Use
Add16.gif	used for multi add
AddLg.gif	
Del16.gif	used for delete
DelLg.gif	
Edit16.gif	used for edit
EditLg.gif	
Filters16.gif	used for filter
FiltersLg.gif	
Help16.gif	used for help
HelpLg.gif	
Links16.gif	used for links menu
LinksLg.gif	
More16.gif	used for more/action menu
MoreLg.gif	
New16.gif	used for new
NewLg.gif	
Next16.gif	used for next
NextLg.gif	
Platform16.gif	used for list view layout management
PlatformLg.gif	
Prev16.gif	used for previous
PrevLg.gif	
Print16.gif	used for print
PrintLg.gif	
Refresh16.gif	used for refresh
RefreshLg.gif	
Save16.gif	used for save
SaveLg.gif	