Introduction

XMod revolutionized how solutions were created for the DotNetNuke environment. Instead of having to be a web programmer or hire one to develop each of your modules, XMod enabled anyone with a basic knowledge of HTML to easily create as many different modules as they needed.

Although XMod has been a best-seller virtually every day since its release, we started to hear from people building web sites that needed more direct control over the database on the backend. What they wanted was the flexibility and convenience of XMod combined with the ability to connect it to their own data. XMod Pro is our answer to the needs of our power users and is the next step in the evolution of XMod.

With XMod Pro, you are in control. You create templates and forms made up of standard HTML. You can create templates for displaying lists of data and templates for displaying a single record. But how does XMod Pro link up with our data? You simply supply standard SQL commands such as SELECT commands and stored procedures. Then, for templates, where ever you want to display the value of a given field, use a field "token" like "[[FirstName]]" to display the value in the FirstName field. For forms, you simply assign a field name to a given control to 'bind' it to that field.

Optionally, you can add a few XMod Pro tags to your templates that give you additional functionality like detail buttons to show the detail template, a format tag to format your numbers, dates, etc. for display and much more. The result is "your" data, displayed "your" way - no programming required.

XMod Pro forms also come with a complement of controls you can use to build your data entry user interface including: Textbox, Textarea, HTML Input, Radio Button, Check Box, Radio Button List, Check Box List, Drop-Down List, Date Input, File Upload, and more. Plus you can build or buy additional custom controls - providing you maximum flexibility.

The original XMod is a great product for quickly building solutions for DotNetNuke. However, interacting directly with your tables and data required we develop a new foundation. As a result, XMod Pro is a completely independent product, with a new form and display engine. Even so, we worked hard to maintain the same type of syntax that devoted XMod users have come to love. Not every thing is the same, but XMod users will feel right at home in XMod Pro.

Next Step: Getting Started
Activating Your License

In order to use XMod Pro on a web site, you must activate the license. Please note that you may use XMod Pro for development purposes only on "localhost". For all other uses you must activate the license. This is done through the License and Activation page. This page is only available to Hosts or SuperUser accounts. To get there, select "License and Activation" from the module's Actions Menu:

The License and Activation page allows you to check the status of your license. If you haven't yet activated, this page allows you to request a Trial License (a temporary license that will expire) or activate your license.
Requesting A Trial License: Tick the "Trial License" check box and click the "Request Activation" or "Manual Activation" button.

Activating Your License:
- Make sure the Trial License check box is not ticked.
- Account: Enter the email address you used to purchase the license.
- Invoice Number: If you purchased from Snowcovered, enter the Invoice Number found on your receipt. If you purchased from PayPal, use the Transaction ID, which can be found on your receipt.
- Ensure you have a connection to the internet.
- Click the "Request Activation" button. If you are behind a firewall or you’re having difficulty activating, click the "Manual Activation" button.
- Click the "Update" link to save your changes.

Manually Activating Your License:
Fill in the Account and Invoice Number text boxes. Ensure this information is correct.
Click the "Manual Activation" button
The following appears:

Use CTRL+A to select all the contents of this box. You will need to copy the contents and paste them into the manual activation page on our site.

Click here to go to the manual activation page on our site

Click this button to activate your software

Press CTRL+A on your keyboard to select ALL the information in the Server ID box and copy that to the clipboard. You will need to paste that information into our manual activation page.
Click the link below the Server ID box to browse to our manual activation page and follow the instructions on that page. Paste the text copied from the Server ID box into the "STEP THREE" text box on the manual activation page.
The manual activation page will generate an Activation Key for you. Copy and paste that text into the "Activation Key" text box on the License and Activation page in XMod Pro.
Click the "Submit Activation Key" button to activate your software
Click the "Update" link to save your changes.
Getting Started

XMod Pro solutions consist of two fundamental components: forms and templates. Forms are used to add and edit data while templates are used to display data. While forms and templates generally work with the same data, there are no rules in XMod Pro that require that be the case. It's possible to use a form and template in the same module instance that operate on distinct data though most of the time they'll usually be linked.

Multiple Views - One Module Instance

Even though each module instance can have only one form, it may contain multiple templates. This is really the heart of what makes XMod Pro so powerful and flexible. It allows you to have multiple views of data within the same module instance space on your page. Each template functions independently but can interact with the others.

For instance, clicking the Headline link in a list of headlines in one template can call up the full article in a second template. Taking it a step further, clicking that Headline can not only call up the full article, but it can also spawn a list of related articles the user may be interested in. All this happens with a single click.

Here's another, less interactive, but no less powerful example. Take the humble customer invoice. In one area, you display the customer's name and address (from the Customers table). Below that you can display the customer's list of orders (from the Orders table, filtered by the Customer's ID). Below that, you can display a list of payments the customer has made (from the Payments table, filtered by the Customer's ID). This can all be displayed within a single module instance.

What's even better is you don't have to do any programming to pull this off. All you supply is a few SQL commands (SELECT statements or stored procedure calls) and combine that with your own HTML and XMod Pro's simple field tokens. At run-time, those tokens are replaced with the values from your query. You can use as much or as little HTML, CSS, Javascript as you need to create the look you desire.

XMod Pro also provides a number of HTML-like tags that provide additional functionality. For instance, if your data contains a currency value, you can use the Format tag to format it with a dollar sign and two decimal places. That same Format tag can also format dates, numbers, and text. Do you need to conditionally display some text based on the user's security role or based on some other condition? Then use the Select tag.

The upshot of all of this is you can have a complete database solution using *your* data and presented according to *your* needs, with no programming, and remarkably little work. We don't ask you to learn a complicated new programming language or script.

If you've used XMod, you'll feel right at home in XMod Pro. Though XMod Pro was written from scratch, it still adheres to the spirit of XMod and shares a number of its conventions. If you haven't used XMod, don't worry. If you know basic HTML and can write basic SQL statements to retrieve, insert, and update data you will have no problems getting started with XMod Pro.

CAUTION:

As its name implies, XMod Pro is designed for professionals. It gives you a lot of power and with that power, it is your responsibility to ensure the integrity of your data by validating user input (we provide tools to help you do this) and maintaining recent backups of your data. This is true for any data-based application and isn't unique to XMod Pro. For instance, a poorly written DELETE command could delete all the records in your table. The developers and owners of XMod Pro are not responsible for any data loss or data corruption or any related costs or damages caused as a result of using XMod Pro.
Control Panel

Beginning with version 2, most administrative tasks have been consolidated and placed in the Control Panel. It is only available to Host and SuperUser accounts. You can access the Control Panel by selecting it from the Actions menu:

On the Control Panel page you'll see the following toolbar:

A brief description of each area is provided below. Click the links to see a more detailed explanation of each area.

**Manage Forms**: In this area you can add, edit, rename, copy, and delete forms

**Manage Templates**: In this area you can add, edit, rename, copy, and delete templates

**Database Tools**: In this area you can create basic tables in your DNN database and, optionally, auto-generate forms and templates from the newly created table.

**Help**: The help page provides a list of videos and instructional blog posts which appear periodically on our site. These items are updated as they are added to our site, so check the Help page periodically to see what has been added.

**Exit**: This exits the Control Panel and returns you to the previous page.
Creating, Editing, Copying, Renaming, and Deleting Forms

Don't Feel Like Reading? Watch the videos instead:

- **Form Manager Grid**: Takes you on a tour of the Form Manager grid on the Manage Forms page and describes the difference between Custom and Auto-Layout forms.
- **Form Builder Tour**: Creating a Contact Us form with Email notification.
- **Creating A Data-Bound Form**: Shows how to use the Form Builder to create forms that add and edit records in your database.
- **Creating Picker Lists**: Builds on the data-bound form video and shows you how to setup drop-down lists that are populated from your database.

Data entry forms are created, edited, copied, renamed, and deleted within the Control Panel's Manage Forms page. The Manage Forms page is only available to Hosts or SuperUser accounts. To get there, select "Control Panel" from the module's Actions Menu:

![Control Panel Menu](image)

On the Control Panel Page, select "Manage Forms":

![Manage Forms](image)

As you can see in the image below, any forms you've already created will be displayed within a two-column grid.
Buttons to Preview ✎, Edit 🖋, Rename 🔗, Copy 🖇️, or Delete ✗ each form are listed in the left column, followed by the form's name. The right-most columns list the date and time the form was last Modified and when it was originally created. You can sort your forms by clicking the header of each column. Click it a second time to reverse the sort order.

**Form Types**

Under the Type column, you can see if the form is a Custom HTML layout form or an Auto-Layout form (added in version 3.0). An Auto-Layout form has been created by and is editable by the Form Builder. You can edit an auto-layout form by converting it to a Custom HTML Layout.
form. This is done by clicking the "Auto-Layout" link for the form in the Type column. Once you do this, XMod Pro will convert the form as a new form. This allows you to convert a form non-destructively.

Reloading and Navigating the List of Forms

To enhance performance, XMod Pro caches the list of forms. If you have added a form via the Form Builder or via some other means, your new form may not appear in the list. You can rectify this by clicking the Reload button found on the left side of the grid's navigation bar.

Also on the navigation bar, you'll find the usual elements: buttons to page to the first page, previous page, next page, and last page. You can type the number of the page into the text box and go directly to that page. You can also select how many items to show by selecting an appropriate number from the drop-down list box.

Previewing A Form

Click the magnifying glass icon on the row of the form you want to preview.

The Add Form will be displayed. If you have added validation to your form, you can click the Add button to show the validation messages:
NOTE: While the preview does not allow you to submit data, since it is a live preview, some form controls like the FileUpload control may be functional. Additionally, the preview is intended to give you a sense for how a form will look at run-time. Because it doesn't save data, it also does not process tokens like User and other tokens. Depending on how you've setup your form, this may affect the display.

**Renaming A Form**

Click the Rename image next to the name of the form you want to rename. The row will change to look similar to this:

Change the name to the desired text and click the green Save icon to save your changes. If you wish to cancel out of the operation and keep the name, click the red Cancel icon.

**Copying A Form**

Click the Copy image on the row of the form you want to copy. XMod Pro creates a duplicate of the form, gives it a new name, appending a number to the end of the name, and places it just below the row of the form you're copying. From there you can rename it to whatever name you'd like.
Deleting A Form

Click the Delete image on the row of the form you want to delete. XMod Pro prompts you to confirm you want to delete the form. Click the Delete button on the dialog and your form will be deleted. Click the Cancel button if you do not want to delete the form.

Editing A Form

Click the Edit icon on the row of the form you want to edit. The form editor is displayed and loaded with the selected form.

1. **Form Name**: The name of the form you're editing is displayed above the editor
2. **Help Topics**: XMod Pro provides online help for form controls and tokens. You can access it by selecting a topic from the drop-down list.

3. **Tabstrip**: New to version 2.7. Easily switch between editing the Add form and Edit form by clicking on the appropriate tab.

4. **Toolbar**:
   - Select some text in the editor and click these buttons to wrap the text in Bold or Italic HTML tags.
   - Inserts the skeleton structure of an HTML table

   - **Tags Dropdown**: Allows you to select from and insert a form control tag from a list of available controls. When you select a tag, a designer will pop-up enabling you specify the tag's properties. An example of the Checkbox List control's dialog is below:

   ![Checkbox List Dialog](image)

   - **Tokens Dropdown**: Allows you to select from a list of XMod Pro tokens. The token will then be inserted into your code.

5. **Editing Area**: New to version 4, the editor now has colored syntax highlighting, line numbers, auto-complete of XMod Pro tags, search and replace, auto-indent, and block indent/outdent. Note in the image that `<HeaderTemplate>`, `<ul>`, and `</FooterTemplate>` are highlighted in red. This indicates invalid HTML syntax. However, since templates dynamically build the HTML at run-time, these error indicators can be ignored.

6. **Update/Cancel Buttons**: Click the Update button to save your changes and return to the main Manage Forms screen or click Cancel to abandon your changes and return to the Manage Forms screen.
Creating A New Form

There are two ways to create a new form in XMod Pro. The Form Builder and the Plain Text Editor. Rather than try to describe all the facets of the Form Builder, we’ve created some videos that show you instead. These videos are hosted on the internet and, thus, require internet access.

- **Form Builder Tour**: Creating a Contact Us form with Email notification.
- **Creating A Data-Bound Form**: Shows how to use the Form Builder to create forms that add and edit records in your database.
- **Creating Picker Lists**: Builds on the data-bound form video and shows you how to setup drop-down lists that are populated from your database.

Creating a Custom HTML Layout Form

The Form Builder is a great way to create forms quickly and easily. Even if you plan to customize it later, we recommend looking at using the Form Builder to jump-start your form development and the converting that form to a custom HTML layout form. The Custom HTML Layout form type gives you the greatest flexibility and options when it comes to layout, styling, and interactivity. Of course, that additional power means you’ll have to do some more work. The Plain Text Editor was created for that purpose. See the section above to see how the editor functions.

To create a new custom HTML layout form, un-tick the "Use Form Builder" checkbox and click the New Form button at the top of the Form Manager grid. You’ll be presented with a screen similar to what you see below.

1. Enter the name to use for the form. You can only use Letters, Numbers, hypens (-) and Underscores (_) in the form name. Special characters and spaces are not allowed.
2. If you want XMod Pro to auto-create a basic add and edit form for you, select a Data Source. You can choose a table in the DNN database or one in an External SQL Server database. If you choose the latter, you’ll need to supply a connection string. If you just want a bare-bones form skeleton, choose "None".
3. If you choose DotNetNuke Database as your Data Source or, if you’ve chosen External SQL Server Database, entered a connection string and clicked the Load Tables button, you’ll see a drop-down list with tables in your datasource. Select the table you want to work with.
4. Once you’ve selected a table, you’ll need to select a Key Field. This is the column in the table that uniquely identifies each record. It is required so that XMod Pro can set up editing for the record.
5. Finally, click the Create button to have your form generated (refer to the plain text form editor description earlier in this topic for usage details). Click the Cancel button to go back to the Form Manager grid.
If you click the Create button, your form will be generated and placed into the form editor, ready for editing. The generated form definition provides the Add and Edit forms, complete with the data commands to insert, retrieve, and update the record. It also provides a basic, CSS-based layout that you can alter or remove. It also makes educated guesses as to which controls to use for your form. Make any changes you want to the form. Once you are satisfied with your edits, click the Save button.
Adding, Editing, Copying and Deleting Templates

Data display templates are created, edited, copied, and deleted within the Manage Templates page. The Manage Templates page is only available to Hosts or SuperUser accounts. To get there, select "Control Panel" from the module's Actions Menu:

![Control Panel Menu](image)

On the Control Panel Page, select "Manage Templates":

![Control Panel Page](image)

As you can see in the image below, any templates you've already created will be displayed within the Template Manager grid.
Buttons to Preview, Edit, Rename, Copy, or Delete each template are listed in the left column, followed by the template's name. The right-most columns list the date and time the template was last modified and when it was originally created. You can sort your forms by clicking the header of each column. Click it a second time to reverse the sort order.

**Reloading and Navigating the List of Templates**

To enhance performance, XMod Pro caches the list of templates. If you have added a template via the Database Tools page via some other means, your new template may not appear in the list. You can rectify this by clicking the Reload button found on the left side of the grid’s navigation bar.

Also on the navigation bar, you’ll find the usual elements: buttons to page to the first page, previous page, next page, and last page. You can type the number of the page into the text box and go directly to that page. You can also select how many items to show by selecting an appropriate number from the drop-down list box.

**Previewing A Template**

Click the magnifying glass icon on the row of the template you want to preview. NOTE: This is a Live preview, with actual data from your site. Not all the styles of your skin are applied to the template for the preview. Therefore it may look different when viewed at run-time. Additionally,
any Edit/Delete buttons will not appear in the preview. You can, however, click Detail buttons/links to see the detail template.

NOTE: Not all styles from your site are applied to the preview. Therefore, it may appear different when added to a page.

When you are done with the preview, click the Return button to return to the Manage Templates grid.

**Renaming A Template**

Click the Rename image next to the name of the template you want to rename. The row will change to look similar to this:

Change the name to the desired text and click the green Save icon to save your changes. If you wish to cancel out of the operation and keep the name, click the red Cancel icon.

**Copying A Template**

Click the Copy image on the row of the template you want to copy. XMod Pro creates a duplicate of the template, gives it a new name, appending a number to the end of the name, and places it just below the row of the template you're copying. From there you can rename it to whatever name you'd like.
Deleting a Template

Click the Delete image on the row of the template you want to delete. XMod Pro prompts you to confirm you want to delete the template. Click the Delete button on the dialog and your template will be deleted. Click the Cancel button if you do not want to delete the template.

Editing a Template

Click the Edit icon on the row of the template you want to edit. The template editor is displayed and loaded with the selected template.

1. **Template Name**: The name of the template you’re editing is displayed above the editor
2. **Help Topics**: XMod Pro provides online help for template controls and tokens. You can access it by selecting a topic from the drop-down list.
3. **Toolbar:**
   - ![Bold](B) ![Italic](I) Select some text in the editor and click these buttons to wrap the text in Bold or Italic HTML tags.
   - ![Table](T) Inserts the skeleton structure of an HTML table
   - Tokens Dropdown: Allows you to select from and insert an XMod Pro token from a list of available tokens you can insert (such as `[[Module:ID]]`, `[[Portal:ID]]`, `[[User:DisplayName]]`, etc.)
   - Tags Dropdown: Allows you to select from and insert a template tag from a list of available tags.

4. **Editing Area:** New to version 4, the editor now has colored syntax highlighting, line numbers, auto-complete of XMod Pro tags, search and replace, auto-indent, and block indent/outdent. Note in the image that `</HeaderTemplate>`, `</ul>`, and `</FooterTemplate>` are highlighted in red. This indicates invalid HTML syntax. However, since templates dynamically build the HTML at run-time, these error indicators can be ignored.

5. **Update/Cancel Buttons:** Click the Update button to save your changes and return to the main Manage templates screen or click Cancel to abandon your changes and return to the Manage templates screen.

**Creating A New Template With the Quick Template Creator**

The Quick Template Creator is a fast way to generate the skeleton code for a template or to generate a list and detail template from a table in your DNN database. When you choose this option, XMod Pro will not only generate a template of your choice but will also create the data commands and controls to link your list and detail templates.

To start, click the New Template button at the top of the grid.
1. To create a new template, first choose the type of view you’d like to create.

2. Choose the source for your data. Options include None, DotNetNuke database, or External SQL Server database.

3. If you select the "DotNetNuke" data source, you can select a table in the database. If you select External SQL Server database, you’ll first need to input a connection string to the database. Then you’ll be able to load a list of tables from the database.

4. Next, you are asked to select a column from that table to use as the Unique Identifier for records from that table. This column will be used when creating the Detail template and the controls to connect the list and detail templates, so that the proper record can be retrieved.

5. Next, choose the columns (fields) the builder should include in your list view.

6. If you’d like the builder to generate a basic detail starter template for you, check this box.

7. If you choose to include a detail view, select which fields you’d like to include in that view.

8. **Choosing Buttons and Setting Permissions**: By default, all buttons are created for you. However, you can choose which buttons to include by selecting the Buttons tab:
Tick the check box next to each button type you’d like to include. You can also click the arrow next to the checkbox to expand/collapse the settings section for each button.

Within each settings section, you can choose how the button will be displayed - as a hyperlink, a button, or a clickable image. You can specify the text to include on the button (or alternate text if the button is an image). If the button is an image you’ll be able to specify a URL to the image.

Next, for each button, you can determine which DNN security roles will be able to see and use the buttons.

9. Finally, click the "Save" button and your template will be generated and placed into the template editor, ready for editing. The generated template definition provides the list and detail templates and their data commands along with some basic styling. If selected, it also provides a button to view the record's details, a button to Edit the record, and a button to create a new record. The Edit button is set to pass the record's unique identifier to a form. It is up to your form to use that ID when editing the record. Once you are satisfied with your edits, type a name for your new template into the "Template Name" text box and click the Save button.
<ms:Template UsePaging="true" Arg="Sales" AddRoles="Role! EditRoles="" DeleteRoles="" DetailRoles=""">
<ms:listDataSource CommandText="SELECT [Title], [Synopsis] FROM Books"/>
<ms:Parameter Name="BookId"/>
<ms:DeleteCommand CommandText="SELECT * FROM Books WHERE [BookId] = $BookId"/>
<ms:Parameter Name="BookId"/>
</ms:Template>

<table>
<thead>
<tr>
<th>Title</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
</tbody>
</table>
</ms:Template>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head></head>
<body>
<table>
<tbody>
</tbody>
</table>
</body>
</html>
Adding, Editing, Copying and Deleting Feeds

NEW TO VERSION 3.0: Feeds are a new feature as of XMod Pro 3. They greatly expand what you can do with XMod Pro. With them, you can create RSS feeds, export data as CSV files, convert SQL data to XML files, even use them to display printer-friendly HTML files -- all using the same types of techniques you use to create templates.

Feeds are created, edited, copied, and deleted within the Manage Feeds page. The Manage Feeds page is only available to Hosts or Super-User accounts. To get there, select "Control Panel" from the module's Actions Menu:

On the Control Panel Page, select "Manage Feeds":

As you can see in the image below, any feeds you've already created will be displayed within the Feed Manager grid:

Note: if your feed relies on parameters that are passed-in to the feed, you may not be able to preview it in this interface. You can access the live feed in your browser (see the "How To: Call A Feed" link above the grid for more information).

Buttons to Preview, Edit, Rename, Copy, or Delete each feed are listed in the left column, followed by the feed's name. The right-most columns list the date and time the feed was last Modified and when it was originally created. You can sort your forms by clicking the header of each column. Click it a second time to reverse the sort order.
Reloading and Navigating the List of Feeds

To enhance performance, XMod Pro caches the list of feeds. If you have added a feed via the Database Tools page via some other means, your new feed may not appear in the list. You can rectify this by clicking the Reload button found on the left side of the grid's navigation bar.

Also on the navigation bar, you'll find the usual elements: buttons to page to the first page, previous page, next page, and last page. You can type the number of the page into the text box and go directly to that page. You can also select how many items to show by selecting an appropriate number from the drop-down list box.

Calling A Feed

Feeds aren't very useful if you can't use them. So, we've given you a quick link on the main page to find out how to call your feed. Clicking the link will pop-up a dialog similar to this:
As the dialog explains, basically you're calling Feed.aspx in the XMod Pro directory and supplying it with the name of your feed and the portal ID where the feed resides.

**IMPORTANT:** Feeds are assumed to be public. There is no special precaution that XMod Pro makes to secure your feed. Should you wish to limit who can view your data, take care in how you formulate your ListDataSource to avoid letting un-authorized users from viewing your data.

**Previewing A Feed**

Click the magnifying glass icon on the row of the feed you want to preview. **NOTE:** This is a Live preview, with actual data from your site.

**Control Panel**

If you have an application installed to load a CSV (comma-separated-values) file, then you may be prompted to open it in that program, just like your user would.
Note that when previewing a feed, the display may not look how you intend it. That is because you are previewing it in a browser — as HTML.

In the example above, we've created an XML feed, but you can't see any XML tags in the preview. To do so, in most browsers, right-click inside your preview and select Inspect Element. This will show the underlying code as here (Firebug is being used):
When you are done with the preview, click the Return button to return to the Manage Feeds grid.

**Renaming A Feed**

Click the Rename image next to the name of the feed you want to rename. The row will change to look similar to this:

![Renaming Feed](image)

Change the name to the desired text and click the green Save icon to save your changes. If you wish to cancel out of the operation and keep the name, click the red Cancel icon.

**Copying A Feed**

Click the Copy image on the row of the feed you want to copy. XMod Pro creates a duplicate of the feed, gives it a new name, appending a number to the end of the name, and places it just below the row of the feed you're copying. From there you can rename it to whatever name you'd like.

![Copying Feed](image)

**Deleting A feed**

Click the Delete image on the row of the feed you want to delete. XMod Pro prompts you to confirm you want to delete the feed. Click the Delete button on the dialog and your feed will be deleted. Click the Cancel button if you do not want to delete the feed.
Editing A feed

Click the Edit icon on the row of the feed you want to edit. The feed editor is displayed and loaded with the selected feed.

IMPORTANT: Feeds are very similar to templates, but they serve a different purpose. Not all tags that work in templates will work in feeds.

Books_CSV
1. **Feed Name**: The name of the feed you're editing is displayed above the editor.
2. **Help Topics**: XMod Pro provides online help for feed controls and tokens. You can access it by selecting a topic from the drop-down list.
3. **Toolbar**:
   - Select some text in the editor and click these buttons to wrap the text in Bold, Italic, or Strikethrough HTML tags.
   - Inserts an HTML image tag (<img>)
   - Inserts an HTML hyperlink tag (<a>)
   - Inserts the skeleton structure of an HTML table
   - Allows you to select from and insert an XMod Pro token from a list of available tokens you can insert (such as [[Module:ID]], [[Portal:ID]], [[User:DisplayName]], etc.)
   - Allows you to select from and insert a feed tag from a list of available tags.
4. **Resizing Handle**: To expand or shrink the size of the editor, click and drag this handle.
5. **Update/Cancel Buttons**: Click the Update button to save your changes and return to the main Manage feeds screen or click Cancel to abandon your changes and return to the Manage feeds screen.

**Creating A New Feed With the Quick Feed Creator**

The Quick Feed Creator is a fast way to generate the skeleton code for a feed or to generate a list and detail feed from a table in your DNN database. When you choose this option, XMod Pro will not only generate a feed of your choice but will also create the data commands and controls to link your list and detail feeds.

To start, click the New Feed button at the top of the grid.
1. To create a new feed, first choose the type of feed you’d like to create. There are a few pre-defined options like Comma-Separated Values and XML, but you can choose Custom and put in whatever you’d like.

2. Optionally, specify a file name for your feed. Doing so enables your user to download the output as a file. This allows them to save it on their hard drive and/or open it with another program.

3. At its simplest, you can create a basic skeleton of a feed, simply by clicking the New feed button. However, to get a jump-start on your feed development, tick the "Generate Feed from Table" checkbox.

4. This presents you with a drop-down list where you can specify where your table is located. You can work with a table in your DNN database, as shown here. Or you can use a table in an External SQL Server database. If you choose the latter option, simply specify the connection string and click the Load Tables button to continue.

5. Once your data source has been specified, XMod Pro will load a list of tables for you to choose from. Choose the one you’re interested in.

6. Next, select the columns from the table that you’d like to include in your feed. SHIFT-Click or CTRL-Click to select multiple columns.

7. Finally, click the "New Feed" button and your feed will be generated and placed into the feed editor, ready for editing. Once you are satisfied with your edits, type a name for your new feed into the "Feed Name" text box and click the Save button.
Database Tools

Beginning with version 2, XMod Pro provides you with a Table Designer that allows you to quickly define and create basic tables in your DNN database. Additionally, you can optionally instruct XMod Pro to auto-generate a form and template for the newly created table - giving you the ability to have a complete base solution up and running with just a few clicks. The Database Tools area is only available to Host and SuperUser accounts. You can access the Database Tools by selecting "Control Panel" from the Actions menu:

On the Control Panel's toolbar, click "Database Tools":

The Database Tools screen is displayed. This screen is designed to allow you to quickly create basic tables in your DNN database. It is not designed as a replacement for dedicated database tools such as SQL Server Management Studio. Rather, it is useful for prototyping or creating basic tables without having to use an external tool.
The table designer is a great way to create basic tables in your DNN database or to quickly prototype a more advanced solution. With the table designer, you can create the table and automatically generate a basic form and template, giving you a complete, working model just a few mouse clicks. It is not designed as a replacement for Management Studio or other dedicated tools.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Size</th>
<th>Nulls</th>
<th>Identity</th>
<th>Default Value</th>
</tr>
</thead>
</table>

To design your table, click the New Column button just beneath the Table Designer grid. This will add a new column to your table and give you the opportunity to modify its characteristics.

Table Name: 

Create Table
First, type a name for your column in the Column Name field.

Next, select the data type for the column from the drop down list of options in the Data Type field.

If you have selected a text-type column that allows a size, the Size field will be available for you to specify the maximum number of characters allowed.

Other options: You can check the Nulls checkbox to allow null values in the column, tick the Identity checkbox to have the database automatically generate a unique ID for the column - starting at 1 and incrementing by 1. Finally, you can specify a Default Value for your column too.

NOTE: depending on the options you choose, other options may not be available. For instance, if you choose to make a column an Identity column, you cannot make the column nullable.

In the screen shot below, we've created a sample Contacts table:
Once you've designed your table, give it a name (Contacts in our example):

Table Name: Contacts

Once you're done, click the Create Table button and your table will be created.
Editing Your Forms and Templates with the Inline Editor

Beginning in version 1.4, XMod Pro includes a quick and easy way to edit the templates and forms configured for a given module. It is available only to Host or SuperUser accounts and is accessed via the new Host Options Panel that appears to Hosts and SuperUsers when working with XMod Pro modules.

**Books**

- **Title:** Book 4a
- **by Charles Dickens**
- **Synopsis:** test book 4
- **Author Id:** 6

View Details Edit Delete

If your modules hasn’t yet been configured with a form or template, the Host Options Panel will not appear.

**IMPORTANT:** The Host Options Panel and the Inline Editor REQUIRE jQuery 1.3.2 or later to be registered in the page.

For DNN 5 and later, jQuery is usually automatically included. To ensure that you are using version 1.3.2 or later, go to the Host Settings/jQuery section. From there you can specify the URL to use for jQuery.

For DNN 4, jQuery is not included by default. Because of this, XMod Pro cannot provide the Host Panel to every module instance. For each module you want to use the panel on, you will need to go to the Configure page for the module (make sure you’re logged in as Host/SuperUser) and check the box to enable the panel. Also, specify a URL to use for the jQuery 1.3.2 library. You should only specify the URL in one module per page because that will include the library for every module on the page.

Once you have successfully included the jQuery library and configured your module to use a form and/or template, you can click on the form/template name and the Inline Editor will be displayed, giving you one-click access to edit the current template/form (note: if you are using Internet Explorer, you editor may look slightly different).
Beyond giving you one-click access for editing, the included editor also makes it easier to add tokens, controls, and tags to your form/template through easy-to-use lists available from the toolbar (numbers 2-4 below).
1. **Tabstrip:** When editing forms, you can choose whether to edit the Add form - the form used when creating new records - or the Edit form - used when updating records - simply by clicking its tab.

2. **Tokens:** Click this button to access the list of XMod Pro tokens such as Field tokens ([[FieldName]]), Portal tokens (e.g. [[Portal:ID]]), User tokens ([[User:DisplayName]]) and others.

3. **Form Controls/Template Tags:** For Forms, this button will display a list of form controls you can choose to use when working with your form. For Templates, this will list the tags that are available for use in your template.

4. **Validation Controls:** For Forms Only: this button enables you to choose from the many validation controls XMod Pro provides (required field validation, CAPTCHA, validation summary, etc.)

5. **Resize Handle:** The editor even provides you with a grab-handle. Click and drag this handle to increase/decrease the height of the editor.

6. **Reload Page:** If you want your changes to be visible immediately, tick this box. This will instruct XMod Pro to save your form/template changes and then reload the DNN page you were viewing. If this box is not ticked, the changes will still be saved and any page that uses the form/template will see those changes when they're reloaded.
Using Javascript

This topic won't teach you how to use Javascript. Instead, it provides techniques for incorporating your scripts into your XMod Pro solutions.

Injecting Script Into the Page

For plain Javascript, you can simply type it into your form or template and it will be rendered out to the page at run-time. If you need to register a Javascript libraries in other files or inject a function or several functions, use a Script Block tag. Use the <mod:ScriptBlock> tag in templates and the <script> tag in forms.

Using Script Inside XMod Pro Tag Attributes

Often, you'll need to react to an event initiated by an XMod Pro control. The most obvious example is the Click event of buttons, links, and clickable images. For this, you should put your Javascript in the OnClientClick attribute of the control. Here's an example. Note that this is not a complete example of the tag since we're only focusing on the use of Javascript

```html
<mod:DeleteButton ... OnClientClick="return confirm('Delete This Record?');" ...>
```

When the button is clicked, the built-in Javascript function "confirm" is called. The confirmation function pops up a dialog which displays a message ("Delete This Record"), and prompts the user to click "OK" or "Cancel". If the user clicks "OK", the function returns "true". If the "Cancel" button is pressed, a value of "false" is returned.

It's important to return a value of true or false in cases like this. If false is returned, the button will stop its processing. In other words, it won't post back to the server and won't delete the record. If true is return, then processing continues normally.

Doing More with [[Join()]]

Let's say you want to pass one of your field values to a Javascript function. How would you do it? For this, you'll need to use XMod Pro's Join function. The Join function allows you to merge text with field values and other token values. It can be used in plain text, but it's primary purpose is for use within tag attributes. See the Function Tokens help topic for the Join function's syntax:

```html
<mod:DeleteButton ... OnClientClick='[[Join("myFunction(""{0}"");", [[CompanyName]])]]' ...>
```

There are a number of things happening in the above snippet:

1. Since we're using a token, we have to delimit the OnClientClick attribute with single quotes (') instead of double-quotes (")
2. Javascript is just text until it's interpreted by the browser, so we pass the Javascript call in as the Input to the Join function (the first parameter). Within the Javascript text, we use numbered placeholders (starting with zero) which correspond with the additional parameters we pass to the function. In this case, we just have one parameter, but we could have two or three or more. At run-time, the {0} will be replaced by the value of the CompanyName column.
3. For the sake of this example, let's say the myFunction function expects a text value, so we have to delimit it with quotes. However, since we're delimiting the OnClientClick attribute with single quotes, we have to use double quotes. The problem, though, is that we're also using the double quotes to delimit the Join function. What to do? We have to escape the double quotes by typing two consecutive double quote characters for each individual double quote character we need.

Getting Form Control Client ID's

If you're using Javascript in your forms, you'll more than likely need to access the controls on the form from within your script. You'll quickly discover, though, that the ID value you use for your control in XMod Pro, doesn't match the ID that is rendered in the final HTML source code. ASP.NET works to ensure that ID's in the final page are unique within the page. So, it takes your ID and generates a long, but unique ID to use in the browser.

While it's nice ASP.NET helps keep your code HTML compliant, all that re-naming ends up making it very difficult to write client-side script that uses those ID's. Well, XMod Pro has your back. The AddForm and EditForm tags each have an attribute called ClientName. Create a unique value and place it in this attribute. Then you can access your form controls with simple dot-notation:

```html
<AddForm ClientName="MyUniqueId">
...
<TextBox ID="txtFirstName" />
```
<a href="#" onclick="alert(document.getElementById(MyUniqueId.txtFirstName);return false;">Click Me</a>

Ah, but wait... What if you are using a couple of FormView modules, each of which uses the SAME form. Your ClientName will no longer be unique in the page. It's not a likely scenario, but you can still pull it off. Use the [[Module:ID]] token and the [[Join()]] function to ensure uniqueness in the page:

```html
<AddForm ClientName='[[Join("MyUniqueId{0}", [[Module:ID]])]'>
```

...
Configuring Your XMod Pro Module

The XMod Pro module configuration page allows you to assign forms and templates and set some security options for the module. The Configuration page is available to portal administrators as well as Hosts and SuperUsers. You can view the Configuration page by selecting “Configure” from the Actions Menu for the module, as seen below.

Most of the settings that define how your XMod Pro application operates are defined within the forms and templates. However, the Configure page is where you determine what forms and templates you’ll use and, if necessary, how they will operate. It contains four tabs:

Settings Tab

On the Settings Tab, you select which template and which form this module will use. You can choose to select just a template and no form, just a form and no template, or both. If you select just a form, then that form will appear when the module is loaded. If you're familiar with XMod, you can think of this as the FormView module. The net effect is that the user, when he/she navigates to the page, will see the form without having to click any links or buttons to do so. This mode is perfect for "Contact Us" or "Feedback" kinds of forms.
New to version 2.7, you can also specify your own **Custom Settings** for each module instance. This essentially allows you to create your own tokens that can be used in the form and template the module is using. To add a setting, click the "+" button. To delete a setting, select it in the list box and click the "X" button. To edit a setting, select it in the list box. For each setting you'll provide a Name and a Value. To use the setting, you'll create a Module token with that Name like so: `[[Module:settingName]]`. At run time, the token will be replaced by the Value you specify. So, in the example shown in the screen shot, if we placed `[[Module:DepartmentTitle]]` in our template, it would be replaced by *Marketing* at run-time.

**DNN Search Integration Tab**

New to version 4.0: XMod Pro now allows you to get your data indexed by DNN's search engine, making it possible to include XMod Pro managed data in site-wide searches.

As with all things XMod, we made search integration as flexible as possible to better accommodate the different scenarios in which it might be used.
Data Command: Just like you would enter SQL for a ListDataSource in a template tag, do the same here to retrieve all the records that should be indexed through this module instance. You can use certain tokens like [[Portal:ID]] and [[Module:ID]] but not user or session specific tokens like [[User:ID]], [[Url:paramName]], or [[Form:paramName]] since this will be executed by DNN without a web page being loaded (and thus no user session). **NOTE:** Simply embed your tokens in the command and they’ll be replaced at run-time. This is different than you would do for data commands in your templates. Field tokens are not allowed in the Data Command.

- **Title:** The title for the record that will appear on the DNN search results page. Titles can be a maximum length of 200 characters. As shown in the example, you can use field tokens combined with plain text for your title. Note that the maximum length is the rendered length - after any field tokens have been processed.
• **Description**: A description of the record that will be displayed in the DNN search results page. Descriptions can be a maximum length of 2000 characters. You can use field tokens and plain text in the description. Note that the maximum length is the rendered length - after any field tokens have been processed.

• **Author ID**: User ID of record author (optional). This should evaluate to the DNN User ID of the record's author. Leave it blank if there is no author. As shown in the example, you can use field tokens for your Author ID.

• **Content**: The actual content that will be indexed by DNN. As shown in the example, you can use field tokens combined with plain text for your title.

• **GUID**: Use this field to add URL parameters to the link that DNN creates that points to the detail view of the record. If your template is displaying a list of books, you might enter: bookid=[[BookId]]. You can chain multiple parameters together just as you would in a standard URL: bookid=[[BookId]]&authorid=[[AuthorId]]. Please ensure that your selected template has a DetailTemplate tag and that the DetailDataSource has Parameter tags set to same names as the URL parameters you pass-in. As shown in the example, you can use field tokens combined with plain text for your GUID.

• **Last Modified Date**: This should evaluate to a valid date and time value. This value should be the date the record was last updated. DNN uses this value to determine if the record should be re-indexed. If you use a date that doesn't change when the record is updated, the record will not be re-indexed. As shown in the example, you can use a field token for your Last Modified Date. **This value is required.**

• **Search Key**: A text value that uniquely identifies this record. Typically this may be the unique numeric ID that is assigned to a record. Search keys can be a maximum of 100 characters. Note that the maximum length is the rendered length - after any field tokens have been processed.

**Security Tab**

On the **Security Tab** you select which DNN roles have permission to add records through the module. Edit, Delete, and Detail View permissions are set within the template. Each user within the selected role(s) will be able to add records. You can select multiple roles by Shift-Clicking to select adjacent roles or Ctrl-Clicking (or Command-Clicking on the Mac) to select non-adjacent roles.
About Tab

On the About Tab you can see the "DLL Version" of XMod Pro (4.0.0.0 in the image above). This may be different than what is reported by DNN in the Module Definitions page. When reporting issues, we may ask you for this version as it will help us identify which release you're using. The value in parentheses is the DNN version that this version of XMod Pro should be running in. If you are running DNN 4.6 and the About tab reports DNN 5.0.1 and later, you are running the wrong version of XMod Pro. You should instead install the version for DNN 4.x.
Localization

XMod Pro provides a number of unique ways to localize not only the static text in your forms and views, but also your content.

Getting the Current Locale ID

(New to version 2.1) To render the locale ID of the currently selected culture out to your form or view (template), use the [Request:Locale] token. If the user has selected Spanish from Mexico, the token will render "es-MX"; For English in Great Britain, it will be "en-GB"; for French spoken in France, it will be "fr-FR".

Formatting Dates and Values in Templates

Use the <xmod:format> tag's InputCulture and OutputCulture attributes for handling currency and dates. See the linked topic for more details.

Localized Date Input in Forms

Use the <DatInput> control's Culture attribute. See the linked topic for more details.

Localizing Static Text

(New to version 2.1) If you just need to change the text that is displayed on your forms and views such as form labels or other text, you can use the [Localize:keyName] token in combination with resource files (.resx). The Localize token function like the other XMod Pro tokens. At run-time, it will be replaced with the translated text for the currently selected culture.

How it Works:

First, you need to create a Resource File for each culture for which you have translations. Resource files follow the standard format for localization resource files in DotNetNuke. Files should be named using the following form:

- FormName.ascx.localeID.resx for forms or
- TemplateName.ascx.localeID.resx for templates

Some examples:

- ContactUs.ascx.es-MX.resx (Spanish-Mexico resource file)
- EmployeeList.ascx.fr-FR.resx (French-France resource file)

The resource file's base name must match the name of the form/template to which it is associated followed by ".ascx". This is then followed by "." plus the locale ID and ".resx".

Resource File Example:

The format of the resource file should follow that outlined for standard localization resource files in DotNetNuke. A sample file is provided below. The first section and the closing <root> tag at the end, in gray, are boilerplate code and can be copied and pasted directly into your file. The section in red is the area you customize with your own text.

```xml
<?xml version="1.0" encoding="utf-8"?>
<root>
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
```
In the example above, we've created a small translation file for Spanish/Mexico. You can create one or more resource files for each language you need to provide translations for. Note that you should also provide a default resource file for your fallback language. Additionally, for the default/fallback language to be loaded properly, you should define a fallback language for each language you install in your DNN website.

Within each file, for each block of text you want to translate, you must specify a <data> tag. This tag has a name property which contains the key name that XMod Pro will use to look up the translation. This name should be in the form "keyName.Text". The .Text portion is required for each of your names. The <data> tag also contains a <value> tag that contains the text you want to inject into the form or template.

**Where to Place the Resource File**

Once you've created your resource file, save it in the same directory in which your form/template is stored. For forms, this is: /Portals/_default/XModPro/Forms/XX where "XX" is the Portal ID. For templates, the folder is: /Portals/_default/XModPro/Templates/XX where "XX" is the portal's ID.

**Linking Your Form/Template to the Resource File**

In your form/template, you would simply add a localization token wherever it was needed - like so:

```xml
<AddForm>
  ...
  <Label for="NameTextBox" text='[[Localize:Name]]'/>
  <TextBox id="NameTextBox" DataField="Name" DataType="String" /><br/>
  <Label for="AddressTextBox" text='[[Localize:Address]]'/>
  <TextBox id="AddressTextBox" DataField="Address" DataType="String" /><br/>
  <Label for="CityTextBox" text='[[Localize:City]]'/>
  <TextBox id="CityTextBox" DataField="City" DataType="String" /><br/>
  ...
</AddForm>
```

At run-time, if the user has selected es-MX as his/her culture, then those tokens will be replaced by: Nombre, Dirección, Ciudad.

**Content Localization Using Multiple Forms/Templates**

Content localization is a tricky issue. It is potentially more complicated when an application like XMod Pro, does not have control over the data. To keep things as simple and flexible as possible, we have implemented the ability to use multiple forms and/or templates for this purpose. At run-time, XMod Pro will look to see if a form/template has been defined for the currently selected culture. If found, that file will be loaded. If not, XMod Pro will load the 'default' file.

This method has many benefits:

1. You can avoid having to create Resource Files and use [[Localize:KeyName]] tokens.
2. You can create the forms/templates just as you do standard forms and templates.
3. You have the opportunity to define distinct data commands for each culture. You could even write/read from distinct culture-specific database tables.
4. You can have a completely different layout and even completely different controls for each culture.
5. You can send notification emails to different addresses based on the culture.

**How It Works**

First, define the form/template that will be your 'default' or 'fallback' file using the tools within XMod Pro.

Second, either create another form/template from scratch or simply copy your form/template to act as a starting point. This second file must follow the following naming convention:

```text
BaseName.localeID
```

Where "BaseName" is the name of your 'default' form/template and "localeID" is the ID of the culture (es-MX, fr-FR, en-GB, etc.)
So, for a form called "ContactUs", you might create the following forms:

- ContactUs (the default form)
- ContactUs.es-MX (Spanish in Mexico)
- ContactUs.fr-FR (French in France)
- ContactUs.en-GB (English in Great Britain)

The default file is always required. You can create as many or as few localized files as you need. Simply define your localized forms/templates as you would any other form/template, making changes as needed to accommodate the language.

**Configuring Your Module to Use Localized Forms/Templates**

There is nothing special that you need to do when configuring your XMod Pro modules to use the localized forms and templates. Simply select the BASE form/template. If a localized version of the form/template exists and it matches the user's currently selected culture, that form/template will be loaded automatically. You can also force a certain localized file to be loaded by selecting it specifically on the configuration page.

That's all there is to it.
Walkthrough: Templates With Multiple Views

In most cases, all you'll need to do in XMod Pro is setup solutions that add, edit, delete, and display data, but XMod Pro is much more powerful than that. In this walkthrough, we'll discuss how to create multiple views within the same template. By leveraging this capability, you can create very powerful self-contained solutions.

In this walkthrough we won't be creating all the components of a sample application. In this example, we want to focus on the mechanisms for creating multi-view templates without bogging down in the details of creating templates and forms as well as adding/editing/deleting the data. Those concepts have been covered elsewhere and we assume you're comfortable with them.

Previously, you've created a template using a single <xmod:template> tag. This creates a single view that can comprise a list view of your data and a detail view that displays the information contained within a single record. When the detail view is displayed, it replaces the list view. That's all well and good for most situations but what if you want to display a list of news headlines on the left side of your module and, when you click a headline, the full article is displayed on the right side of your module? With most modules, you're out of luck, but not with XMod Pro. All it takes is a second <xmod:template> tag and a new button type.

Master View

"Wait," you may be saying, "How can a template contain more than one template?" To keep things straight, let's introduce the concept of a "master view". When you create a template via the Manage Templates page, you're actually creating a master view. This master view typically contains only one template tag, but it can hold multiple template tags - enabling it to present the user with multiple views.

Each template tag is self-contained. It knows only how to deal with its own data. However, XMod Pro is designed so that one template can communicate with other templates in the same master view. Template A can tell Template B to display its list of records and Template C to display its detail view. More importantly, Template A can pass parameters to Template B and Template C that determines what data they're going to display.

So, for our news article example, we'd have two templates: Template A would display a list of headlines and Template B would display the full article. We'd create a simple two-column table to establish the layout - the Headline template goes in the left column and the Article template goes in the right column. The basic idea of how this would look is laid out below:

<table>
<thead>
<tr>
<th>Headlines (Template A)</th>
<th>Article (Template B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article One by John Smith</td>
<td>Article One by John Smith</td>
</tr>
<tr>
<td>Article Two by Adam Smith</td>
<td></td>
</tr>
<tr>
<td>Article Three by Joanna Smith</td>
<td></td>
</tr>
</tbody>
</table>

This is just a simple HTML table in our master view - one row with two columns. How you choose to lay things out is completely up to you. Put an <xmod:template> tag in the left column and another <xmod:template> tag in the right column. The Headlines template would only contain...
the information necessary to display a list of news headlines, while the Article template would only contain the information needed to display the full article.

Beyond that, all we need is a way for the Headlines template to tell the Article template which article to display. This is handled by a new button type - called a command. As with most buttons in XMod Pro, the command button can be a push-button (<xmod:commandbutton>), a hyperlink (<xmod:commandlink>), or a clickable image (<xmod:commandimage>). In the mock-up above, the article titles are <xmod:commandlink> tags.

Here's what the command link code looks like:

```xml
<mod:CommandLink text="[[Headline]]">
  <Command target="Article" type="Detail">
    <Parameter name="ArticleId" value="[[ArticleId]]"/>
  </Command>
</mod:CommandLink>
```

The Command Link control uses the article's Headline column for its display text. More importantly, it contains a Command tag. In this tag we've specified the target - i.e. the ID of the template we want to command. In this case, it's the Article template. Next, the "type" attribute allows us to specify which command we want to execute in that target template - the Detail data source in this case. Finally, we need to pass the identifier for the article to the detail template so it can retrieve the correct record. This is done with the <Parameter> tag. The parameter has a name of "ArticleId" and the parameter's value is supplied by the current record's ArticleId column. Important, the name of the parameter must match the parameter name in the <DetailDataSource> tag in the Article template.

The important components of the template are listed below. The HTML used for layout has been removed to simplify the code.

```xml
<mod:template id="Headlines">
  <ListDataSource CommandText="SELECT ArticleId,Headline FROM Articles"/>
  <ItemTemplate>
    <mod:CommandLink text="[[Headline]]">
      <Command target="Article" type="Detail">
        <Parameter name="ArticleId" value="[[ArticleId]]"/>
      </Command>
    </mod:CommandLink>
    by [[Author]]<br />
    ...</ItemTemplate>
  </mod:template>

<mod:template id="Article">
  <DetailDataSource CommandText="SELECT * FROM Articles WHERE Id=@@ArticleId">
    <Parameter name="ArticleId" value="-1"/>
  </DetailDataSource>
  ...<DetailTemplate>
    <strong>[[Headline]]</strong><br />
    by [[Author]]<br />
    <div>[[Article]]</div>
  </DetailTemplate>
</mod:template>
```

**Notes:**

- The Command Link tag contains a Command tag that specifies "Article" as its target. The value of the target attribute must be the ID of the template you want to control.
- The Command tag must specify if it wants to execute the Detail data source or List data source in that target template. In our case, we want to execute the Detail data source, so we supply "Detail" to the 'type' attribute.
- The Command tag's Parameter child tag, if supplied must have its name match the name of the parameter in the target template's data source (the Detail data source in our example).
- When the Command Link is clicked, XMod Pro will look for a template with the ID of "Article". If found, it will then search for the "Detail" data source, passing the value of ArticleId to it. The target template then takes over and retrieves its data and displays the detail template.
- Notice the Parameter tag on the DetailDataSource. It's value is set to -1. This is a dummy value. In our table all Article ID's are positive numbers, so -1 will ensure that no records are returned when the page is first loaded.

The full template is below:

```xml
<table>
  <tr>
    <td>
      <xmod:template id="Headlines">
        <ListDataSource CommandText="SELECT ArticleId,Headline FROM Articles"/>
        <ItemTemplate>
          <xmod:CommandLink text="[[Headline]]">
            <Command target="Article" type="Detail">
              <Parameter name="ArticleId" value="[[ArticleId]]'/
            </Command>
          </xmod:CommandLink>
        </ItemTemplate>
        by [[Author]]<br />
      </xmod:template>
    </td>
  </tr>
  <tr>
    <td>
      <xmod:template id="Article">
        <DetailDataSource CommandText="SELECT * FROM Articles WHERE Id=@ArticleId">
          <Parameter name="ArticleId" value="-1"/>
        </DetailDataSource>
        <DetailTemplate>
          <strong>[[Headline]]</strong><br />
          by [[Author]]<br />
          <div>[[Article]]</div>
        </DetailTemplate>
      </xmod:template>
    </td>
  </tr>
</table>
```
FAQ

General

- **Can users other than Hosts or SuperUser accounts add, edit, and delete forms and templates?**
  No. XMod Pro enables you full access to your database. This provides you with enormous power. However, it also means it’s possible to do quite a bit of damage to your data. It is possible to not only wipe out the data in your portal’s database, but the data in every other portal. Because of this, we decided to only allow access to Hosts. However, Administrators of individual portals can configure their modules, selecting forms and templates you’ve created. As a Host, once you create a form/template while logged into a given portal, that form/template then becomes available to the Admin of that portal for use.

Forms

- **How do I link controls in my form with fields in my data?**
  All controls designed to work with XMod Pro supply you with two attributes: DataField and DataType. Put the name of the field you want to link to the control in the DataField attribute. Next, specify what the data type of that field is. If it is text, use "string" at the datatype. If it is true/false, use "boolean", if it is a number, use "int32" or "int64", etc.

- **Can I use space and/or punctuation marks when naming my forms?**
  Only letters, numbers, hyphens (-), and underscores (_) are allowed. To help distinguish different words in the name, you can separate them with hyphens and/or underscores or you can use "camel case" where you capitalize the first letter in each word.

Templates

- **How Do I Tell XMod Pro Where to Display My Data?**
  Within your templates, use Field Tokens to identify where to place your data.

- **Can I use space and/or punctuation marks when naming my templates?**
  Only letters, numbers, hyphens (-), and underscores (_) are allowed. To help distinguish different words in the name, you can separate them with hyphens and/or underscores or you can use "camel case" where you capitalize the first letter in each word. For example: MyForm
Revision History

4.2.0

- ADDED: CommandButton now works in NoItemsTemplate. You can use non-Field Tokens. (change made in Template.vb)
- ADDED: Add Redirect Property to Command buttons (<xmod:CommandButton/Image/Link>). No POST values can be passed, but you can use ",. " as the Redirect url and can do the redirect via GET or POST.
- ADDED: [[DateAdd]] token now allows output of time component (HH:mm:ss).
- ADDED: <xmod:IfNotEmpty> and <xmod:IfNotEmpty> template tags which will render their contents if the Value property is empty/not there or not empty/exists, respectively.
- ADDED: Ability to override the default FormView mode (i.e. show AddForm or EditForm) via a URL parameter: "xmfvo". When this is set to "1", the module will display the AddForm. When it is "2" the EditForm will be displayed.
- ADDED: Added DateAdd token and more User tokens to text editor toolbars.
- ADDED: Tool tip property to <xmod:ToggleLink>
- ADDED: Placeholder property to <TextBox> and <TextArea> This allows you to provide the user with informational text inside the control when it is empty. When the control receives focus, the placeholder text will disappear. When the control loses focus, if the control's value is still empty, the placeholder text will re-appear. The placeholder text is a usability component that exists on the user interface only. It does not get sent to the database if the control is empty when the form is submitted. NOTE: This will only work on HTML5-capable browsers that support this feature. All other browsers will simply ignore it.

4.1.2

- FIXED: <DualList> form control - when removing items from the 2nd list during record edit, an EventValidation error would be thrown in DNN 5.6x and up.
- FIXED: Checkbox list tag designer doesn't populate list items correctly.
- FIXED: Manage Feeds - feed builder doesn't generate code.
- FIXED: Unable to delete a feed in Manage Feeds page.
- FIXED: Template Editor - <SearchSort> tag designer does not include DisplayTemplate if one has been defined.
- FIXED: Additional form themes were not installed.
- FIXED: Template Editor - <xmod:jQueryReady> tag designer didn't function correctly.
- FIXED: Manage Templates - Unable to set name for new template after editing a template.
- FIXED: Inline Editor - the "format all" button didn't work. It did work in the Control Panel editors.

4.1.1

- FIXED: [[DateAdd]] token threw an exception when used in templates.
- FIXED: Depended List with 2 or more dependent lists would throw an exception.
- FIXED: FormBuilder's RadioButtonList throws 'object undefined' error when adding it to the form.

4.1.0

- ADDED: "If" property to <AddToRoles> tag. This allows for conditionally adding roles to users.
- ADDED: <AddUser> tag now passes the newly created user downstream as "__UserId". So, you can reference the new ID in <AddToRoles>, for instance and add roles (conditionally if needed) to the newly created user.
- ADDED: <AddToRoles> has 3 new properties: StartDate, EndDate, and Culture which allow you to specify a time frame the user should be in a certain role. Culture is used to specify that the dates are in a format other than the current system's culture.
- ADDED: Inline editor now displays the name of the form/template you're editing.
- ADDED: FormBuilder's ControlDataSource designer now allows you to choose a column to sort on and specify the sort order.
• ADDED: New DateAdd token. The token allows you to return a date relative to the current date. You can specify an interval such as day, week, month, or year and the number of that interval to add to the current date. A positive number will net you a date in the future. If you use a negative number, you’ll get a date in the past. You can also specify a format that will be used when generating the date.

• ADDED: Forms can now specify a ScrollToTop property. When this is true, when the user clicks the Add or Update button, the browser will scroll to the top of the page on page load rather than trying to scroll to the bottom of the form. This is handy if you have a long form or a lot of content that you replace with a SuccessTemplate. It can help prevent the view from being less jarring to users.

• FIXED: `<AddToRoles>` did not properly recognize the UserId property.

• FIXED: Issue where users could not convert forms from Auto-Layout to Custom HTML when working with Global forms.

• FIXED: Issue where `<Login>` tag's RememberMe property would not accept a field token as its value.

• FIXED: Issue where `<AddUser>` tag's Approved property wouldn’t accept a field token as its value.

• FIXED: Issue where `<AddUser>` tag's UpdatePasswordOnNextLogin property wouldn’t accept a field token as its value.

• FIXED: Issue where a `<DateInput>` tag that was set to DateOnly=True or had its DataType set to Date and did not specify a Format would display the time component if it was bound to a data value.

• FIXED: The `<Validate>` tag designer in the text editor generated validate tags with a missing "=" for the Message property.

4.0.3

• FIXED: Issue where `<AddUser>` didn’t save some properties on a new user when used on non-root portals.

4.0.2

• ADDED: New CheckboxList validator (`<Validate Type="CheckboxList" />`) which will validate that a checkbox list control has at least one checkbox ticked.

• ADDED: Handling for Global forms in FormView module.

• FIXED: `<UpdateUser>` tag wouldn’t update Password.

• MODIFIED: `<UpdateUser>` tag to remove the "Password" property. It has been replaced by the "NewPassword" property. Additionally, an "OldPassword" property has been added.

• FIXED: Improved end-user error reporting from the `<UpdateUser>` tag for bad passwords.

• FIXED: FormBuilder UI issues preventing tag designers for `<AddUser>`, `<Login>`, and `<RemoveFromRoles>` tags from appearing.

• FIXED: `<UpdateUser>` tag's designer output javascript when generating the tag.

• FIXED: `<AddUser>` tag's designer generated improperly closed tag.

• FIXED: Issue where the "." shortcut for redirects only worked in `<Redirect>` tags and not in Add/Update/Cancel buttons in form.
4.0.1

- FIXED: Installation issue with some missing files on DNN 5.4 and earlier.
- FIXED: Paging issue where if a record was deleted on the last page an empty resultset would be displayed rather than the previous page.
- FIXED: issue where Select tag designer would put Else child tag as a property of the <xmod:Select> tag rather than as a child tag.
- FIXED: Pager Tag Designer generates "on" instead of "true/false"
- FIXED: Email designer putting "SendRule" property in <Email> tag for form text editor
- FIXED: Compare Validator's DataTypeCheck operator doesn't function correctly.
- FIXED: Issue with <UpdateUser> tag's UserId property would not accept field tokens. It would throw a compilation error.

4.0.0

- ADDED: <AddToRoles> Action to forms. You can now add a user to one or more DNN security roles on successful form submission.
- ADDED: <RemoveFromRoles> Action to forms. You can now remove a user from one or more DNN security roles on form submission.
- ADDED: <AddUser> Action to forms. This action enables you to register a new user in the DNN site upon form submission and assign some roles to that user. It provides a basic set of properties such as FirstName and Address but also allows you to specify custom profile properties too.
- ADDED: <UpdateUser> Action to forms. This action enables you to provide users with the ability to edit their user information.
- ADDED: <Action> Custom Action tag to forms. Now you or a developer can build custom actions that get executed when a form is successfully submitted and any data saved to the database. The custom action is even able to modify or add form values that can be used by other Actions that execute later in the sequence, including <Email> and redirection. So, you can dynamically modify the content of an email and or change the destination for a redirection.
- ADDED: DNN Search Integration. Now you can specify a SQL command that gets executed when DNN indexes your XMod Pro module instance. You're able to integrate field tokens with plain text to populate the Author, Title, Synopsis, etc. that DNN indexes and displays to end users during a site search.
- ADDED: The <xmod:Redirect> tag can now be used in AddSuccessTemplate and EditSuccessTemplate tags. This allows you to use formvalues and values returned from stored procedures to create a link/button/clickable image that can be POST'ed to other pages.
- ADDED: Edit forms no longer require a SubmitCommand. This allows you to more easily create forms that can display, say, the details of a record and then send that record via Email or Redirect the user and data to a different target without having to put a 'dummy' SubmitCommand in the form.
- ADDED: Ability to use Stored Procedure Return values in Add/EditSuccessTemplate tags.
- ADDED: Ability to use stored procedure OUTPUT parameter values in Add/EditSuccessTemplate tags as well as in redirects.
• ADDED: New `<Validate Type="Database" />` tag to forms. Only one is recognized per form. Use it to display error messages returned from the database. These can be actual errors raised from the database or friendly error messages passed back from a stored procedure.

• ADDED: New `<Validate Type="Action" />` tag to forms. Only one is recognized per form. Use it to display error messages returned from special action tags like `<AddUser>`. This validator allows you to display more mundane errors like a "username already exists" error to the user so he/she can correct it.

• ADDED: In forms, if are using a stored procedure in your SubmitCommand and you set an OUTPUT parameter to have a name of ERROR and it is notnull or empty, XMod Pro will treat it as if an error was thrown. It will display the error to the end user using the `<Validate Type="Database" />` tag if available and will not continue processing. One use case: if you have a form that allows the user to create a new profile and profile name. You can do a check in your stored procedure to see if that profile name already exists and, if so, return a message to the user to pick another profile name. Even though no error was technically thrown, XMod Pro will treat it as such.

• ADDED: `[[Request:PageName]]` token which returns the name of the current page without the file extension.

• ADDED: Ability to specify a comma-delimited list of Target Control ID's to `<DropdownList>` form control. This allows one list to dynamically populate multiple list controls. Previously it could only control one other list control.

• ADDED: `<SilentPost>` action to form. This has one property: Url and optional `<Field>` child tags. Each field tag has a Name and Value property. SilentPost will send an HTTP POST to the Url and send the specified field names and values without redirecting the user.

• ADDED: `<Redirect>` Conditional Redirects to form. Using the one or more of the new `<Redirect>` tags and their `If` property, you can set up a list of possible targets to redirect the user to and based on values from form, determine where to direct the user after form submission.

• ADDED: `<AddToRoles>` , `<Redirect>` , `<Text>` , and `<Label>` tags to the Formbuilder UI.

• ADDED: Ability to choose which columns to include in a template when generating the template using the Template builder.

• ADDED: Form Builder UI: RadioButtonList and CheckboxList can now have their RepeatColumns, RepeatDirection, RepeatLayout, and SelectedItemsSeparator set in their respective tag designers.

• ADDED: ReplyTo property to `<Email>` tag.

• ADDED: ReplyTo, CC, and BCC properties to the FormBuilder UI for the `<Email>` tag

• ADDED: Ability for `<ControlDataSource>` to easily retrieve a list of users, pages, roles, or countries using the DNN API rather than from direct database calls. To use this, set the Source property of the tag to `\`DNN`\` and set the CommandText to: Users for a list of users in current portal, Roles for a list of roles in current portal, Pages for a list of pages in the current portal, and ListCountries for al list of countries contained in DNN's Lists table.

• ADDED: New `<xmod:AjaxButton>`, `<xmod:AjaxLink>`, and `<xmod:AjaxImage>` tags for templates. These will render as a button, hyperlink, or image respectively. When clicked, it will execute a jQuery.ajax call to the URL specified in the Url property and insert the HTML results into the DOM element specified in the Target property. Use any jQuery selector for the Target property. The tags also provide the ability to specify an image to display while waiting for a response (LoadingImageUrl) and that image can be styled using the LoadingImageCssClass property.

• ADDED: New `<AjaxButton>` , `<AjaxLink>` , and `<AjaxImage>` tags for forms. These will render as a button, hyperlink, or image respectively. When clicked, it will execute a jQuery.ajax call to the URL specified in the Url property and insert the HTML results into the DOM element specified in the Target property. Use any jQuery selector for the Target property. The
tags also provide the ability to specify an image to display while waiting for a response (LoadingImageUrl) and that image can be styled using the LoadingImageCssClass property.

- ADDED: New <xmod:jQueryReady> tag. This tag provides a convenient way to place your code inside a jQuery(document).ready() function and embed it in the page. Importantly, the script will be placed near the bottom of the page for better perceived load time and will also be placed inside a closure so that your script’s scope is protected and so that you can use the shorthand "$" instead of jQuery. By using this tag, you also do not have to add <script> tag or register your script in the page via a ScriptBlock tag - as that is handled for you behind the scenes.

- ADDED: New <jQueryReady> tag for forms. This tag provides a convenient way to place your code inside a jQuery(document).ready() function and embed it in the page. Importantly, the script will be placed near the bottom of the page for better perceived load time and will also be placed inside a closure so that your script’s scope is protected and so that you can use the shorthand "$" instead of jQuery. By using this tag, you also do not have to add <script> tag or register your script in the page via a ScriptBlock tag - as that is handled for you behind the scenes.

- ADDED: New <xmod:LoadFeed> tag. This makes it a cinch to pull an XMod Pro feed directly into your page. Simply specify the feed's name and the HTML element you want to fill with the results. Everything else is taken care of. Should you desire, you can specify a progress or "loading" image as well as send additional data to the feed.

- ADDED: New <xmod:LoadFeedButton>, <xmod:LoadFeedImage>, and <xmod:LoadFeedLink> template tags. These buttons simplify including the results of an XMod Pro feed directly into your page using AJAX. The feed is loaded when the user clicks the button/image/link. Additionally, these tags provide an ability to implement "Infinite Paging" in which clicking the button will return, say 10 results and add them to the existing list of results. If the button is clicked again, another set of 10 results will be added to the existing list. Like the <xmod:LoadFeed> tag, you have the option of specifying a "loading" image.

- ADDED: Inline Editor now expands to cover more screen when viewed on larger monitors.

- ADDED: Property to <Pager> in Templates called ScrollToTop. When True (the default setting), the page will scroll to the top when changing pages rather than the default DNN behavior of maintaining scroll position. This will provide users with a more natural interaction when viewing long lists. NOTE: This only occurs if the user clicked on the bottom pager since, presumably, clicking there means the user is at the bottom of a long list. If the top pager is clicked, the scroll position is maintained because the user would expect to maintain his/her position.

- ADDED: New ViewRoles property to <xmod:Feed> tag. This allows you to specify one or more comma-delimited DNN security roles. When specified, XMod Pro will check to see if the user is in one of the roles before rendering the feed. If the request is coming from outside a DNN session, the feed won't be rendered. Feeds will be rendered if the request is coming from a Host/SuperUser or the Administrator account.

- ADDED: <DropdownList> control’s TargetControlID and TargetDataControlID properties can now accept more than one value, separated by commas, enabling a drop-down list to update the data and values of multiple list controls.

- ADDED: Inline Editor now saves your results via AJAX without dismissing the dialog. This gives you the opportunity to save your work and continue editing. An example workflow is to have the editor displayed in one browser tab with the page also open in another tab. Then, you can save your work, switch tabs, preview your changes, switch back to the editor tab and continue editing as needed.

- ADDED: Email can now be sent if SSL is set in your DNN site settings.

- ADDED: You can make forms and templates accessible to all portals on your website with Global forms and templates.

- ADDED: New form Action tag called <Login>-. This tag enables you to create custom login forms for your DNN website.
• ADDED: Redirection in forms now accept "." a shortcut to the current page. You can specify this as the Target property (or the Redirect property on buttons).

• MODIFIED: For better consistency with other areas of XMod Pro, the AddRoles, EditRoles, DeleteRoles, and DetailRoles properties of the <xmod:Template> and <xmod:DataList> tags now accept comma-delimited roles. For backward-compatibility, semi-colon-delimited roles are still accepted but are officially deprecated.

3.1.2
• FIXED: Issue where error would be thrown in a DataList when clicking to view the detail of a record. This occurred only when UsePaging on the DataList was set to False. Thanks to Jack Dunn for finding this.

3.1.1
• FIXED: In Template tag and DataList tag, when Ajax was set to true, a parsing error would be thrown when trying to click a Redirect button in the NoItemsTemplate.

• FIXED: <TextArea> HtmlEncode property did not work correctly.

3.1.0
• ADDED: [[User:Photo]] token support for DNN 6. It returns the URL to the user's profile picture.

• ADDED: Date SQL Server date data type for those versions of SQL Server that support it

• ADDED: Support for TinyInt SQL data type (values of 0-128). When linking to a TinyInt column, use "Byte" as the DataType in your form's control.

• ADDED: Support for SmallInt SQL data type

• ADDED: Support for XML SQL data type for those versions of SQL Server that support it. Use the String XMod Pro control data type and consider adding the new XML Validate tag.

• ADDED: Support for UniqueIdentifier SQL data type. NOTE: You must validate your input is in the proper GUID format or you'll get a database error.

• ADDED: Support for DateTime2 SQL data type for those versions of SQL Server that support it. This is a larger data type that holds a greater range of dates. Use DateTime as the XMod Pro control DataType.

• ADDED: XML Validate Tag for use in validating basic well-formed XML.

• ADDED: "DateOnly" property to DateInput control. When True, the control will only process the date component of the entered value. If the data type in the database is DateTime, the time component will be 12:00:00 AM.

• ADDED: Ability to set a form control as Nullable in the FormBuilder. This allows you to specify that an empty value in the control will be passed to the database as a NULL and NULL values coming from the database will result in the control being empty.

• ADDED: DateInput form control now will be empty when first loading if no value is supplied.

• ADDED: FileUpload control now displays 'new file' button after an upload - per customer request.

• FIXED: Database Tools issue - when creating a new table, specifying a default value for a column wouldn't work.

• FIXED: Some issues when using the Redirect property on form buttons as well as an issue where a record could only be added once per user session.
- FIXED: Event Validation error when using DualList form control, spawned by change in DNN 5.6.3 and later which enabled event validation.

- FIXED: Event Validation error when using Custom Module Settings feature available on the module configuration page, spawned by change in DNN 5.6.3 and later which enabled event validation.

- FIXED: <Variable> tag values now work with <SelectCommand>'s in forms.

3.0.7

- FIXED: Implement a fix for when a record is added via the New Record button in a template and the user then reloads the page. The result is that the form is submitted again, resulting in a duplicate record.

- FIXED: When external table is selected in wizards and builders, columns do not show up for selection in some cases.

- FIXED: Email form tag. If a FileUpload control is used for the Email's Attachment property but no file has been uploaded, an exception is thrown.

- FIXED: Cookie Token - If the cookie name contained periods or dashes, the cookie token would not function.

3.0.6

- FIXED: Issue in DualList form control where SelectedItemsSeparator property did not work

- FIXED: Issue in DualList which prevented values from being saved correctly to the database.

- FIXED: Issue in DualList where if the control was inside a tab, bound to a ControlDataSource and being used on the Form-View when in EditMode, the control would not pre-select the record's values.

- FIXED: Issue in DualList where AppendDataBoundItems was not being respected.

- FIXED: Issue in DualList: Data returned in right-hand list (list 2) comes back in descending order, not the order in which the values appeared originally in the list box.

- FIXED: Issue in DualList: When editing a record, if the user does not interact with the control (i.e. select a new item), then the values are not saved.

3.0.5

- FIXED: Extra properties on RegEx, Compare, and Range Validators not persisted by FormBuilder.

- FIXED: Minor display issue in Validate tags designer tab. Saved validators now have proper captions.

- FIXED: Adjusted width of validator list to accommodate width of "Regular Expression".

- FIXED: Issue where Tokens in Add/Update/Cancel button/image/link tag properties like Text wouldn't get processed.

- FIXED: Issue where a TRIAL license, after expiration, could not be activated as a normal license because the fields to enter account and invoice were hidden.

3.0.4

- FIXED: Issue where changes made to a copied form were reflected in the original form instead.
3.0.3

- FIXED: Unable to rename forms in Manage Forms page.
- FIXED: Renaming an Auto-Layout form may make it un-editable.

3.0.2

- FIXED: IE8-only issue where form builder control designers wouldn’t appear.
- FIXED: IE8-only layout issue in form builder control designers.
- FIXED: Issue in License/Activation page which may prevent activation on some systems.

3.0.1

- FIXED: Auto-Layout conversion link would not appear in the Form Manager Grid on some systems.
- FIXED: Issue where Control Panel toolbar would appear very narrow in some skins.

3.0.0

- ADDED: New Form Builder as part of the Manage Forms page. The Form Builder allows you to easily design and edit attractive data entry forms (called Auto Layout forms within XMod Pro). The new form builder can also work with external SQL server tables outside the DNN database.
- ADDED: Ability to generate feeds and other types of templated export of your data. Now you can generate XML, comma-delimited, or other types of output in the same way you create list views (i.e. templates).
- ADDED: Custom Data Commands to templates. Now you can trigger other types of data commands in your templates like approving a record, triggered from a button, link, or clickable image.
- ADDED: Ability to convert Auto-Layout forms (form builder forms) to custom HTML forms with a single click, while preserving the original form.
- ADDED: Date Modified and Date Created columns to the Manage Forms, Manage Templates, and Manage Feeds grids.
- ADDED: Ability to sort grids in the Manage Forms, Manage Templates, and Manage Feeds grids by clicking column headers.
- ADDED: Ability to use external SQL Server tables and views in the form, template, and feed editors.
- ADDED: AppendDataBoundItems property to the DualList form control.
- ADDED: New Cookie token to use the value of cookies in your forms and templates.
- ADDED: Theme CSS stylesheets no longer have to be named with the 1.7.2 jQuery UI version number. Instead, just ensure there is only one CSS sheet in the theme’s directory.
- ADDED: In-app video tutorials (requires internet access).
- REMOVED: Final vestige of DevExpress controls within the application. XMod Pro should no longer present DevExpress versioning issues. Old DevExpress DLL’s that were only used by XMod Pro may be removed. Of course, if other modules rely on them, they shouldn’t be removed.
- FIXED: Issue where Inline Editor window would pop-up but the animated "loading" graphic would never be replaced by the actual editor. This issue only appeared when both a standard XMod Pro module and an XMod Pro FormView module were on the same page.
2.7.7
- FIXED: Validation wouldn’t work in <Panel>
- FIXED: <xmod:Redirect> wouldn't work when template/datalist was set to use Ajax.

2.7.6
- FIXED: <xmod:Redirect> tag's POST method wasn't functioning correctly after 2.7.3 changes.

2.7.5
- FIXED: <Variable> values set by tokens didn't work.
- FIXED: <xmod:Format> tag's MaxLength property didn't function correctly.

2.7.4
- FIXED: <AddButton> in <AddForm> of main XMP module would not POST form values. An <AddButton> in a FormView module would.
- FIXED: <UpdateButton> in <EditForm> of main XMP module would not POST form values. An <UpdateButton> in a FormView module would.
- FIXED: <CancelButton> in <AddForm> of main XMP module would not POST form values. A <CancelButton> in a FormView module would.

2.7.3
- FIXED: Enabled redirecting to work for CheckBoxList - so that it passes its item values instead of it previously sending a series of "on" values. This required a significant change to the way redirects are handled on the backend and may impact existing users.
- FIXED: Pager did not respect setting for ShowFirstLast and ShowPrevNext.
- FIXED: FormView does not process user-defined Module Settings.
- FIXED: Issue where DetailDataSource was using the same CommandType as the ListDataSource. This may have manifested as an error: "Could not find stored procedure ",." if your ListDataSource was a stored procedure but your DetailDataSource was not.
- FIXED: Added fix for ToggleLink/Button/Image - When Target is calculated (i.e. it uses tokens), the value is lost when the page is posted back from a different module (i.e. paging through a second XMP module on the page).

2.7.2
- FIXED: FileUpload form control - 'object reference not set' error when DisplayMode was set to FilePicker and a file was uploaded.
- FIXED: FileUpload form control - Drop-down file picker list did not refresh to add the newly uploaded file to the list.
- FIXED: DNN manifest should now correctly show a version of 02.07.02 for this version.
- FIXED: Form Editor - AddImage/CancelButton/UpdateImage helpers misspelled "AlternateText" property name.
- FIXED: Template Editor - ListDataSource/DetailDataSource/DeleteCommand helpers misspelled "Parameter"
- FIXED: Template Editor - AddImage/ReturnImage/ToggleImage helpers misspelled "AlternateText" property name.

2.7.1
- FIXED: [[Localize]] tag not functioning
- FIXED: Compare validator not recognizing the CompareValue property.
- FIXED: Unable to preview a template in the Manage Templates page in some cases. User receives an 'object reference not set' error.
- FIXED: No visual feedback in template that a record has been deleted.

2.7.0
- ADDED: New <Variable> tag to forms. This allows you to specify a value to be used elsewhere in your form for configuration purposes or as additional data. The value can be bound to controls on your form and used in your data commands. Values of variables can be hard-coded or XMod Pro tokens like [[Portal:ID]]. See the AddForm/EditForm topic for details.
ADDED: Module-level, user-defined settings. These are editable by Host users on the Configure screen. They are used like other tokens - [[Module:settingName]] where settingName is the name you give the setting. See the "Configuring Your XMod Pro Module" topic for details.

ADDED: FileUpload form control now has an option to be a FilePicker with optional upload or a FilePicker with no upload option - in addition to the already implemented functionality.

ADDED: Templates can now be auto-generated from Views in addition standard tables.

ADDED: Improved error reporting when trying to retrieve a record.

ADDED: Improved form editing on the Manage Forms screen as well as the Inline Editor for forms. When editing a form, the AddForm and EditForm appear in their own editors on their own tabs. This makes it much easier to identify and work with them.

ADDED: Improved clarity of Host "Edit" panel. The configured form and template are now labeled as such and their order of appearance now matches the Manage Forms, Manage Templates order of appearance on the Control Panel screen's toolbar.

ADDED: Slideshow and DataList to Quick Template Creator on Manage Templates screen. The Creator now pops-up in a dialog box.

FIXED: Unused Field Tokens in <Email> tags as well as in button redirects are now removed.

FIXED: Host "Edit" panel is now hidden when you set your DNN page to "View" mode.

FIXED: Issue where tabs on Configure page wouldn't render on some DNN 4.3+ installations.

FIXED: Localized Host "Edit" panel labels.

2.6.1

FIXED: "Object reference not set" error that occurred in forms with <Email> tags that didn't contain the SendIf attribute.

2.6.0

ADDED: AJAX to <xmod:Template> and <xmod:DataList> tags. By setting 'Ajax="True" ' on your tag, your view will have AJAX-enabled paging, search, sort, detail views, deletions, as well as commands (i.e. click a command button in one template and the content of another template will update without a page postback.). Important: All add, edit, delete, detail, and command buttons/images/links MUST have their ID specified in order to function correctly when AJAX is enabled.

ADDED: Inline Editor now automatically resizes to fill available space. So, if you are working with a larger monitor you automatically get a larger editor in which to work without having to manually resize the editor.

ADDED: Checkbox validator <Validate Type="Checkbox" .../>. Set the MustBeChecked property to True or False depending on whether checked or un-checked makes the control valid. The default value is True.

ADDED: Basic conditional email sending by using the new SendIf attribute of the <Email> tag. This attribute does basic equality checking - text-only, ignoring case. It also evaluates field tokens [[FieldName]]. Just use "=" or "<>". Example: SendIf='[[Department]] = Sales'

FIXED: When deleting the last record on the last page, the entire template disappears.

FIXED: Pager - First/Last links would not display if ShowFirstLast was set to true but no Pager Template was specified.

FIXED: Pager - Prev/Next would display even if ShowPrevNext was False if no Pager template was specified.

FIXED: [[Module:TabId]] token now works in templates. Previously it only worked in forms.

2.5.1

FIXED: Issue with Live Preview in Form Builder where theme previewing would not work due to Javascript error.

2.5.0

ADDED: New DataList View control. This control is used in the same manner as the standard Template control. It enables you to layout your records in a grid-like pattern. So, using the RepeatColumns property, you can specify that the control display three records in each row. Further, you can have the control display the records in a left-to-right, then top-to-bottom
order or a top-to-bottom, then left-to-right.

- **ADDED:** New Slideshow View control. This control is used in the same manner as the standard Template control, except it is geared specifically to display a fading slideshow of images. All you have to do is supply it with a list of image URL's from your database.

- **ADDED:** [[Request:Locale]] token. This returns the current culture in the form of en-US, en-GB, es-MX, fr-FR, etc.
- **ADDED:** Form/Template static text localization via the use of resource (.resx) files and the [[Localize:keyName]] token.
- **ADDED:** Content localization and dynamic loading of forms/templates based on current culture. This allows forms and templates to be defined for each culture. XMod Pro will automatically select the appropriate form/template based on the currently selected culture.

- **ADDED:** CC property to form <email> tag to allow carbon copy recipients of emails.
- **ADDED:** BCC property to form <email> tag to allow blind carbon copying emails to a list of recipients.
- **ADDED:** CharacterCount attribute (and supporting CharacterCountLabel, CharacterCountClass, and MaxLength attributes) to <Textarea> form control. This optional attribute can be set to CountUp or CountDown. When set to CountUp, the number of characters the user has typed into the control will be displayed just after the control. If the value is set to CountDown, the number of characters remaining will be displayed. Remaining characters are calculated based on the MaxLength property. Additional text can be displayed with CharacterCountLabel. The number and label can be styled with a CSS class name via the CharacterCountClass.

- **FIXED:** <xmod:MetaTags> tag values don't persist after changing pages via the pager.
- **FIXED:** <xmod:MetaTags> tag data-bound values don't persist when postback occurs outside the template in which the MetaTags tag resides.
- **FIXED:** Install problem on DNN 4.3.x.
- **FIXED:** Added localization text for Control Panel (ControlPanel.ascx.resx)

### 2.1.0

- An interim release never released publicly.

### 2.0.3

- **FIXED:** File not found error when attempting to preview a form with a very large number of controls.
- **FIXED:** Editor for forms and templates inserts "ClientScript" as the BlockType when inserting a "StartupScript" ScriptBlock tag.
- **FIXED:** Form Builder - Issue on some installations where Theme list and Form Preview did not appear.

### 2.0.2

- **FIXED:** "Table Not Found" issue when generating forms/templates from tables on sites whose tables use a prefix.
- **FIXED:** Minor issue where Unique ID drop-down list would not re-appear on Form Builder after user had selected the blank item in the table list and then selected a table.

### 2.0.1

- **FIXED:** "Object Reference Not Set" error being returned when trying to load list of tables in Manage Forms and Manage Templates.
- **FIXED:** When creating a table, JSON, rather than HTML was returned for success message, resulting in a cryptic message.

### 2.0.0

- **ADDED:** Ability to use <ListItem> child tags in DualList form control
- **ADDED:** New AJAX-based Control Panel for administration of XMP.
- **ADDED:** New Manage Forms page as part of Control Panel. This includes the ability to preview, edit, rename, copy, and delete items. It also includes an enhanced editor and the ability to automatically generate forms from tables in the DNN database.
• ADDED: New Manage Templates page as part of Control Panel. This includes the ability to preview, edit, rename, copy, and delete items. It also includes an enhanced editor and the ability to automatically generate templates from tables in the DNN database.

• ADDED: New Database Tools page that allows you to create basic tables in the DNN database quickly.
• ADDED: New Form Builder page in Control Panel that makes it a cinch to quickly create forms tied to database tables in the DNN database. The form builder includes options for layout and themes/styling of the form along a live preview.

• FIXED: Issue with FileUpload control where using JOIN or other tokens in Path attribute resulted in the control ignoring the Path.
• FIXED: Template & Form Editor - ScriptBlock, missing closing " around type="text/javascript". Also tweaked spacing a bit.
• FIXED: Issue where dependent lists threw Object Reference Not Set Error when lists in a Tabstrip

1.5.0

• ADDED: Dual-List form control. This control displays two list boxes and provides buttons to move items from one list box to the other and vice versa. The control is a handy alternative to a CheckBoxList as it provides more efficient use of space when many options are available.

• ADDED: Dependent Lists. The DropDownList control now has the capability to cause a second DropDownList control to reload its values based on the selected value in the first control. A good example of this would be selecting a country in control one and having a list of cities from that country be loaded into the second control.

• FIXED: Issue where form controls in TabStrip would not have their ID's populated in the Javascript Helper.
• FIXED: Issue with FormView's inline editor not showing in child portals.

1.4.2

• FIXED: Issue where Inline Editor wouldn't pull up the form or template in a child portal.

1.4.1

• ADDED: Inline Editor for configured templates and forms. You now have one-click access to quickly edit and save changes to your templates and forms. Plus, the new inline editor has a load of helpful shortcuts to quickly add tags, controls, and tokens to your forms/templates.

• ADDED: POSTed value persistence. For instance, if a value is POSTed to an XMP instance and used as a <ListDataSource> <parameter> value to filter that data, doing a search now searches in the filtered result set.

• ADDED: <SeparatorTemplate> to <xmod:Template> tag. This enables code to be inserted between records and not have any extra code at the end, as would happen if it were included in the <ItemTemplate> and/or <AlternatingItemTemplate> (e.g. Before: 1,2,3, Now: 1,2,3)

• ADDED: New Portal Tokens: Alias (gets domain); TimeZoneOffset (minutes from GMT); Description; Expiry; LoginTabId
• ADDED: New Module Token: TabId
• ADDED: Further enhanced error reporting to better report errors which occur during data-binding of controls.
• ADDED: Ability for <parameter> in <DeleteCommand> to use Alias property. To use it, the DeleteButton/Image/Link must use the name specified in Alias of the DeleteCommand’s <parameter> tag - just as they already had to specify the same name as the Name property of the <DeleteCommand>'s <parameter> tag when using that.

• ADDED: Help File - Added note to <xmod:redirect> topic that tilde (~) can be used to represent the site root in the Target attribute.
• FIXED: Issue with parser - [[Join()]], [[SelectDataTokens]], etc. weren't parsed correctly in Add/Update/Cancel button/image/link if the the tag was not named with all lower-case letters (<addbutton> <updatelink>, etc.) This has been fixed.
• FIXED: Issue in <xmod:CommandLink> - OnClientClick didn't work.
• FIXED: When ShowTopPager in <Pager> was set to False, an "Object reference not set" exception would be thrown after deleting a record.
• FIXED: GET method on <xmod:redirect> wouldn't recognize tilde (~).
• FIXED: <Add/Update/CancelButton/Image/Link> did not process JOIN token when redirect method was POST data - only when doing a GET.

1.4.0
• Beta release only.

1.3.3
• Beta release only.

1.3.2
• FIXED: CommandImage (and CommandButton) only executes one command.

1.3.1
• FIXED: If a parameter in a template did not have a value, a Conversion of DBNull to String error was thrown. This has been corrected.

1.3.0
• ADDED: Ability to use SQL OUTPUT parameter values in templates.
• ADDED: Ability to use values from <SelectCommand> in form control attribute values. Simply use the [[FieldName]] syntax as with Field Tokens in templates.
• ADDED: Added function to <xmod:format> tag to HTMLEncode/HTMLDecode text.
• ADDED: Added function to <xmod:format> tag to URLEncode/URLDecode text.
• ADDED: Add property to TextBox and TextArea to encode as HTML.
• ADDED: Ability for function tokens [[Portal]] [[Join()]] etc. to be processed in emails in addition to [[Field]] tokens.
• ADDED: Enhanced trapping of errors. More errors should now be displayed on-screen rather than just logged to the event viewer.
• ADDED: User-selectable page size for Manage Forms and Manage Templates. Selection is persisted via cookies.
• FIXED: DeleteCommand ConnectionString issue. DeleteCommand was not recognizing a ConnectionString when specified, regardless of whether it was hard-coded or a [[ConnectionString:connName]] token. It has been fixed and tested.

• FIXED: Fix Bug in Parser where [[Form:....]] token wasn't being properly parsed in <NoItemsTemplate> tag.
• FIXED: Cancel button won't cancel if CAPTCHA is invalid.
• FIXED: Help File: <xmod:EditLink> topic's example used <editlink> instead of <xmod:editlink> tag.
• FIXED: Issue when exporting module settings. Part of the resulting XML was malformed.
• FIXED: Help file <xmod:template> topic showed "UpdateRoles" as a property. This was changed to show the correct property name "EditRoles".
• FIXED: When paging is enabled and only one page is returned, "1" appeared in the pager. Instead the label for that page should be hidden when only one page.

1.2.1
• FIXED: Issue where License and Activation page was still looking for DevExpress v.7.3 files, not 9.1 files.

1.2.0
• ADDED: <xmod:ToggleLink>, <xmod:ToggleButton>, and <xmod:ToggleImage> template tags. These leverage jQuery in DNN 5+ and make it easy to hide and show elements anywhere on your page. No need to write your own Javascript.

• ADDED: <Text> form control. This control enables you to render retrieved values (from a <SelectCommand> ) out to the form. It performs one-way binding. It does not send its value to the <SubmitCommand>.
• ADDED: ShowTopPager and ShowBottomPager to <pager> template tag. Both attributes are True by default, meaning both the top and bottom paging controls will be displayed when paging is enabled. Setting either to False will hide the corresponding paging control.

• ADDED: <ControlDataSource> tag in forms now can be used with <parameter> tags to filter the results returned based on dynamic data such as the Portal ID or User ID or a parameter sent via the URL.

• ADDED: Ability for tilde (~) character to be used in the Url attribute of <xmod:ScriptBlock> and <ScriptBlock>. At runtime, the tilde character will be replaced with the path to the root of the web application.

• FIXED: Help File, Function Tokens topic. Incorrect syntax in example for Join()
• FIXED: Fix Help File for <ContinueButton> <ContinueImage> and <ContinueLink> plus the example(s) in <AddForm>, <EditForm> topic. They should be: <xmod:ContinueButton>, <xmod:ContinueImage>, <xmod:ContinueLink>

• FIXED: Form's <AddSuccessTemplate> syntax is incorrect in the help file. It is listed as <AddSuccess> and should be <AddSuccessTemplate>
• FIXED: Editor - incorrect parsing of [[Form: paramName]] in a <parameter> tag.
• FIXED: Editor - incorrect parsing of [[Join()]] function token when used in <addbutton>, <updatebutton>, and <cancelbutton>.
• FIXED: Editor - incorrect parsing parsing of <xmod:register> tags
• FIXED: BUG - When UsePager is set to False on <xmod:template>, if you delete a record, an Object Reference Not Set error is thrown.
• FIXED: Help File: <xmod:AddButton/Image/Link> doesn't specify you can use <parameter> tags.
• CHANGED: XMP now uses DevExpress v9.1 under the hood for administration pages. Previous users MUST delete DevExpress v7.3 files from their bin directory or they will get "ambiguous" errors.

1.0.0

• ADDED: 'function' and [[Field]] tokens to form redirect buttons. They are processed in Add, Update, and Cancel buttons in FormView and Main
• ADDED: Can use ~ in POST redirect to represent the root of the site.
• ADDED: If one and only one field token is used in a redirect, it is assumed to be a URL and is not encoded.
• FIXED: BUG - Tabstrip causing 2 copies of list items when using AppendDataBoundItems
• FIXED: Duplicate list control items when using AppendDataBoundItems
• FIXED: Help File - Add "UsePaging" attribute to <xmod:template> topic.
• FIXED: [[Form:...]] token not working.
• FIXED: No DefaultValue attribute for List/Detail DataSource tags in Templates.
• FIXED: Extra space in <AddForm> and <EditForm> when editing.
• FIXED: <email> tags throw error when using tokens.
• FIXED: Field tokens don't work in form button redirects
• FIXED: Help File: <AddForm><EditForm> topic, <email> tag in example incorrectly used the "target" attribute. It should be "to".

1.1.0

• Initial Release
End User License Agreement (EULA)

END USER LICENSE AGREEMENT

1. READ THIS BEFORE PURCHASING:

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE PURCHASING THE LICENSE TO USE THIS SOFTWARE. BY PURCHASING THE LICENSE TO USE THIS SOFTWARE YOU ACCEPT THESE TERMS AND CONDITIONS AND UNDERSTAND THAT THEY WILL BE LEGALLY BINDING ON YOU, ANY END USER, AND THE COMPANY. IF YOU DO NOT AGREE WITH THEM, OR DO NOT WANT THEM TO BE BINDING ON YOU, YOU SHOULD NOT PURCHASE THE LICENSE TO USE THIS SOFTWARE.

2. OWNERSHIP

It is hereby understood and agreed that Kelly Ford ("Licensor") is the owner of all right title and interest to XMod Pro ("Software") which you are about to purchase a license to use and download. You, as licensee ("Licensee") through your purchase of this license to use the software do not acquire any ownership rights to the Software. The Software which accompanies this license is the property of Licensor and is protected by copyright law. Licensor reserves the right to transfer its rights herein and ownership interest in the Software. In the event Licensor transfers its rights herein and ownership interest in the Software, such transfer shall not affect the rights and obligations of any party to this agreement and this agreement shall remain in full force affect.

3. LICENSE

This is a license, not a transfer of title or ownership to the Software. In consideration of the payment of a license fee, Licensor hereby grants Licensee a nonexclusive license to use one copy of the Software on a single production website and an unlimited number of development websites. However, the websites the Software is used for, whether production or development, may only be used with a single DotNetNuke database.

For the purposes of this agreement, “development websites” are defined as a website or websites used solely for configuring, testing, and developing applications with the Software prior to installation of the Software on a production website for a single DotNetNuke database. For the purposes of this agreement, a “production website” is defined as a website running a single DotNetNuke database, which may be either a public-facing web server or a private server running on an intranet. Under no circumstances may the Software be used on more than one DotNetNuke database without purchasing additional licenses.

4. TERM

A. The license is effective until terminated. Licensee may terminate it at any time by destroying the Software together with all copies thereof.

B. This license will terminate upon conditions set forth elsewhere within this Agreement or if Licensee fails to comply with any term or condition of this Agreement. In such event, no notice shall be required by Licensor to effect such termination.

C. Upon termination of this agreement, Licensee agrees to destroy the Software together with all backup copies, modifications, printed or written materials, and merged portions in any form, or return same to Licensor at Licensee's expense.

5. RESTRICTIONS ON USE

Except for the initial downloading of the Software on a hard disk or for archival/backup purposes as provided for above, Licensee shall not, without Licensor's express written consent:

(1) Copy or reproduce the Software in any way.

(2) Use the software on more than three DotNetNuke databases without purchasing additional licenses.

(3) Transfer, distribute, modify or translate the Software, or any portion thereof.

(4) Disassemble, decompile, or otherwise reverse engineer the Software or any of its data in any way, shape, or form.

(5) Distribute License Codes, in whole or in part, for this product to any other party.

7. UPDATES

This license does not entitle Licensee to any upgrades. Licensor may, at its sole discretion, make certain free patches and service packs available to Licensees.
This license, if purchased with a subscription, entitles the Licensee to certain free upgrades, patches, and service packs made available by the Licensor during the subscription period.

Any license to use any upgrades, supplements, add-on components, patches, and service packs to the software subsequently provided by Licensor, whether purchased or not, are subject to the same terms and conditions as provided herein. Licensor reserves the right to supplement this agreement with additional terms along with any subsequent upgrades, supplements, add-on components, patches, and service packs to the software.

8. WARRANTY

A. THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. LICENSOR DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE. The entire risk as to the quality and performance of the Software is with you. Should the Software prove defective, you (and not the Licensor) assume the entire cost of all necessary servicing, repair, or correction.

B. Licensee shall assume responsibility for the selection of the Software and for the installation, use, and results obtained from the Software.

C. Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights that vary from state to state.

D. By purchasing license to use and download the software, you agree to accept all liability regarding its use or misuse.

9. DISCLAIMER

In no event will Licensor be liable for any damages, including but not limited to, lost profits, lost savings, or other incidental or consequential damages arising out of the use or inability to use such Software even if Licensor or its authorized dealer has been advised of the possibility of such damages, or for any claim by any other party. Some states do not allow the limitations or exclusion of liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

11. JURISDICTION AND DISPUTES

A. This Agreement and the Limited Warranty shall be governed by the laws of Arizona.

B. All disputes hereunder shall be resolved in the applicable state or federal courts of Arizona. The parties consent to the jurisdiction of such courts, agree to accept service of process by mail, and waive any jurisdictional or venue defenses otherwise available.

12. ENTIRE AGREEMENT AND SEVERABILITY

This Agreement constitutes the entire understanding of the parties, and revokes and supersedes all prior agreements, oral or written, between the parties, and is intended as a final expression of their Agreement. It shall not be modified or amended except in writing signed by the parties hereto and specifically referring to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict therewith. If any provision of this Agreement is held to be void, invalid, unenforceable or illegal, the other provisions shall continue in full force and effect.

LICENSEE HEREBY ACKNOWLEDGES THAT IT HAS READ THIS AGREEMENT, UNDERSTANDS IT, AND BY PURCHASING THE LICENSE TO USE THIS SOFTWARE AGREES TO BE BOUND BY ITS TERMS AND CONDITIONS.
Data View Controls

<xmod:AddButton>

The AddButton tag renders as a push-button at run-time. It is used to show the form defined by the <addform> tag in the module instance's selected form.

Syntax

```
<xmod:AddButton
   BackColor="color name|#dddddd"
   BorderColor="color name|#dddddd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge| Inset|Outset"
   BorderWidth="size"
   CssClass="string"
   Font-Bold="True|False"
   Font-Italic="True|False"
   Font-Names="string"
   Font-Overline="True|False"
   Font-Strikeout="True|False"
   Font-Underline="True|False"
  ForeColor="color name|#dddddd"
   Height="size"
   OnClientClick="javascript"
   Style="string"
   Text="string"
   ToolTip="True|False"
   Visible="True|False"
   Width="size">
   ...
   <Parameter Name="string" Value="string" />
   <Parameter Name="string" Value="string" />
   ...
   additional parameters as needed ...
</xmod:AddButton>
```

Remarks

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return `true` then the control will perform its normal processing.
- **Parameter Tags**: Use of the `<parameter>` tags is optional. Use them if you want to pass values to the `<AddForm>`.
**Example**

```xml
<div>
<table width="100%">
<tr>
  <td width="250" valign="top">
    <!-- EMPLOYEES TEMPLATE -->
    <xmod:template id="Employees">
      <listdatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
        <parameter name="DepartmentId" alias="DepartmentId"/>
      </listdatasource>
      <headertemplate>
        <p>Employees</p>
      </headertemplate>
      <itemtemplate>
        <div style="text-align: middle;">
          <strong>[[FirstName]] [[LastName]]</strong>
        </div>
      </itemtemplate>
      <footertemplate>
        <xmod:addbutton text="New Employee" />
      </footertemplate>
    </xmod:template>
  </td>
</tr>
</table>
</div>
```

**Syntax**

The **AddImage** tag renders as a clickable image at run-time. It is used to show the form defined by the **addform** tag in the module instance's selected form.

**Remarks**

**Example**

```xml
<xmod:AddImage>
  AlternateText="string" 
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Normal="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  ImageAlign="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbsBottom|AbsMiddle|TextTop"
  ImageUrl="url"
  OnClientClick="javascript"
  Style="string"
  ToolTip="string"
</xmod:AddImage>
```
Visible="True|False"
Width="size">
<Parameter Name="string" Value="string" />
<Parameter Name="string" Value="string" />
...additional parameters as needed ...
</xmod:AddImage>

Remarks

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.
- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application’s root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Parameter Tags**: Use of the <parameter> tags is optional. Use them if you want to pass values to the s.

Example

```xml
<div>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <!-- EMPLOYEES TEMPLATE -->
      </td>
    </tr>
    <xmod:template id="Employees">
      <listdatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
        <parameter name="DepartmentId" alias="DepartmentId"/>
      </listdatasource>
      <headertemplate>
        <p>Employees</p>
      </headertemplate>
      <itemtemplate>
        <div style="text-align:middle;">
          <strong>[[FirstName]] [[LastName]]</strong>
        </div>
      </itemtemplate>
      <footertemplate>
        <xmod:addimage alternatetext="New Employee" imageurl="~/images/add.gif" />
      </footertemplate>
    </xmod:template>
  </table>
</div>
```

<xmod:AddLink>
Syntax Remarks Example

The AddLink tag renders as a clickable image at run-time. It is used to show the form defined by the <addform> tag in the module instance's selected form.
Syntax

```xml
<modx:AddLink
   BackColor="color name|#dddedd"
   BorderColor="color name|#dddedd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
   BorderWidth="size"
   CssClass="string"
   Font-Bold="True|False"
   Font-Italic="True|False"
   Font-Names="string"
   Font-Overline="True|False"
   Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
   Font-Strikeout="True|False"
   ForeColor="color name|#dddedd"
   Height="size"
   OnClientClick="javascript"
   Style="string"
   Text="string"
   ToolTip="string"
   Visible="True|False"
   Width="size">
   <Parameter Name="string" Value="string" />
   <Parameter Name="string" Value="string" />
   ...additional parameters as needed ...
</modx:AddLink>
```

Remarks

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return `true` then the control will perform its normal processing.

- **Parameter Tags**: Use of the `<parameter>` tags is optional. Use them if you want to pass values to the `<AddForm>`.s
Example

```xml
<div>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <!-
        -- EMPLOYEES TEMPLATE -->
        <xmod:template id="Employees">
          <listdatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
            <parameter name="DepartmentId" alias="DepartmentId"/>
          </listdatasource>
          <headertemplate>
            <p>Employees</p>
          </headertemplate>
          <itemtemplate>
            <div style="text-align: middle;">
              <strong>[[FirstName]] [[LastName]]</strong>
            </div>
          </itemtemplate>
          <footertemplate>
            <xmod:addlink text="New Employee"/>
          </footertemplate>
        </xmod:template>
      </td>
    </tr>
  </table>
</div>
```

**Syntax:** Example

The `CommandButton` tag renders as a push-button at run-time. It is used to execute data commands in another template within the module instance. For instance, if you had two templates, you might put a `CommandButton` in template #1 to pass a parameter to the `<List-dataSource>` of template #2, causing that template to re-load with the new result set.
Syntax

```xml
<xml:CommandButton
   BackColor= "color name | #dddddd"
   BorderColor= "color name | #dddddd"
   BorderStyle= " NotSet | None | Dotted | Dashed | Solid | Double | Groove | Ridge | Inset | Outset"
   BorderWidth= "size"
   CssClass= "string"
   Font-Bold= "True | False"
   Font-Italic= "True | False"
   Font-Names= "string"
   Font-Overline= "True | False"
   Font-Size= "string | Smaller | Larger | XX-Small | X-Small | Small | Medium | Large | X-Large | XX-Large"
   Font-Strikeout= "True | False"
   ForeColor= "color name | #dddddd"
   Height= "size"
   OnClientClick= "javascript"
   Style= "string"
   Text= "string"
   ToolTip= "string"
   Visible= "True | False"
   Width= "size">
  <Command Target= "string" Type= "List | Detail">
    <Parameter Name= "string" Value= "string" />
    <Parameter Name= "string" Value= "string" />
    additional parameters as needed ... 
  </Command>
  additional commands as needed ...
</xml:CommandButton>
```

Remarks

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.

- **Usage**: Command controls are used to execute a pre-defined data command in another template. You can execute multiple commands within a single control. Commands are defined in the `<command>` child tags. Each command will be executed in sequence, but please note that there is no transaction assumed. If a command fails, all those that went before it will NOT be rolled back.

  For each command, you identify the template whose datasource you want to execute by specifying the template's ID in the "target" attribute. If you set the "type" attribute to List, the `<ListDataSource>` will be executed and passed any parameters you specify via `<parameter>` child tags.
### Example

```html
<example>
<example>

```
The CommandImage tag renders as a clickable image at run-time. It is used to execute data commands in another template within the module instance. For instance, if you had two templates, you might put a CommandImage in template #1 to pass a parameter to the template #2, causing that template to re-load with the new result set.

**Syntax**

```xml
<modx:CommandImage
    AlternateText="string"
    BackColor="#dddddd"
    BorderColor="#dddddd"
    BorderRadius="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge" Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Small|Large|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
   ForeColor="#dddddd"
    Height="size"
    ImageAlign="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbsBottom|AbsMiddle|TextTop"
    ImageUrl="url"
    OnClientClick="javascript"
    Style="string"
    ToolTip="string"
    Visible="True|False"
    Width="size">

    <Command Target="string" Type="List|Detail">
        <Parameter Name="string" Value="string" />
        <Parameter Name="string" Value="string" />
        additional parameters as needed ...
    </Command>

additional commands as needed ...

```/modx:CommandImage>

**Remarks**

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.
- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application's root directory. For instance: ImageUrl="/~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Usage**: Command controls are used to execute a pre-defined data command in another template. You can execute multiple commands within a single control. Commands are defined in the <command> child tags. Each command will be executed
in sequence, but please note that **there is no transaction assumed**. If a command fails, all those that went before it will NOT be rolled back.

For each command, you identify the template whose datasource you want to execute by specifying the template's ID in the "target" attribute. If you set the "type" attribute to List, the `<ListDataSource>` will be executed and passed any parameters you specify via `<parameter>` child tags.

[Back to top](#)
### Example

```html
<div>
<table width="100%">
  <tr>
    <td colspan="2">
      <!-- DEPARTMENTS TEMPLATE -->
      <xmod:template id="Departments">
        <listdatasource commandtext="SELECT DepartmentId, DepartmentName FROM XMPDemo_Departments ORDER BY DepartmentName" />
        <itemtemplate>
          <xmod:commandbutton text='[[DepartmentName]]'>
            <command target="Employees" type="list">
              <parameter name="DepartmentId" value='[[DepartmentId]]'/>
            </command>
            <command target="EmployeeProfile" type="detail">
              <parameter name="EmployeeProfile" value="-1"/>
            </command>
          </xmod:commandbutton>&nbsp;
        </itemtemplate>
      </xmod:template>
    </td>
  </tr>
  <tr>
    <td width="250" valign="top">
      <!-- EMPLOYEES TEMPLATE -->
      <xmod:template id="Employees">
        <listdatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
          <parameter name="DepartmentId" alias="DepartmentId"/>
        </listdatasource>
        <headertemplate>
          <p>Employees</p>
        </headertemplate>
        <itemtemplate>
          <div style="text-align: middle;">
            <xmod:commandimage text="Profile" imageurl="/images/icon_hostusers_32px.gif" imagealign="absmiddle">
              <command type="detail" target="EmployeeProfile">
                <parameter name="EmployeeProfile" value='[[EmployeeProfile]]'/>
              </command>
            </xmod:commandimage>
          </div>&nbsp;
        </itemtemplate>
      </xmod:template>
    </td>
    <td width="500" valign="top">
      <!-- EMPLOYEE PROFILE TEMPLATE -->
      <xmod:template id="EmployeeProfile">
        <detaildatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmployeeId">
          <parameter name="EmployeeProfile" alias="EmployeeProfile" value="-1"/>
        </detaildatasource>
        <detailtemplate>
          <h1>Employee Profile</h1>
          <p style="font-size: 14px; font-weight: bold;">[[FirstName]] [[LastName]]</p>
          <p style="font-size: 12px; font-weight: bold;"><em>[[JobTitle]]></em></p>
          <p>[[Resume]]</p>
        </detailtemplate>
      </xmod:template>
    </td>
  </tr>
</table>
</div>
```
The CommandLink tag renders as a hyperlink at run-time. It is used to execute data commands in another template within the module instance. For instance, if you had two templates, you might put a CommandLink in template #1 to pass a parameter to the <listdatasource> of template #2, causing that template to re-load with the new result set.

Syntax

```xml
<mod:CommandLink
  BackColor="color name"|#dddddd"
  BorderColor="color name"|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge| Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name"|#dddddd"
  Height="size"
  OnClientClick="javascript"
  Style="string"
  Text="string"
  ToolTip="string"
  Visible="True|False"
  Width="size">
  <Command Target="string" Type="List|Detail">
    <Parameter Name="string" Value="string" />
    <Parameter Name="string" Value="string" />
  </Command>
  additional parameters as needed ...
</mod:CommandLink>
```

Remarks

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Usage**: Command controls are used to execute a pre-defined data command in another template. You can execute multiple commands within a single control. Commands are defined in the <command> child tags. Each command will be executed in sequence, but please note that there is no transaction assumed. If a command fails, all those that went before it will NOT be rolled back.

For each command, you identify the template whose datasource you want to execute by specifying the template's ID in the "target" attribute. If you set the "type" attribute to List, the <ListDataSource> will be executed and passed any parameters you specify via <parameter> child tags.
Example

```xml
<div>
<table width="100%">
<tr>
<td colspan="2">
<!-- DEPARTMENTS TEMPLATE -->
<xmod:template id="Departments">
<listdatasource commandtext="SELECT DepartmentId, DepartmentName FROM XMPDemo_Departments ORDER BY DepartmentName" />
<script>

</script>
</xmod:template>
</td>
</tr>
<tr>
<td width="250" valign="top">
<!-- EMPLOYEES TEMPLATE -->
<xmod:template id="Employees">
<listdatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
<script>

</script>
</xmod:template>
</td>
<td width="500" valign="top">
<!-- EMPLOYEE PROFILE TEMPLATE -->
<xmod:template id="EmployeeProfile">
<detaildatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmployeeId">
<script>

</script>
</xmod:template>
</td>
</tr>
</table>
</div>
```

Back to top
<xmod:DataList>

**Syntax**

The DataList tag, like the Template tag, is a View control that is used for displaying and interacting with records from your datasource. It is used in essentially the same manner as the Template tag and can be used together with the Template tag within your templates. Like the Template tag, it contains the data commands and layout instructions for your display and enables you to specify who is allowed to add, edit, and delete records as well as who is allowed to view the details of records. The primary difference between the DataList and Template is that the DataList allows you to layout your records in a grid pattern.

<table>
<thead>
<tr>
<th>Record1</th>
<th>Record2</th>
<th>Record3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record4</td>
<td>Record5</td>
<td>Record6</td>
</tr>
<tr>
<td>Record7</td>
<td>Record8</td>
<td></td>
</tr>
</tbody>
</table>

----- OR ----

<table>
<thead>
<tr>
<th>Record1</th>
<th>Record4</th>
<th>Record7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record2</td>
<td>Record5</td>
<td>Record8</td>
</tr>
<tr>
<td>Record3</td>
<td>Record6</td>
<td></td>
</tr>
</tbody>
</table>

This type of layout is handy for scenarios like displaying a list of images or products where you want to control the specific number of columns and the order in which they're displayed.

You can use multiple DataList tags, enabling you to have side-by-side (or however you want to lay them out using HTML) displays within the same module instance, each being fed by different datasources.

**Syntax**

```xml
<xmod:DataList
    AddRoles="DNNRoleName1,DNNRoleName2"
    Ajax="True|False"
    ConnectionString="string"
    DeleteRoles="DNNRoleName1,DNNRoleName2"
    DetailRoles="DNNRoleName1,DNNRoleName2"
    EditRoles="DNNRoleName1,DNNRoleName2"
    ID="string"
    RepeatColumns="integer"
    RepeatDirection="Horizontal|Vertical"
    RepeatLayout="Table|Flow"
    UsePaging="True|False">
    <ListDataSource CommandText="string"
        ConnectionString="string"/>
    <DetailDataSource CommandText="string"
        ConnectionString="string"/>
    <DeleteCommand CommandText="string"/>
    <Pager>...
        </Pager>
    <SearchSort>...
        </SearchSort>
    <HeaderTemplate>...
        </HeaderTemplate>
    <ItemTemplate>...
        </ItemTemplate>
    <AlternatingItemTemplate>...
        </AlternatingItemTemplate>
    <SeparatorTemplate>...
        </SeparatorTemplate>
    <FooterTemplate>...
        </FooterTemplate>
    <DetailTemplate>...
        </DetailTemplate>
    <NoItemsTemplate>...
        </NoItemsTemplate>
</xmod:DataList>
```
Remarks

- **AddRoles/EditRoles/DeleteRoles/DetailRoles**: Each of these properties are lists of DNN role names, separated by commas. Host/SuperUser and Admin accounts always have add/update/delete/detail permissions. By default, anyone who has access to view the module also has detail viewing permissions. Supply the DetailRoles attribute to limit who can view the details of records within the DataList. Users other than Host/Admin must be explicitly granted add/update/delete rights. Prior to version 4.0, role names were separated by semi-colons. To enhance consistency with other areas of XMod Pro, commas are now used. For backwards compatibility, you can still use semi-colons, but their usage is deprecated as of version 4.0. You should use commas going forward.

  **NOTE**: These permissions apply ONLY to controls within the `<xmod:DataList>` tag. If you have an `<xmod:AddButton>`, and `<xmod:AddLink>`, or `<xmod:AddImage>` tag placed outside of the `<xmod:DataList>` tag, you will need to set permissions at the module level by going to the module's Configure page and choosing the Security tab. There you can choose the roles allowed to add records.

- **Ajax**: New to version 2.6, when this attribute is true, actions within the template tag will be performed without a full page refresh. This value is false by default. **NOTE**: buttons that postback such as the DetailButton (or DetailImage or DetailLink) or AddButton or DeleteButton or CommandButton, etc. must have their ID property set to function correctly. If you have to click twice on a button, for instance, make sure its ID has been set.

- **ConnectionString**: If you need to specify a SQL Server database other than the current DotNetNuke database, you can supply a connection string here. When specified, it serves as the default connection string for the ListDataSource and DetailDataSource. You can also use a connection string defined in the web.config file. To do so, use the ConnectionString token like so:

  `<xmod:DataList ... ConnectionString="[[ConnectionString:connectionName]]" ...>

- **RepeatColumns**: Determines the maximum number of columns that will be used for displaying records. If there are more records than the number of columns, the DataList will create new rows. Keep in mind that, unlike standard tables where each column contains a field from the record, in a DataList each column contains a full record.

- **RepeatLayout**: Determines the underlying HTML that is used to create the column layout. **Table** indicates the grid pattern will be created by SPAN and BR tags.

- **RepeatDirection**: Determines in which order the columns and rows are filled. If set to **Horizontal**, the columns and rows are filled from left to right and then top to bottom. If set to **Vertical**, columns and rows are filled from top to bottom first and then left to right.

- **UsePaging**: If set to true, the default, list views will display a pager if more than one page is available for display. The pager can be set using the `<Pager>` tag.

- **Data Sources**: The data for each view is a SQL command. This could be a SQL SELECT command or a stored procedure. Supply the SQL command as the commandtext attribute. For stored procedures, use EXEC sprocName. In the initial release, data sources can only point to the current DotNetNuke database. Also note that Bit columns are returned as True/False - not 1/0. The data source for list views is defined in the `<ListDataSource>` tag while the data source for the detail view is defined in the DetailDataSource tag.

In addition to its attributes, the DataList tag also contains numerous child tags which define most of the tag's functionality:

- **ListDataSource**: Provides the data for the DataList when it is displaying in list view. It accepts `<parameter>` tags to pass parameters as part of the command. The "alias" attribute is optional and is used to avoid conflicts or to accept a parameter with a specific name (the "name" attribute) but use a different parameter name in the command (the "alias" attribute). If no "alias" is specified, the "name" will be used.

  **Output Parameters**: New to version 1.3, you can retrieve and use the value of SQL Output parameters. To do so, add the **Direction="Output"** to the `<parameter>` tag and specify a **DataType** and **Size** (required for Strings). See Example #4 and the Data Parameter Tokens Topic.

Examples:
#1
<ListDataSource CommandText="SELECT FirstName, LastName FROM Users" />

#2
<ListDataSource CommandText="SELECT FirstName, LastName FROM Users WHERE ZipCode = @zip" />

#3
<ListDataSource CommandText="EXEC GetUsers @zip" />

#4
<ListDataSource CommandText="GetEmployesByDepartment" CommandType="StoredProcedure">
  <parameter name="DepartmentId" value="32"/>
  <parameter name="DepartmentName" direction="Output" datatype="string" size="100"/>
</ListDataSource>

You can optionally specify a **ConnectionString** for this data source. If none is specified, the default connection will be used - either the connection specified by the ConnectionString property of the parent `<mod:DataList>` tag or the current DotNetNuke database if the DataList tag doesn't define a connection. As with the DataList tag, you can use the [{ConnectionString:connectionName}] token to use a connection defined in the web.config file.

- **DetailDataSource**: Provides the data for the DetailTemplate when it is displaying a single record. It accepts `<parameter>` tags like the `<ListDataSource>`. You can optionally specify a **ConnectionString** for this data source. If none is specified, the default connection will be used - either the connection specified by the ConnectionString property of the parent `<mod:DataList>` tag or the current DotNetNuke database if the DataList doesn't define a connection. As with the DataList tag, you can use the [[ConnectionString:connectionName]] token to use a connection defined in the web.config file.

DetailDataSource tags can also take **Output Parameters**. See the ListDataSource topic above for details.

- **DeleteCommand**: Provides the command to execute when a Delete Button/Image/Link is pressed within the DataList. It accepts `<parameter>` tags like the `<ListDataSource>`

- **HeaderTemplate**: When displaying a list view, HTML, text, and controls that will be rendered once, at the beginning of the list. This template is optional.

- **ItemTemplate**: When displaying a list view, HTML, text, field tokens, and controls that will be rendered for each record in the result set. If the AlternatingItemTemplate is supplied, then the ItemTemplate will be used for all odd-numbered records.

- **AlternatingItemTemplate**: When displaying a list view, HTML, text, field tokens, and controls that will be rendered for the even-numbered records in the result set. This template is optional.

- **SeparatorTemplate**: When displaying a list view, this will render any HTML or text that you specify between each record. One good example is if you're creating a list of comma-separated records (we'll use numbers for simplicity). If you put your comma in the ItemTemplate, you would get a trailing comma like this: 1,2,3. But if you put your comma in the SeparatorTemplate, you won't get that trailing comma: 1,2,3. (Added in version 1.4)

- **FooterTemplate**: When displaying a list view, HTML, text, and controls that will be rendered once, at the end of the list. This template is optional.
**DetailTemplate:** Defines the layout for viewing a single record. When the user clicks a detail button, this template will be used. This template can contain HTML, text, Field Tokens, and controls. It is required if you will be using XMod Pro's detail buttons in your list view.

**NoItemsTemplate:** Defines the layout and text to display when the data source returns no records. This tag is optional. If not specified, nothing will be shown.

**Pager:** This optional tag allows you to override the default configuration and look of the top and bottom pagers. Additionally, you can optionally override the default layout of the top and bottom pagers using your own HTML. See the `<Pager>` topic for more details.

**SearchSort:** This allows you to define basic searching and sorting for your DataList. For more details see the `<SearchSort>` topic.

**Field Tokens:** In order to display data from your data source, XMod Pro uses "field tokens". These are essentially placeholders that contain the name of the column or field in your data source. At run-time, these tokens are replaced with the value from record. Field Tokens are written in this form: `[[FieldName]]"FieldName"` is the name of the field or column in your data source. This name is case sensitive and must match the field/column's name exactly. It must be surrounded by `[]`. You can use the Field Token in many places. However, when you use it as the attribute value for an XMod Pro tag or other third party control, you must delimit the attribute value with single quotes, rather than double quotes. See the `<xmod:detailbutton>` code in the example. When used in the `<parameter>` tag, the attribute is written as: `value='[[UserID]]'` rather than: `value="[[UserID]]"`. When using a Field Token with HTML tag attributes, this is not necessary.

```xml
<Pager>
</Pager>
```
**Syntax**

The Pager tag can only be used within a Template or DataList tag. The tag defines how the pagers in the view will look and function. The Pager attributes provide the information necessary to define how the components of the top and bottom pagers will look while the inner content of the Pager allows you to use HTML to determine how the components are arranged. If no inner content is specified, the default layout will be used. If no tag is specified, the pagers will not be shown.

NOTE: To turn off paging for the view, use the **UsePaging** attribute of the `<xmod:template>` or `<xmod:DataList>` tag.

**Syntax**

```
<xmod:Template|DataList ...>
...
<Pager
  FirstLastCssClass="string|CommandButton"
  FirstPageCaption="string|First"
  LastPageCaption="string|Last"
  MaxPageNumButtons="integer|5"
  NextPageCaption="string|Next"
  PageNumCssClass="string|CommandButton"
  PageSize="integer|10"
  PrevNextCssClass="string|CommandButton"
  PrevPageCaption="string|Prev"
  ScrollToTop="True|False"
  ShowBottomPager="True|False"
  ShowFirstLast="True|False"
  ShowPrevNext="True|False"
  ShowTopPager="True|False">
  ...Display Template (see Remarks)...</Pager>
</xmod:Template|DataList>
```

**Remarks**

- **FirstLastCssClass**: The Cascading Style Sheet (CSS) class name to associate with the First Page and Last Page navigation links. The default value is CommandButton.
- **FirstPageCaption**: The text to use for the First Page navigation link, when it is visible. The default value is "First".
- **LastPageCaption**: The text to use for the Last Page navigation link, when it is visible. The default value is "Last".
- **MaxPageNumButtons**: Determines the maximum number of page link buttons which will be displayed. Page number buttons enable the user to navigate to a specific page by clicking on them. They appear between the Previous and Next navigation buttons. The default value is 5.
- **NextPageCaption**: The text to use for the Next Page navigation link, when it is visible. The default value is "Next".
- **PageNumCssClass**: The Cascading Style Sheet (CSS) class name to associate with the page number link buttons. The default value is CommandButton.
- **PageSize**: Determines the maximum number of records to display on each page. The default value is 10.
- **PrevNextCssClass**: The Cascading Style Sheet (CSS) class name to associate with the Previous Page and Next Page navigation links. The default value is CommandButton.
- **PrevPageCaption**: The text to use for the Previous Page navigation link, when it is visible. The default value is "Prev".
- **ScrollToTop**: A true/false value which is set to true by default. When true, if the user clicks a link in the Bottom Pager, the page, when it is reloaded, will scroll to the top of the page. This is helpful if you have a long list of items and the user has scrolled down before clicking the Next page (or any pager) link. The default DNN action would maintain his/her position on the new page, which is not what the user would expect. By setting ScrollToTop to true, you can take the user back to the top of the page. NOTE: If the user clicks a link in the Top Pager, the page will not scroll to the top - even if this property is true. Instead, it will maintain the user's current position. (new to version 4.0)
- **ShowBottomPager**: A true/false value which determines if the Top paging control will be visible. This optional attribute is True by default. If both ShowBottomPager and ShowTopPager are set to False, no paging controls will be shown but the results will still be paged. In other words, only the first page of results will be shown without any visible mechanism to
navigate to other pages.
(Added in version 1.2)
- **ShowFirstLast**: A true/false value which determines if the First Page and Last Page navigation links should be used. The default value is False.
- **ShowPrevNext**: A true/false value which determines if the Previous Page and Next Page navigation links should be used. The default value is True.
- **ShowTopPager**: A true/false value which determines if the Bottom paging control will be visible. This optional attribute is True by default. If both ShowBottomPager and ShowTopPager are set to False, no paging controls will be shown but the results will still be paged. In other words, only the first page of results will be shown without any visible mechanism to navigate to other pages.
(Added in version 1.2)
- **Display Template**: Within the `<Pager> </Pager>` tags, you can use HTML and special pager tokens to arrange and style the components of the pager. See the Example for one possibility. Below are the tokens you can use:
  - `{PageNumber}`: Will be replaced with the current page number.
  - `{PageCount}`: Will be replaced with the total number of pages.
  - `{Pager}`: Will be replaced with the navigation links (First, Previous, Next, Last, and the page number links)

**Example**

```xml
<xmod:Template ...>
  ...
  <Pager PageSize="15" pageNumCssClass="CommandButton"
         FirstPageCaption="[First]" LastPageCaption="[Last]">
    <table>
      <tr>
        <td>Page <strong>{PageNumber}</strong> of <strong>{PageCount}</strong></td>
      </tr>
      <tr align="right">
        {Pager}
      </tr>
    </table>
  </Pager>
  ...
</xmod:Template>
```

**<SearchSort>**

**Syntax Remarks Example**

The SearchSort tag can only be used within a Template tag or DataList tag. The tag defines how the search and sort panel in the view will look and function. The SearchSort tag attributes provide the information necessary to define how the components of the search and sort panel will look and function, while the tag's inner content, if specified, allow you to use HTML and SearchSort tokens to determine how the components of the search and sort panel are arranged.
Syntax

```xml
<%:Template|DataList ...>
...
<SearchSort
BackColor="color name|#dddddd"
BorderColor="color name|#dddddd"
BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
BorderWidth="size"
FilterExpression="string"
Font-Bold="True|False"
Font-Italic="True|False"
Font-NormalTextBox="string"
Font-Underline="True|False"
ForeColor="#dddddd"
Height="size"
ReverseSortCssClass="string|Normal"
ReverseSortText="string|Reverse"
SearchBoxCssClass="string|NormalTextBox"
SearchButtonCssClass="string|CommandButton"
SearchButtonText="string|Search"
SearchLabelCssClass="string|Normal"
SearchLabelText="string|Search"
SortButtonCssClass="string|CommandButton"
SortButtonText="string|Sort"
SortFieldLabels="comma-delimited list of sort field labels"
SortFieldListCssClass="string|NormalTextBox"
SortFieldNames="comma-delimited list of field names"
SortLabelCssClass="string|Normal"
SortLabelText="string|Sort"
Width="size">
 ...
</%:Template|DataList>
```

Remarks

- **FilterExpression**: Essentially the "WHERE" clause of a SELECT query without the WHERE. It is executed when the user clicks the Search button. You use the placeholder `{0}` to represent the value entered by the user. For example: FilterExpression="FirstName LIKE '%{0}'". In this example, the `{0}` will be replaced by the value entered by the user in the Search Box. If the user enters "John" then the resulting expression will be: "FirstName LIKE '%John%'".
- **ReverseSortCssClass**: The Cascading Style Sheet (CSS) class name to associate with the Reverse Sort checkbox. The default value is "Normal"
- **ReverseSortText**: The text to use for the Reverse Sort checkbox caption. The default value is "Reverse".
- **SearchBoxCssClass**: The Cascading Style Sheet (CSS) class name to associate with the search phrase input box. The default value is "NormalTextBox"
- **SearchButtonCssClass**: The Cascading Style Sheet (CSS) class name to associate with the button that initiates the Search. The default value is "CommandButton"
- **SearchButtonText**: The caption to use for the Search Button.
- **SearchLabelCssClass**: The Cascading Style Sheet (CSS) class name to associate with the Search Label. In most cases, the Search Label precedes the Search Box to identify the purpose of the box. The default value is "Normal"
- **SearchLabelText**: The text to use in the Search Label. In most cases, the Search Label precedes the Search Box to identify the purpose of the control. The default value is "Search"
- **SortButtonCssClass**: The Cascading Style Sheet (CSS) class name to associate with the button that, when pressed, executes the sort. The default value is "CommandButton".
- **SortButtonText**: The text to use for the button that, when pressed, executes the sort. The default value is "Sort"
- **SortFieldLabels**: Many times, the names you use for fields in your data source aren't user-friendly. Supply a comma-delimited list of captions to use in the sort field list control.
- **SortFieldListCssClass**: The Cascading Style Sheet (CSS) class name to associate with the list control containing the list of fields the user can sort on. The default value is "NormalTextBox"
- **SortFieldNames**: The names of the fields in your data source that can be sorted on. This is a comma-delimited list of names.
- **SortLabelText**: The Cascading Style Sheet (CSS) class name to associate with the Sort Label. In most cases, the Sort Label precedes the sort list control to identify the purpose of the control.
- **SortLabelText**: The text to use in the Sort Label. In most cases, the Sort Label precedes the sort list control to identify the purpose of the control.
- **Display Template**: Within the `<SearchSort>` tags, you can use HTML and special tokens to arrange and style the components of the search/sort panel. See the Example for one possibility. Below are the tokens you can use:
  - `{SearchButton}`: Will be replaced with the button used to initiate the search.
  - `{SearchBox}`: Will be replaced with the input box used to enter the search phrase.
  - `{SearchLabel}`: Will be replaced by the label used to identify the purpose of the search box and search button.
  - `{SortLabel}`: Will be replaced by the label used to identify the purpose of the sort list control and button.
  - `{SortFieldList}`: Will be replaced by the list control containing the list of fields the user can sort on.
  - `{ReverseSort}`: Will be replaced by the checkbox that, when ticked, will cause the results to be sorted in reverse order.
  - `{SortButton}`: Will be replaced by the button used to initiate the sort.

### Example

```xml
<xmod:template ...>
  ...
  <SearchSort FilterExpression="FirstName LIKE '%(0)%'"
    SearchLabelText="Search For:" SearchButtonText="GO"
    SortFieldNames="FirstName,LastName,Zip"
    SortFieldLabels="First Name, Last Name, Zip Code">
    <table>
      <tr>
        <td><strong>{SearchLabel}</strong>{SearchBox} {SearchButton}</td>
        <td align="right">
          <strong>{SortLabel}</strong>{SortFieldList} {SortButton} Reverse {ReverseSort}
        </td>
      </tr>
    </table>
  </SearchSort>
</xmod:template>
```

### Syntax | Remarks | Example
---|---|---

The `DeleteButton` tag renders as a push-button at run-time. It is used to execute the `<DeleteCommand>` of its parent `<xmod:template>`.
**Syntax**

```xml
<mod:DeleteButton
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    OnClientClick="javascript"
    Style="string"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size">
    <Parameter Name="string" Value="string" />
    <Parameter Name="string" Value="string" />
    ...additional parameters as needed ...
</mod:DeleteLink>
```

**Remarks**

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns *false* the control will not perform its normal processing. If you return *true* then the control will perform its normal processing.

- **Usage**: The XMod Pro Delete controls work in conjunction with the `<DeleteCommand>` tag of the `<xmod:template>` tag. Typically, the delete command will include one or more `<parameter>` tags that identify which record(s) should be deleted. The delete control should use the same parameter names and fill them with valid values, typically from the current record. That’s why delete controls are typically found in `<ItemTemplate>` and `<AlternatingItemTemplate>` tags.
### Example

```xml
<example>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <!-- EMPLOYEES TEMPLATE -->
        <xmod:template id="Employees">
          <listdatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
            <parameter name="DepartmentId" alias="DepartmentId"/>
          </listdatasource>
          <deletecommand commandtext="DELETE FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
            <parameter name="EmployeeId" alias="EmpID"/>
          </deletecommand>
          <headertemplate>
            <p>Employees</p>
          </headertemplate>
          <itemtemplate>
            <div style="text-align: middle;">
              <strong>[[FirstName]] [[LastName]]</strong>
              <xmod:deletebutton text="Delete" onclientclick="return confirm('Are you sure you want to delete this employee?');">
                <parameter name="EmployeeId" value='[[EmployeeId]]' />
              </xmod:deletebutton>
            </div>
          </itemtemplate>
        </xmod:template>
      </td>
    </tr>
  </table>
</example>
```

### Syntax

```xml
<xmod/DeleteImage>
  <param name="AlternateText" value="string"/>
  <param name="BackColor" value="#dddddd"/>
  <param name="BorderColor" value="#dddddd"/>
  <param name="BorderStyle" value="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"/>
  <param name="BorderWidth" value="size"/>
  <param name="CssClass" value="string"/>
  <param name="Font-Bold" value="True|False"/>
  <param name="Font-Italic" value="True|False"/>
  <param name="Font-Names" value="string"/>
  <param name="Font-Overline" value="True|False"/>
  <param name="Font-Size" value="string|Smaller|Larger|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"/>
  <param name="Font-Strikeout" value="True|False"/>
  <param name="Font-Underline" value="True|False"/>
  <param name="ForeColor" value="#dddddd"/>
  <param name="Height" value="size"/>
  <param name="ImageAlign" value="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbsBottom|AbsMiddle|TextTop"/>
  <param name="ImageUrl" value="url"/>
</xmod/DeleteImage>
```

### Remarks

The DeleteImage tag renders as a clickable image at run-time. It is used to execute the `<DeleteCommand>` of its parent `<xmod:template>`.
Remarks

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.
- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application’s root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Usage**: The XMod Pro Delete controls work in conjunction with the `<DeleteCommand>` tag of the `<xmod:template>` tag. Typically, the delete command will include one or more `<parameter>` tags that identify which record(s) should be deleted. The delete control should use the same parameter names and fill them with valid values, typically from the current record. That's why delete controls are typically found in `<ItemTemplate>` and `<AlternatingItemTemplate>` tags.
Example

```html
<div>
<table width="100%">
  <tr>
    <td width="250" valign="top">
      <!-- EMPLOYEES TEMPLATE -->

      <xmod:template id="Employees">
        <listdatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
          <parameter name="DepartmentId" alias="DepartmentId"/>
        </listdatasource>
        <deletecommand commandtext="DELETE FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
          <parameter name="EmployeeId" alias="EmpID" />
        </deletecommand>
        <headertemplate>
          <p>Employees</p>
        </headertemplate>
        <itemtemplate>
          <div style="text-align: middle;">
            <strong>
              [[FirstName]] [[LastName]]
            </strong>
            <xmod:deleteimage alternatetext="Delete Employee" onclientclick="return confirm('Are you sure you want to delete this employee?');" imageurl="/images/delete.gif">
              <parameter name="EmployeeId" value='[[EmployeeId]]' />
            </xmod:deleteimage>
          </div>
        </itemtemplate>
      </xmod:template>
    </td>
  </tr>
</table>
</div>
```

```
Syntax Remarks Example

The DeleteLink tag renders as a clickable image at run-time. It is used to execute the <DeleteCommand> of its parent <xmod:template>.
```
### Syntax

```xml
<xmod:DeleteLink
    BackColor="color name|#dddddd"
   BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    OnClientClick="javascript"
    Style="string"
    Text="string"
    ToolTip="String"
    Visible="True|False"
    Width="size">
    ...
    <Parameter Name="string" Value="string" />
    <Parameter Name="string" Value="string" />
    ...additional parameters as needed ...
</xmod:DeleteLink>
```

### Remarks

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.

- **Usage**: The XMod Pro Delete controls work in conjunction with the `<DeleteCommand>` tag of the `<xmod:template>` tag. Typically, the delete command will include one or more `<parameter>` tags that identify which record(s) should be deleted. The delete control should use the same parameter names and fill them with valid values, typically from the current record. That's why delete controls are typically found in `<ItemTemplate>` and `<AlternatingItemTemplate>` tags.
Example

The DetailButton tag renders as a push-button at run-time. It is used to retrieve the `<DetailDataSource>` of its parent `<xmod:template>` and displays that data in the `<DetailTemplate>` defined in the parent `<xmod:template>` tag.
Syntax

```xml
<xmod:DetailButton
   BackColor="color name|#dddddd"
   BorderColor="color name|#dddddd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
   BorderWidth="size"
   CssClass="string"
   Font-Bold="True|False"
   Font-Italic="True|False"
   Font-Names="string"
   Font-Overline="True|False"
   Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
   Font-Strikeout="True|False"
   ForeColor="color name|#dddddd"
   Height="size"
   OnClientClick="javascript"
   Style="string"
   Text="string"
   ToolTip="string"
   Visible="True|False"
   Width="size">
   <Parameter Name="string" Value="string" />
   <Parameter Name="string" Value="string" />
   ...additional parameters as needed ...
</xmod:DetailButton>
```

Remarks

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Usage**: The XMod Pro Detail controls work in conjunction with the `<DetailDataSource>` tag of the `<xmod:template>` tag. Typically, the detail command will include one or more `<parameter>` tags that identify which record should be retrieved. The detail control should use the same parameter names and fill them with valid values, typically from the current record. That’s why detail controls are typically found in `<ItemTemplate>` and `<AlternatingItemTemplate>` tags.
Example

```xml
<example>
<tbl width="100%">
  <tr>
    <td width="250" valign="top">
    <!-- EMPLOYEES TEMPLATE -->
    
    <xmod:template id="Employees">
      <listdatasource commandtext="SELECT EmployeeId, FirstName, LastName FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
        <parameter name="DepartmentId" alias="DepartmentId"/>
      </listdatasource>
      
      <detaildatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
        <parameter name="EmployeeId" alias="EmpID"/>
      </detaildatasource>
      
      <headertemplate>
        <p>Employees</p>
      </headertemplate>
      
      <itemtemplate>
        <div style="text-align: middle;">
          <strong>[[FirstName]] [[LastName]]</strong>
          <xmod:detailbutton text="View Profile">
            <parameter name="EmployeeId" value='[[EmployeeId]]' />
          </xmod:detailbutton>
        </div>
      </itemtemplate>
      
      <detailtemplate>
        <h1>Employee Profile</h1>
        <h3>[[FirstName]] [[LastName]]</h3>
        <h4>Biography:</h4>
        <div>[[Bio]]</div>
      </detailtemplate>
    </xmod:template>
  </td>
</tr>
</tbl>
</example>
```

<back to top>

Syntax Remarks Example

The DetailImage tag renders as a clickable image at run-time. It is used to execute the `<DetailDataSource>` of its parent `<xmod:template>` and displays that data in the `<DetailTemplate>` defined in the parent `<xmod:template>` tag.
### Syntax

```xml
<xmod:DetailImage
    AlternateText="string"
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    ImageAlign="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbsBottom|AbsMiddle|TextTop"
    ImageUrl="url"
    OnClientClick="javascript"
    Style="string"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size">
    <Parameter Name="string" Value="string" />
  <Parameter Name="string" Value="string" />
  ...additional parameters as needed ...
</xmod:DetailImage>
```

### Remarks

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.
- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application's root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Usage**: The XMod Pro Detail controls work in conjunction with the `<DetailDataSource>` tag of the `<xmod:template>` tag. Typically, the detail command will include one or more `<parameter>` tags that identify which record should be retrieved. The detail control should use the same parameter names and fill them with valid values, typically from the current record. That's why detail controls are typically found in `<ItemTemplate>` and `<AlternatingItemTemplate>` tags.
**Example**

```xml
<div>
<table width="100%">
  <tr>
    <td width="250" valign="top">
      <!-- EMPLOYEES TEMPLATE -->

      <xmod:template id="Employees">
        <listdatasource commandtext="SELECT EmployeeId, FirstName, LastName FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
          <parameter name="DepartmentId" alias="DepartmentId"/>
        </listdatasource>

        <detaildatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
          <parameter name="EmployeeId" alias="EmpID"/>
        </detaildatasource>

        <headertemplate>
          <p>Employees</p>
        </headertemplate>

        <itemtemplate>
          <div style="text-align: middle;">
            <strong>[[FirstName]] [[LastName]]</strong>
            <xmod:detailimage alternatetext="View Profile" imageurl="~/images/person.gif">
              <parameter name="EmployeeId" value='[[EmployeeId]]'/>
            </xmod:detailimage>
          </div>
        </itemtemplate>

        <detailtemplate>
          <h1>Employee Profile</h1>
          <h3>[[FirstName]] [[LastName]]</h3>
          <p>Biography:</p>
          <div>[[Bio]]</div>
        </detailtemplate>
      </xmod:template>
    </td>
  </tr>
</table>
</div>
```

**Back to top**

**<xmod:DetailLink>**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Remarks</th>
<th>Example</th>
</tr>
</thead>
</table>

The DetailLink tag renders as a clickable image at run-time. It is used to execute the `<DetailDataSource>` of its parent `<xmod:template>` and displays data in the `<DetailTemplate>` defined in the parent `<xmod:template>` tag.
Syntax

```xml
<xmod:DetailLink
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
   ForeColor="color name|#dddddd"
    Height="size"
    OnClientClick="javascript"
    Style="string"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size">
    <Parameter Name="string" Value="string" />
    <Parameter Name="string" Value="string" />
    ...additional parameters as needed ...
</xmod:DetailLink>
```

Remarks

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.

- **Usage**: The XMod Pro Detail controls work in conjunction with the `<DetailDataSource>` tag of the `<xmod:template>` tag. Typically, the detail command will include one or more `<parameter>` tags that identify which record should be retrieved. The detail control should use the same parameter names and fill them with valid values, typically from the current record. That's why detail controls are typically found in `<ItemTemplate>` and `<AlternatingItemTemplate>` tags.
Example

<example>
<table width="100%">
  <tr>
    <td width="250" valign="top">
      <!-- EMPLOYEES TEMPLATE -->
      <xmod:template id="Employees">
        <listdatasource commandtext="SELECT EmployeeId, FirstName, LastName FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
          <parameter name="DepartmentId" alias="DepartmentId"/>
        </listdatasource>
        <detaildatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
          <parameter name="EmployeeId" alias="EmpID" />
        </detaildatasource>
        <headertemplate>
          <p>Employees</p>
        </headertemplate>
        <itemtemplate>
          <div style="text-align: middle;">
            <strong>[[FirstName]] [[LastName]]</strong>
            <xmod:detaillink text="View Profile">
              <parameter name="EmployeeId" value='[[EmployeeId]]' />
            </xmod:detaillink>
          </div>
        </itemtemplate>
        <detailtemplate>
          <h1>Employee Profile</h1>
          <h3>[[FirstName]] [[LastName]]</h3>
          <h4>Biography: </h4>
          <div>[[Bio]]</div>
        </detailtemplate>
      </xmod:template>
    </td>
  </tr>
</table>
</example>

Syntax Remarks Example

The EditButton tag renders as a push-button at run-time. It is used to show the form defined by the <editform> tag in the module instance's selected form.
Syntax

```xml
<xmod:EditButton
    BackColor="color name|#ddddd"
    BorderColor="color name|#ddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderRadius="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="String|Smaller|Large|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    ForeColor="color name|#ddddd"
    Height="size"
    OnClientClick="javascript"
    Style="string"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size"
/>
```

```xml
<Parameter Name="string" Value="string" />
<Parameter Name="string" Value="string" />
...additional parameters as needed ...
</xmod:EditButton>

Remarks

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns *false* the control will not perform its normal processing. If you return true then the control will perform its normal processing.

Back to top
Example

<!-- EMPLOYEES TEMPLATE -->

<xmod:template id="Employees">
	<headertemplate>
		<p>Employees</p>
	</headertemplate>
	<itemtemplate>
		<div style="text-align: middle;">
			<strong>[[FirstName]] [[LastName]]</strong>
			<xmod:editbutton text="Edit Employee">
				<parameter name="EmployeeId" value='[[EmployeeId]]' />
			</xmod:editbutton>
		</div>
	</itemtemplate>
</xmod:template>

...<EditForm>
	<SelectCommand CommandText="SELECT * FROM XMMDemo_Employees WHERE EmployeeId=@EmployeeId"/>
...
</EditForm>

Syntax Remarks Example

The EditImage tag renders as a clickable image at run-time. It is used to show the form defined by the <editform> tag in the module instance's selected form.
Syntax

```xml
<xm:<xmod:EditImage
    AlternateText="string"
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    ImageAlign="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbaBottom|AbaMiddle|TextTop"
   ImageUrl="url"
    OnClientClick="javascript"
    Style="string"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size"

    <Parameter Name="string" Value="string" />
    <Parameter Name="string" Value="string" />
    ...additional parameters as needed ...

</xm:<xmod:EditImage>
```

Remarks

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.
- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application's root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return `true` then the control will perform its normal processing.
Example

```xml
<div>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <!-
        -- EMPLOYEES TEMPLATE -->
        <xmod:template id="Employees">
          <listdatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
            <parameter name="DepartmentId" alias="DepartmentId"/>
          </listdatasource>
          <headertemplate>
            <p>Employees</p>
          </headertemplate>
          <itemtemplate>
            <div style="text-align: middle;">
              <strong>
                [[FirstName]] [[LastName]]
              </strong>
              <editimage alternatetext="Edit Employee" imageurl="~/images/edit.gif">
                <parameter name="EmployeeId" value='[[EmployeeId]]' />
              </editimage>
            </div>
          </itemtemplate>
        </xmod:template>
      </td>
    </tr>
  </table>
</div>
```

---

...<br />

```xml
<EditForm>
  <SelectCommand CommandText="SELECT * FROM XMPDemo_Employees WHERE EmployeeId=@EmployeeId"/>
</EditForm>
```

Back to top

<xmod:EditLink>

**Syntax**

Remarks

Example

The EditLink tag renders as a clickable image at run-time. It is used to show the form defined by the `<editform>` tag in the module instance's selected form.
**Syntax**

```xml
<%mod:EditLink
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  FontStyle="Bold=True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="String|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  OnClientClick="javascript"
  Style="string"
  Text="string"
  ToolTip="string"
  Visible="True|False"
  Width="size">

  <Parameter Name="string" Value="string" />
  <Parameter Name="string" Value="string" />

  ...additional parameters as needed ...

</%mod:EditLink>
```

**Remarks**

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return `true` then the control will perform its normal processing.

[Back to top]
### Example

```xml
<root>
    <table width="100%">
        <tr>
            <td width="250" valign="top">
                <!-- EMPLOYEES TEMPLATE -->

                <xmod:template id="Employees">
                    <listdatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
                        <parameter name="DepartmentId" alias="DepartmentId"/>
                    </listdatasource>
                    <headertemplate>
                        <p>Employees</p>
                    </headertemplate>
                    <itemtemplate>
                        <div style="text-align: middle;">
                            <strong>
                                [[FirstName]] [[LastName]]
                            </strong>
                            <xmod:editlink text="Edit Employee">
                                <parameter name="EmployeeId" value='[[EmployeeId]]' />
                            </xmod:editlink>
                        </div>
                    </itemtemplate>
                </xmod:template>
            </td>
        </tr>
    </table>
</root>
```

---

### Command Syntax

```xml
<Command CommandText="SELECT * FROM XMPDemo_Employees WHERE EmployeeId=@EmployeeId"/>
```

### Syntax

The Feed tag, first introduced in version 3.0, is used to define the output of feeds. A feed typically is thought of as an RSS feed, but it can be much more than that. With the Feed tag, you can produce virtually any XML-based output, a printer-friendly HTML page, a plain text page, even a comma-delimited file that can be automatically opened by Excel or a similar program.

### Creating A Feed

Creating a feed is a two-step process:

1. Create your feed on the Manage Feeds page in the Control Panel. The Feed is very similar to XMod Pro's Template tag and is created in a similar fashion. You begin by clicking the "New Feed" button. On the dialog which pops-up, choose the type of feed you want to create, select a data source, and click the dialog's "New Feed" button to create it. From there, you can edit your feed in the editor.

The Feed Tag is defined much the same way as the Template tag. You define your data source in the ListDataSource tag. You use the Header, ItemTemplate, AlternatingItemTemplate, and Footer tags to format the display of your data.

**Helpful Formatting Tip:** If you are creating a CSV (comma-separated values) file or a file with similar plain text
formatting, you will probably need to pay attention to the spacing of the tags in your feed. Unlike HTML, extra whitespace and linefeeds will show up in a plain text file.

2. Call your feed. You can get an example of how to call your feed by clicking the "How To Call Your Feed" link on the Manage Feeds page. Basically, you link to it like you would any other page. You'll call XMod Pro's "Feed.aspx" page and pass it a few required parameters and any additional parameters your data source may need to retrieve the data.

Use a URL in the form of:

```
http://sitename/DesktopModules/XModPro/Feed.aspx?xfd=FeedName&pid=PortalId
```

Where sitename is your site's domain, FeedName is the name of the feed as listed in the grid, and PortalId is the numeric ID of the portal for which the feed is created.

A Note About Security

It's important to note that feeds, by their nature, are typically intended to be public. No special security measures have been implemented for securing the feed. So you should be careful about what data you make available through a feed. One way you could make your feed more secure is by creating and passing a token to the feed. This token would then be checked by your data source (most likely a stored procedure) to determine its validity, returning the data only if it's valid.

Syntax

```
<xm:feed
   Doctype="string"
   ContentType="string"
   ConnectionString="string"
   Filename="string"
   ViewRoles="role1,role2,role3">
   
   <ListDataSource CommandText="string"
                    ConnectionString="string"/>
   
   <HeaderTemplate>...</HeaderTemplate>
   <ItemTemplate>...</ItemTemplate>
   <AlternatingItemTemplate>...</AlternatingItemTemplate>
   <SeparatorTemplate>...</SeparatorTemplate>
   <FooterTemplate>...</FooterTemplate>

</xm:feed>
```

Remarks

- **Doctype**: (Optional) This property allows you to specify the DOCTYPE for the resulting page. Consumers of the feed may use the DOCTYPE to determine how to render the feed.
- **ContentType**: Defaults to "text/html". Helps the feed consumer determine the nature of the content. This is often referred to as MIME types. Other examples are: "text/plain", "text/xml", "text/javascript", "application/vnd.ms-excel", etc.
- **ConnectionString**: If you need to specify a SQL Server database other than the current DotNetNuke database, you can supply a connection string here. When specified, it serves as the default connection string for the ListDataSource and DetailDataSource. You can also use a connection string defined in the web.config file. To do so, use the ConnectionString token like so:

```
<xmod:feed ... ConnectionString='[[ConnectionString:connectionName]]' />
```

- **Filename**: (Optional) This property can be set to give the browser a filename for downloading purposes. It's a great way to provide your users with an export file. For example, by giving your feed a file name of "MyFile.csv" and setting the ContentType to "application/vnd.ms-excel" you can have the file opened by Microsoft Excel (assuming Excel is installed on the
Data Sources: The data for each feed is a SQL command. This could be a SQL SELECT command or a stored procedure. Supply the SQL command as the CommandText attribute. For stored procedures, use EXEC sprocName. Note that Bit columns are returned as True/False - not 1/0. The data source for a feed is defined in the <ListDataSource> tag.

In addition to its attributes, the Feed tag also contains numerous child tags which define most of the tag's functionality:

- **ListDataSource**: Provides the data for the template when it is displaying in list view. It accepts <parameter> tags to pass parameters as part of the command. The "alias" attribute is optional and is used to avoid conflicts or to accept a parameter with a specific name (the "name" attribute) but use a different parameter name in the command (the "alias" attribute). If no "alias" is specified, the "name" will be used.

Examples:

```xml
#1
<ListDataSource CommandText="SELECT FirstName, LastName FROM Users" />

#2
<ListDataSource CommandText="SELECT FirstName, LastName FROM Users
WHERE ZipCode = @zip">
  <parameter name="zip" alias="zip" value="12345" />
</ListDataSource>

#3
<ListDataSource CommandText="EXEC GetUsers @zip">
  <parameter name="zip" alias="zip" value="12345" />
</ListDataSource>
```

You can optionally specify a **ConnectionString** for this data source. If none is specified, the default connection will be used - either the connection specified by the ConnectionString property of the parent <xmod:feed> tag or the current DotNetNuke database if the tag doesn't define a connection. As with the template tag, you can use the {{ConnectionString:connectionName}} token to use a connection defined in the web.config file.

- **HeaderTemplate**: When displaying a list view, HTML, text, and controls that will be rendered once, at the beginning of the list. This template is optional.

- **ItemTemplate**: When displaying a list view, HTML, text, field tokens, and controls that will be rendered for each record in the result set. If the AlternatingItemTemplate is supplied, then the ItemTemplate will be used for all odd-numbered records.

- **AlternatingItemTemplate**: When displaying a list view, HTML, text, field tokens, and controls that will be rendered for the even-numbered records in the result set. This template is optional.

- **SeparatorTemplate**: When displaying a list view, this will render any HTML or text that you specify between each record. One good example is if you're creating a list of comma-separated records (we'll use numbers for simplicity). If you put your comma in the ItemTemplate, you would get a trailing comma like this: 1,2,3. But if you put your comma in the SeparatorTemplate, you won't get that trailing comma: 1,2,3. (Added in version 1.4)

- **FooterTemplate**: When displaying a list view, HTML, text, and controls that will be rendered once, at the end of the list. This template is optional.

- **Field Tokens**: In order to display data from your data source, XMod Pro uses "field tokens". These are essentially placeholders that contain the name of the column or field in your data source. At run-time, these tokens are replaced with the value from the record. Field Tokens are written in this form: {{FieldName}}. "FieldName" is the name of the field or column in your data source. It must be surrounded by [[] and ]]. You can use the Field Token in many places. However, when you use
it as the attribute value for an XMod Pro tag or other third party control, you must delimit the attribute value with single quotes, rather than double quotes. For example, when used in the <parameter> tag, the attribute is written as: value='[[UserID]]' rather than: value="[[UserID]]". When using a Field Token with HTML tag attributes, this is not necessary.

- **ViewRoles**: New to version 4.0. Takes a comma-delimited list of DNN security roles. If the user is logged in and is a member of one of the roles, or is a Host/SuperUser or Admin, the user will be able to see the feed. If the user is not logged-in or is not in one of the roles, he or she will not be able to see the feed. If ViewRoles is empty or is not specified, all users will be able to see the feed.

---

**Example**

```xml
<modx:Feed Content-Type="application/vnd.ms-excel" Filename="test.csv">
<ListDataSource CommandText="SELECT [AuthorId], [FirstName], [LastName], [GenreId] FROM Authors"/>
<HeaderTemplate>Author Id, First Name, Last Name, Genre Id</HeaderTemplate>
<ItemTemplate>{{[AuthorId]},{{[FirstName]},[[LastName]],[[GenreId]]}}</ItemTemplate>
</modx:Feed>
```

---

**<modx:Format>**

---

**Syntax Remarks Example**

The Format tag allows you to present your data in a more user-friendly format. With it, you can format currency, numbers, dates. You can perform text substitutions and regular expression substitutions. It also provides you with the ability to cloak text - i.e. obfuscate it so that web 'bots have more difficulty scraping your web pages for data like email addresses.

**Syntax**

```xml
<modx:Format
  Type="Numeric|Float|Date|Text|RegEx|Cloak|HtmlEncode|HtmlDecode|UrlEncode|UrlDecode"
  Value="string"
  Pattern="string"
  Replacement="string"
  MaxLength="integer"
  InputCulture="locale ID"
  OutputCulture="locale ID"
/>
```

**Remarks**

The Format tag is used to operate on data - usually from form fields, but it can also operate on hard-coded values. It gives you the ability to take a value and adjust its appearance. For instance the value "1" could be formatted as "01", "1", or "1.00". The value "2005-05-21" could be formatted like "05/21/2005", "Sat May 21 2005", etc.

The Format tag will format values as numbers (no floating point), floating-point numbers, and dates. It will also allow you to perform text substitutions as well as regular expression substitutions. Format tags are empty tags, meaning that they do not contain any inner text. Please follow the XHTML syntax for empty tags which is to write them as a single tag. (i.e. <modx:format ... /> See example)

- **Type**: This determines how it should treat the value it will be formatting. For numbers and dates, the input value must be convertible to the type you specify. Valid values are below:
  - **Numeric**: the input value will be treated as a whole number (1, 100, -1, 1000, etc.).
  - **Float**: the input value will be treated as a floating point number (1.01, -0.315, etc.)
  - **Date**: the input will be treated as a date/time value (5/21/05, 2005-05-21 1:00 pm, etc.)
  - **Text**: the input is treated as text. This mode allows you to perform replacements in that text by using placeholders ({0}, {1}, etc.) and a list of replacement values. The first item in the list would replace {0}, the second would replace {1}, and so on. (see examples for more info)
- **RegEx**: the input is treated as text. This mode allows you to put your regular expression skills to work to perform magical feats of text manipulation. In the pattern attribute, you would place the regular expression pattern for matching. In the replacement attribute, you would place your regular expression replacement pattern (see examples for more info).

- **Cloak**: This type of formatting can be used for potentially sensitive information that you want to hide from robots and spiders which crawl the web looking to harvest information. The primary use for this is to obfuscate email addresses but other data can be used as well. When using the cloak format type, you only need to supply a value. All other attributes are ignored.

NOTE: Cloaking uses Javascript to obscure the data within the HTML source code. If Javascript is enabled in the browser, the text will display. If Javascript is disabled, the text will not display. While this is effective at hiding data like email addresses from spam bots, no solution is foolproof. Additionally, this method cannot be used to supply data to a "mailto" hyperlink because of the Javascript involved.

- **HtmlEncode/Decode**: the input value will have its HTML encoded (or decoded) before rendering out to the page.

- **UrlEncode/Decode**: the input value will be encoded for URL’s (or decoded from a URL-encoded value).

- **Value**: This is the input value that you intend to format. It can be hard-coded - i.e. a value you type in, or it can be a field token representing the value of a field. NOTE: Sometimes the value you're dealing with may contain double and/or single quotes. In many cases this may prevent the parser from correctly recognizing the correct value of the "value" attribute. In these cases, you can place the value inside the opening and closing tags. See the examples section for details.

- **Pattern**: This is the pattern XMod will use to perform the formatting. What you place here will depend on what type you have specified. Some examples are listed below.
  - **Numeric**: "0", "#0", "00", "0.00", "c" (currency), "g" (general number format). On a US system, the results for the input value "1" would be: "1", "1", "01", "1.00", "$1.00", "1"
  - **Date**: "MM/dd/yyyy" (01/31/2005), "ddd MMM dd yyyy" (Wed Apr 13 2005)
  - **Text**: For text replacements the "pattern" (\{0\}, \{1\}, etc.) is embedded in the input value, so no pattern is specified.

- **RegEx**: The input value may be: "Hello NAME, How's the CONDITION" In this case your pattern may be: "\((.*)(NAME)(.*)(CONDITION)\)" and your replacement value might be "$1John$3weather" with the result being: "Hello John, How's the weather".

- **Cloak**: This attribute is not used.

- **Replacement**: The replacement is used for text and regex formatting types. What you place in this attribute depends on which formatting type you're using:
  - **Text**: For text replacements, you would supply a comma-delimited list of values to use for the replacements. Their order will determine which placeholder (i.e. \{0\}, \{1\}, etc.) they replace. For example, if the input value is "Hello \{0\}, How's the \{1\}?" your replacement might be "John,weather" with the result being "Hello John, How's the weather?"
  - **RegEx**: For regular expression-based formatting, you would specify a pattern to match in the pattern attribute and use the captured items in your replacement. For example: The input value may be: "\"Hello NAME, How's the CONDITION\" In this case your pattern may be: "\((.*)(NAME)(.*)(CONDITION)\)" and your replacement value might be "$1John$3weather" with the result being: "Hello John, How's the weather". Notice there are 4 captures in the pattern: the first and third match any characters. The second and fourth match "NAME" and "CONDITION" respectively. In your replacement, the $1 stands for the first capture, $3 stands for the 3rd capture.

- **MaxLength**: If this value is 1 or greater, the format tag will truncate the resulting text and add an ellipsis (...). The total number of characters (including the ellipsis) will not exceed the number of characters specified. This attribute is ignored if the Format Type is Cloak because truncating the results could prevent the cloaked text from displaying correctly.

- **InputCulture/OutputCulture**: If the input value is in a culture-specific format, you can specify what culture should be used when trying to convert that value. Dates, for instance, vary widely based on culture. There are many pre-defined culture names (known as LCID's or locale ID's) which are valid values such as: en-US (English - United States), en-GB (English - United Kingdom), fr-FR (French - France), de-DE (German - Germany), es-ES (Spanish - Spain), es-MX (Spanish - Mexico), etc.
  - **InputCulture** is used to tell the tag what culture to assume when processing the incoming value (the value attribute).
  - **OutputCulture** is used to tell the tag what culture to assume when performing the formatting.
For example, the input value may be a US date, but you want to format it for display to a French audience. In that case, you would specify "en-US" as the InputCulture and "fr-FR" as the OutputCulture. When used this way, you can avoid using the pattern attribute altogether and rely on the default formatting for the respective cultures. See the "date" examples below for an example of this.

Example

```xml
<xmod:template ...>

...<itemtemplate>

EXAMPLE #1
If Quantity=5 Output Value Will Be: 05.00
<xmod:format type="numeric" value='[[Quantity]]' pattern="0#.00")

EXAMPLE #2
If Quantity=5 Output Value Will Be: 05
<xmod:format type="numeric" value='[[Quantity]]' pattern="d2")
NOTES: The "d" pattern is a pre-defined pattern for working with numeric values. It works only with whole numbers. It instructs the tag to display the value as a decimal (base 10). The "2" instructs the tag to display the number as a 2 digit number.

EXAMPLE #3
If Quantity=5
   Output On US Systems:$5.00
   Output On United Kingdom Systems:£5.00
<xmod:format type="numeric" value='[[Quantity]]' pattern="c")
NOTES: No output culture is specified so the output defaults to the web server's culture. The "c" pattern stands for formatting the value as currency.

EXAMPLE #4
If Quantity=5
   Output On US Systems:£5.00
   Output On United Kingdom Systems:£5.00
<xmod:format type="numeric" value='[[Quantity]]' pattern="d2"
   outputculture="en-GB")
NOTES: United Kingdom English is specified as the output culture, effectively forcing the value to be formatted with the £ symbol, even on US systems.

</itemtemplate>
...
</xmod:template>
```

<Back to top>

<lxml>
Syntax

```xml
<xm:IfEmpty
    Value="string">
    ...Content Goes Here...
</xm:IfEmpty>
```

Remarks

- **Value**: This can be a hard-coded string value theoretically, however, the main purpose is to accept a Field Token. NOTE, when using a Field Token, you must use single quotes rather than double quotes to delimit it:

  ```xml
  <xm:IfEmpty Value='[[MyField]]'>...</xm:IfEmpty>
  ```

Example

```xml
<verbatim>
<verbatim>
</verbatim>
```

(New to version 4.2) The `IfNotEmpty` tag is a quick way to display content only if there is a value in a column. In other words if the value is not an empty string ("") or a Null value, then the content of the `IfNotEmpty` tag will be rendered out to the page.
Syntax

```xml
<xmod:IfNotEmpty
  Value="string">
  ...Content Goes Here...
</xmod:IfNotEmpty>
```

Remarks

- **Value**: This can be a hard-coded string value theoretically, however, the main purpose is to accept a Field Token. NOTE, when using a Field Token, you must use single quotes rather than double quotes to delimit it:

```xml
<xmod:IfNotEmpty Value='[[MyField]]'>...</xmod:IfNotEmpty>
```

Example

```xml
<div>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <!-- EMPLOYEES TEMPLATE -->

        <xmod:Template Id="Employees">
          <DetailDataSource CommandText="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
            <Parameter Name="EmployeeId" Alias="EmpID" />
          </DetailDataSource>

          <DetailTemplate>
            <h1>Employee Profile</h1>
            <h3>[[FirstName]] [[LastName]]</h3>
            <xmod:IfEmpty Value='[[imageUrl]]'>
              <img src="/images/NoImage.png" />
            </xmod:IfEmpty>
            <xmod:IfNotEmpty Value='[[imageUrl]]'>
              <img src="[[imageUrl]]" />
            </xmod:IfNotEmpty>
            <h4>Biography:</h4>
            <div>[[Bio]]</div>
            <xmod:MetaTags>
              <Title>Employee Profile for [[FirstName]] [[LastName]]</Title>
              <Keywords append="true">[[FirstName]], [[LastName]]</Keywords>
            </xmod:MetaTags>
          </DetailTemplate>
        </xmod:Template>
      </td>
    </tr>
  </table>
</div>
```

The jQueryReady tag is a quick and easy way to embed a jQuery "ready" event in the page. This tag requires jQuery be included in the page.
Syntax

```xml
<xmod:jQueryReady>
  {[jQuery and/or Javascript script]}<xmod:jQueryReady>
</xmod:jQueryReady>
```

Remarks

- **Usage**: Use this tag to quickly insert Javascript and/or jQuery code that should only be run after the document's DOM has been loaded. The jQuery(document).ready() function is standard fare when working with jQuery. This tag allows you to forget about the boilerplate script and focus on your script. The tag will place your script near the bottom of the page - standard practice for improving page load times. Importantly, the tag automatically creates a "closure" for your script allowing you to use the "$" shortcut instead of "jQuery" in your code. Use of the closure also helps ensure your script operates in its own scope and will be not be impacted by other Javascript on the page. This tag requires jQuery be included in the page.

Example

In the example below, we've set the DIV tag containing the Employee's biography to initially be hidden (style="display:none;"). Then, we used the jQueryReady tag to attach some code to the H4 tag's ("Biography" header) click event. It simply makes the biography DIV tag visible.

```xml
<div>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <!- EMPLOYEES TEMPLATE -->
        <xmod:template id="Employees">
          <detaildatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
            <parameter name="EmployeeId" alias="EmpID" />
          </detaildatasource>
          <detailtemplate>
            <h1>Employee Profile</h1>
            <h3>[[FirstName]] [[LastName]]</h3>
            <h4 class="bio">Biography:</h4>
            <div style="display:none;">[[Bio]]</div>
          </detailtemplate>
        </xmod:template>
      </td>
    </tr>
  </table>
  <xmod:jQueryReady>
    $('h4.bio').click(function(){
      $(this).next().show();
    });
  </xmod:jQueryReady>
</div>
```

The LoadFeed tag makes it a cinch to dynamically load XMod Pro feeds into your page. This tag requires jQuery be included in the page. The LoadFeed tag will grab the feed data when the page is loaded. Depending on the complexity of your page, it is possible the HTML element which will house the results may not be rendered yet. It is good practice to place the load feed tags after the HTML elements in the page if this is an issue. To allow the user to trigger the loading of a feed see: <xmod:LoadFeedButton>, <xmod:LoadFeedImage>, and <xmod:LoadFeedLink>. 
Syntax

```xml
<xmod:LoadFeed
    FeedName="string"
    LoadingCssClass="CSS Class Name(s)"
    LoadingImageUrl="url"
    Target="jQuery element selector">
    [one or more optional Field tags can be used to pass data to the feed]
    <Field Name="string" Value="string" />
</xmod:LoadFeed>
```

Remarks

- **FeedName**: The name you have given your feed on the Manage Feeds page. Note that you should set the feed's ContentType to "text/html" (see examples below).

- **LoadingCssClass**: One or more CSS class names (separated by a space) that will be applied to the HTML image tag used to display a "loading" status to the user. This property is ignored if no LoadingImageUrl has been specified.

- **LoadingImageUrl**: The path to the image you'd like to display as a "loading" status indicator to your users. Typically this will be an animated GIF or similar file. You may use the tilde (~) character as a short-hand for the site's root directory.

- **Target**: A jQuery selector identifying the element whose content will be replaced by the results of the feed.

- **Field Tags**: If you need to pass additional information to your feed, use the child Field tags. The Field's Name property should be the name of the parameter your feed is looking for and the Value of that Field will be the value you pass in for the parameter.

Example

This example shows that you don't even need `<xmod:Template>` tags in your template. The LoadFeed tags will load their data when the page is loaded. It creates to LoadFeed tags at the top of the template and points them to two DIV tags where the results of the feeds will be placed. The two feeds are shown after the template example.

```xml
<!-- TOP AUTHORS -->
<div id="TopAuthors"/>

<!-- TOP CRIME BOOKS -->
<div id="TopCrime"/>

<!-- LOAD FEED TAGS -->
<xmod:LoadFeed FeedName="Top20Authors" Target="#TopAuthors" LoadingImageUrl="~/images/loading.gif" />
<xmod:LoadFeed FeedName="Top20CrimeBooks" Target="#TopCrime">
    <Field Name="GenreId" Value="20" />
</xmod:LoadFeed>
```
**Example Feeds -- Top20Authors**

```xml
<xs:Feed ContentType="text/html">
  <ListDataSource CommandText="SELECT FirstName, LastName, AuthorId, SalesRank FROM Authors WHERE SalesRank <= 20" />
  <HeaderTemplate>
    <table>
      <thead>
        <tr>
          <th>Rank</th>
          <th>Author</th>
        </tr>
      </thead>
      <tbody>
      </tbody>
    </table>
  </HeaderTemplate>
  <ItemTemplate>
    <tr>
      <td>[[SalesRank]]</td>
      <td>[[FirstName]] [[LastName]]</td>
    </tr>
  </ItemTemplate>
  <FooterTemplate>
  </FooterTemplate>
</xs:Feed>
```

**Example Feeds -- Top20CrimeBooks**

```xml
<xs:Feed ContentType="text/html">
  <ListDataSource CommandText="SELECT Title, SalesRank FROM Books WHERE GenereId = @GenreId" />
  <Parameter Name="GenreId" Value='[[Url:GenreId]]' />
  <HeaderTemplate>
    <table>
      <thead>
        <tr>
          <th>Rank</th>
          <th>Title</th>
        </tr>
      </thead>
      <tbody>
      </tbody>
    </table>
  </HeaderTemplate>
  <ItemTemplate>
    <tr>
      <td>[[SalesRank]]</td>
      <td>[[Title]]</td>
    </tr>
  </ItemTemplate>
  <FooterTemplate>
  </FooterTemplate>
</xs:Feed>
```
The LoadFeedButton tag makes it a cinch to dynamically load XMod Pro feeds into your page. This tag requires jQuery be included in the page. The LoadFeedButton tag will grab the feed data when the user clicks the button.

**Syntax**

```xml
<xmod:LoadFeedButton
    FeedName="string"
    Height="size"
    IDSelector="jQuery element selector"
    InfinitePaging="True|False"
    InsertMode="Replace|Append|Prepend"
    LoadingCssClass="CSS Class Name(s)"
    LoadingImageUrl="url"
    Target="jQuery element selector"
    Text="string"
    Width="size">
    [one or more optional Field tags can be used to pass data to the feed]
    <Field Name="string" Value="string" />
</xmod:LoadFeedButton>
```

**Remarks**

- **FeedName**: The name you have given your feed on the Manage Feeds page. Note that you should set the feed's ContentType to "text/html" (see examples below).

- **IDSelector**: Used if InfinitePaging is True, this is a jQuery Selector that locates the element that contains the ID for each record. XMod Pro will look at the inner text of the element. If you don't want the ID to appear to be visible, we recommend inserting a `<span>` tag that has its display set to none like so:
  ```html
  <table id="AuthorsTable">
    <tr>
      <td>[[FirstName]] [[LastName]]<span style="display:none;">[[AuthorID]]</span></td>
      <td>[[SalesRank]]</td>
    </tr>
  </table>
  ```
  In the above case, if you wanted to append new records to the end of the table, you would set your IDSelector like so:
  ```html
  IDSelector="#AuthorsTable tr:last td:first span"
  ```

- **InfinitePaging**: True or False. False by default. When set to true, XMod Pro will add jQuery and Javascript to the page enabling your button to trigger adding feed results to an existing list. For instance, you can display your initial results in a table and, when the button is clicked, the feeds results can be added to the existing table. You’ll often see this behavior on mobile phones when viewing data from the web. At the bottom of a list of results you’ll see a link like “Show 10 More...”. Clicking that will add 10 more results to the list. Clicking the button again will add 10 more results the list, and so on. The LoadFeedButton makes the front end of this process pretty painless to implement. You’ll just need to pair it with an XMod Pro feed that can return data in the format you expect. See the Example for details.

- **InsertMode**: Set to "Replace" by default. You can also set it to Append or Prepend. When set to replace, the feed's results will replace the content of the target element (see Target property). When set to Append, the results will be added to the end of the target. Likewise, when set to Prepend, it will insert the results before the target. Note, when using Append and Prepend, it is important your XMod Pro feed return results using HTML that works with your page. For instance, if you’re Appending to an existing table, your feed should return table rows (<tr> tags).

- **LoadingCssClass**: One or more CSS class names (separated by a space) that will be applied to the HTML image tag used to display a "loading" status to the user. This property is ignored if no LoadingImageUrl has been specified.
- **LoadingImageUrl**: The path to the image you’d like to display as a "loading" status indicator to your users. Typically this will be an animated GIF or similar file. You may use the tilde (~) character as a short-hand for the site's root directory.

- **Target**: A jQuery selector identifying the element whose content will be replaced by the results of the feed.

- **Text**: The caption that will be displayed on the button.

- **Field Tags**: If you need to pass additional information to your feed, use the child Field tags. The Field's Name property should be the name of the parameter your feed is looking for and the Value of that Field will be the value you pass in for the parameter.

**Example**

This example shows two LoadFeedButton tags. They retrieve different feeds but place them in the same element on the page (the DIV with id of Content). When the "Show Top Authors" is clicked, the Content DIV will display the list of Top 20 authors. If you then click the "Show Top Crime Books" the contents of the DIV tag will be replaced with the top 20 crime books. The two feeds are shown after the template example.

```xml
<!-- Main Content DIV Feed Results Will Be Placed Here -->
<div id="Content"></div>

<!-- LOAD FEED TAGS -->
<xmod:LoadFeedButton Text="Show Top Authors" FeedName="Top20Authors" Target="#Content" LoadingImageUrl="~/images/loading.gif"/>
<xmod:LoadFeedButton Text="Show Top Crime Books" FeedName="Top20CrimeBooks" Target="#Content">
  <Field Name="GenreId" Value="20"/>
</xmod:LoadFeedButton>

**Example Feeds -- Top20Authors**

<xmod:Feed ContentType="text/html">
  <ListDataSource CommandText="SELECT FirstName, LastName, AuthorId, SalesRank FROM Authors WHERE SalesRank <= 20"/>
  <HeaderTemplate>
    <table>
      <thead>
        <tr>
          <th>Rank</th>
          <th>Author</th>
        </tr>
      </thead>
      <tbody>
      </tbody>
    </table>
  </HeaderTemplate>
  <ItemTemplate>
    <tr>
      <td>[[SalesRank]]</td>
      <td>[[FirstName]] [[LastName]]</td>
    </tr>
  </ItemTemplate>
  <FooterTemplate>
  </FooterTemplate>
</xmod:Feed>
```
Example Feeds -- Top20CrimeBooks

<xmod:Feed ContentType="text/html">
  <ListDataSource CommandText="SELECT Title, SalesRank FROM Books WHERE GenereId = @GenreId">
    <Parameter Name="GenreId" Value='[[Url:GenreId]]'/>
  </ListDataSource>
  <HeaderTemplate>
    <table>
      <thead>
        <tr>
          <th>Rank</th>
          <th>Title</th>
        </tr>
      </thead>
      <tbody>
      </tbody>
    </table>
  </HeaderTemplate>
  <ItemTemplate>
    <tr>
      <td>[[SalesRank]]</td>
      <td>[[Title]]</td>
    </tr>
  </ItemTemplate>
  <FooterTemplate>
    </tbody>
    </table>
  </FooterTemplate>
</xmod:Feed>

Example - Infinite Paging

This example shows how you can setup "Infinite Paging". The idea behind infinite paging is that the user is presented with an initial list of, say, 10 results. If there are 100 total records, normally the user would page to the next set of 10 and keep paging to browse through the results. Infinite paging leverages the power of AJAX to present all those results on the same page. So, the user is presented with a button that, when clicked, will load the next set of 10 results and add them to the existing list. So, now the list contains a total of 20 results. If the button is pressed again, 10 more results will be added to the list for a total of 30 results. The two feeds are shown after the template example.

NOTE: Infinite Paging will send your feed a parameter called "LastId" that you can use to determine the next set of values to return. However, Infinite Paging needs you to tell it where it can get the LastId. You do this by setting the IDSelector property. This should be a jQuery selector that points to the element that contains that value. XMod Pro will grab the innerText of that element and use that as the LastId. In the example below, we're storing the Author's ID in a hidden SPAN tag. Since the last-retrieved record id is going to be in the last record we display, we'll find the LastId in the list item.

<xmod:Template>
  <ListDataSource CommandText="SELECT TOP 10 AuthorId, FirstName, LastName FROM Authors"/>
  <HeaderTemplate>
    <ul id="AuthorsList">
    </HeaderTemplate>
  <ItemTemplate>
    <li>[[FirstName]] [[FirstName]]<span style="display:none;">[[AuthorId]]</span></li>
  </ItemTemplate>
  <FooterTemplate>
    </ul>
    <xmod:LoadFeedButton Text="Show Next 10 Authors" FeedName="Authors_Chunked" LoadingImageUrl="/images/loading.gif" Target="#AuthorsList" IDSelector="#AuthorsList li:last span" InsertMode="Append"/>
  </FooterTemplate>
</xmod:Template>
Example Feed -- Infinite Paging, Authors

This feed is designed to work with the template specified above. There are several important items to implement to ensure successful implementation of infinite paging.

1. The feed must have its ContentType set to "text/html" since we're sending HTML back to the template.
2. The ListDataSource should use the URL parameter called "LastId" to determine which record(s) to return.
3. The ListDataSource is set to select the first 10 records that match the WHERE clause criteria. This allows you to return the next 10 records to the template. If your template was set to retrieve the next 20 records, the ListDataSource should SELECT TOP 20 ...
4. The ListDataSource has a <Parameter> tag that grabs the URL parameter called "LastId" that is passed automatically by XMod Pro.
5. Since we are injecting HTML into the template's page, we only need to render out the minimum HTML to add List Items to the unordered list. So, we don't have a HeaderTemplate or FooterTemplate - just an ItemTemplate matches the ItemTemplate in the <xmod:Template> tag above. This is important not only to maintain visual consistency but you must also be sure to include the <span> tag that contains the record's ID.

```xml
<feed type="text/html">
  <list data-source="SELECT TOP 10 FirstName, LastName, AuthorId FROM Authors WHERE AuthorId > @LastId ORDER BY AuthorId ASC">
    <parameter name="LastId" value="[[Url:LastId]]" />
  </list>
  <item>
    <li>[[FirstName]] [[LastName]]<span style="display:none;">[[AuthorId]]</span></li>
  </item>
</feed>
```

Syntax Remarks Example

The LoadFeedImage tag makes it a cinch to dynamically load XMod Pro feeds into your page. This tag requires jQuery be included in the page. The LoadFeedImage tag will grab the feed data when the user clicks the button.

Syntax

```xml
<xmod:LoadFeedImage
  alternatetext="string"
  feedname="string"
  height="size"
  idselector="jQuery element selector"
  loadingcssclass="CSS Class Name(s)"
  loadingimageurl="url"
  target="jQuery element selector"
  text="string"
  width="size">
  [one or more optional Field tags can be used to pass data to the feed]
</xmod:LoadFeedImage>
```

Remarks

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.
- **FeedName**: The name you have given your feed on the Manage Feeds page. Note that you should set the feed's ContentType to "text/html" (see examples below).

- **IDSelector**: Used if InfinitePaging is True, this is a jQuery Selector that locates the element that contains the ID for each record. XMod Pro will look at the inner text of the element. If you don't want the ID to appear to be visible, we recommend inserting a `<span>` tag that has its display set to none like so:

  ```html
  <table id="AuthorsTable">
    <tr>
      <td><span style="display:none;">[[FirstName]] [[LastName]]</span></td>
    </tr>
    <td>[[AuthorID]]</td>
    <td>[[SalesRank]]</td>
  </tr>
</table>
```

In the above case, if you wanted to append new records to the end of the table, you would set your IDSelector like so:

```
IDSelector="#AuthorsTable tr:last td:first span"
```

- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application’s root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.

- **InfinitePaging**: True or False. False by default. When set to true, XMod Pro will add jQuery and Javascript to the page enabling your button to trigger adding feed results to an existing list. For instance, you can display your initial results in a table and, when the button is clicked, the feeds results can be added to the existing table. You’ll often see this behavior on mobile phones when viewing data from the web. At the bottom of a list of results you’ll see a link like “Show 10 More...”. Clicking that will add 10 more results to the list. Clicking the button again will add 10 more results the list, and so on. The LoadFeedImage makes the front end of this process pretty painless to implement. You’ll just need to pair it with an XMod Pro feed that can return data in the format you expect. See the Example for details.

- **InsertMode**: Set to "Replace" by default. You can also set it to Append or Prepend. When set to replace, the feed’s results will replace the content of the target element (see Target property). When set to Append, the results will be added to the end of the target. Likewise, when set to Prepend, it will insert the results before the target. Note, when using Append and Prepend, it is important your XMod Pro feed return results using HTML that works with your page. For instance, if you’re appending to an existing table, your feed should return table rows (`<tr>`) tags.

- **LoadingCssClass**: One or more CSS class names (separated by a space) that will be applied to the HTML image tag used to display a "loading" status to the user. This property is ignored if no LoadingImageUrl has been specified.

- **LoadingImageUrl**: The path to the image you’d like to display as a "loading" status indicator to your users. Typically this will be an animated GIF or similar file. You may use the tilde (~) character as a short-hand for the site’s root directory.

- **Target**: A jQuery selector identifying the element whose content will be replaced by the results of the feed.

- **Text**: The caption that will be displayed on the button.

- **Field Tags**: If you need to pass additional information to your feed, use the child Field tags. The Field's Name property should be the name of the parameter your feed is looking for and the Value of that Field will be the value you pass in for the parameter.

### Example

This example shows two LoadFeedImage tags. They retrieve different feeds but place them in the same element on the page (the DIV with id of Content). When the "Show Top Authors" is clicked, the Content DIV will display the list of Top 20 authors. If you then click the "Show Top
Crime Books" the contents of the DIV tag will be replaced with the top 20 crime books. The two feeds are shown after the template example.

<!-- Main Content DIV Feed Results Will Be Placed Here -->
<div id="Content"></div>

<!-- LOAD FEED TAGS -->
<xmod:LoadFeedImage AlternateText="Show Top Authors" ImageUrl="~/images/Top20.png"
FeedName="Top20Authors" Target="#Content" LoadingImageUrl="~/images/loading.gif"/>
<xmod:LoadFeedImage AlternateText="Show Top Crime Books" ImageUrl="~/images/TopCrime.png"
FeedName="Top20CrimeBooks" Target="#Content">
<Field Name="GenreId" Value="20"/>
</xmod:LoadFeedImage>

**Example Feeds -- Top20Authors**

<xmod:Feed ContentType="text/html">
<ListDataSource CommandText="SELECT FirstName, LastName, AuthorId, SalesRank FROM Authors WHERE SalesRank <= 20"/>
<HeaderTemplate>
<table>
<thead>
<tr>
<th>Rank</th>
<th>Author</th>
</tr>
</thead>
<tbody>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td>[[SalesRank]]</td>
<td>[[FirstName]] [[LastName]]</td>
</tr>
</ItemTemplate>
.FooterTemplate>
</tbody>
</table>
</FooterTemplate>
</xmod:Feed>

**Example Feeds -- Top20CrimeBooks**

<xmod:Feed ContentType="text/html">
<ListDataSource CommandText="SELECT Title, SalesRank FROM Books WHERE GenereId = @GenreId">
<Parameter Name="GenreId" Value='[[Url:GenreId]]'/>
</ListDataSource>
<HeaderTemplate>
<table>
<thead>
<tr>
<th>Rank</th>
<th>Title</th>
</tr>
</thead>
<tbody>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td>[[SalesRank]]</td>
<td>[[Title]]</td>
</tr>
</ItemTemplate>
</FooterTemplate>
</xmod:Feed>
Example - Infinite Paging

This example shows how you can set up "Infinite Paging". The idea behind infinite paging is that the user is presented with an initial list of, say, 10 results. If there are 100 total records, normally the user would page to the next set of 10 and keep paging to browse through the results. Infinite paging leverages the power of AJAX to present all those results on the same page. So, the user is presented with a button that, when clicked, will load the next set of 10 results and add them to the existing list. So, now the list contains a total of 20 results. If the button is pressed again, 10 more results will be added to the list for a total of 30 results. The two feeds are shown after the template example.

NOTE: Infinite Paging will send your feed a parameter called "LastId" that you can use to determine the next set of values to return. However, Infinite Paging needs you to tell it where it can get the LastId. You do this by setting the IDSelector property. This should be a jQuery selector that points to the element that contains that value. XMod Pro will grab the innerText of that element and use that as the LastId. In the example below, we're storing the Author's ID in a hidden SPAN tag. Since the last-retrieved record id is going to be in the last record we display, we'll find the LastId in the list item.

Example Feed -- Infinite Paging, Authors

This feed is designed to work with the template specified above. There are several important items to implement to ensure successful implementation of infinite paging.

1. The feed must have its ContentType set to "text/html" since we're sending HTML back to the template.
2. The ListDataSource should use the URL parameter called "LastId" to determine which record(s) to return.
3. The ListDataSource is set to select the first 10 records that match the WHERE clause criteria. This allows you to return the next 10 records to the template. If your template was set to retrieve the next 20 records, the ListDataSource should SELECT TOP 20 ...
4. The ListDataSource has a <Parameter> tag that grabs the URL parameter called "LastId" that is passed automatically by XMod Pro.
5. Since we are injecting HTML into the template's page, we only need to render out the minimum HTML to add List Items to the unordered list. So, we don't have a HeaderTemplate or FooterTemplate - just an ItemTemplate matches the ItemTemplate in the <xmod:Template> tag above. This is important not only to maintain visual consistency but you must also be sure to include the <span> tag that contains the record's ID.
The LoadFeedLink tag makes it a cinch to dynamically load XMod Pro feeds into your page. This tag requires jQuery be included in the page. The LoadFeedLink tag renders as a hyperlink at run-time and will grab the feed data when clicked.

**Syntax**

```xml
<xmod:LoadFeedLink
  FeedName="string"
  IDSelector="jQuery element selector"
  InfinitePaging="True|False"
  InsertMode="Replace|Append|Prepend"
  LoadingCssClass="CSS Class Name(s)"
  LoadingImageUrl="url"
  Target="jQuery element selector"
  Text="string"
  [one or more optional Field tags can be used to pass data to the feed]
</xmod:LoadFeedLink>
```

**Remarks**

- **FeedName**: The name you have given your feed on the Manage Feeds page. Note that you should set the feed's ContentType to "text/html" (see examples below).

- **IDSelector**: Used if InfinitePaging is True, this is a jQuery Selector that locates the element that contains the ID for each record. XMod Pro will look at the inner text of the element. If you don't want the ID to appear to be visible, we recommend inserting a `<span>` tag that has its display set to none like so:

  ```html
  <table id="AuthorsTable">
    <tr>
      <td>[[FirstName]] [[LastName]]<span style="display:none;">[[AuthorID]]</span></td>
      <td>[[SalesRank]]</td>
    </tr>
  </table>
  ```

  In the above case, if you wanted to append new records to the end of the table, you would set your IDSelector like so:

  ```javascript
  IDSelector="#AuthorsTable tr:last td:first span"
  ```

- **InfinitePaging**: True or False. False by default. When set to true, XMod Pro will add jQuery and Javascript to the page enabling your button to trigger adding feed results to an existing list. For instance, you can display your initial results in a table and, when the button is clicked, the feeds results can be added to the existing table. You'll often see this behavior on mobile phones when viewing data from the web. At the bottom of a list of results you'll see a link like "Show 10 More...". Clicking that will add 10 more results to the list. Clicking the button again will add 10 more results the list, and so on. The
LoadFeedLink makes the front end of this process pretty painless to implement. You’ll just need to pair it with an XMod Pro feed that can return data in the format you expect. See the Example for details.

- **InsertMode**: Set to "Replace" by default. You can also set it to Append or Prepend. When set to replace, the feed’s results will replace the content of the target element (see Target property). When set to Append, the results will be added to the end of the target. Likewise, when set to Prepend, it will insert the results before the target. Note, when using Append and Prepend, it is important your XMod Pro feed return results using HTML that works with your page. For instance, if you’re appending to an existing table, your feed should return table rows (<tr> tags).

- **LoadingCssClass**: One or more CSS class names (separated by a space) that will be applied to the HTML image tag used to display a "loading" status to the user. This property is ignored if no LoadingImageUrl has been specified.

- **LoadingImageUrl**: The path to the image you’d like to display as a "loading" status indicator to your users. Typically this will be an animated GIF or similar file. You may use the tilde (~) character as a short-hand for the site's root directory.

- **Target**: A jQuery selector identifying the element whose content will be replaced by the results of the feed.

- **Text**: The caption that will be displayed on the button.

- **Field Tags**: If you need to pass additional information to your feed, use the child Field tags. The Field's Name property should be the name of the parameter your feed is looking for and the Value of that Field will be the value you pass in for the parameter.

---

**Example**

This example shows two LoadFeedLink tags. They retrieve different feeds but place them in the same element on the page (the DIV with id of Content). When the "Show Top Authors" is clicked, the Content DIV will display the list of Top 20 authors. If you then click the "Show Top Crime Books" the contents of the DIV tag will be replaced with the top 20 crime books. The two feeds are shown after the template example.

```xml
<!-- Main Content DIV Feed Results Will Be Placed Here -->
<div id='Content'></div>

<!-- LOAD FEED TAGS -->
<xmod:LoadFeedLink Text='Show Top Authors' FeedName='Top20Authors' Target='#Content' LoadingImageUrl='~/images/loading.gif' />
<xmod:LoadFeedLink Text='Show Top Crime Books' FeedName='Top20CrimeBooks' Target='#Content'>
  <Field Name='GenreId' Value='20' />
</xmod:LoadFeedLink>

**Example Feeds -- Top20Authors**

```xml
<xmod:Feed ContentType='text/html'>
  <ListDataSource CommandText='SELECT FirstName, LastName, AuthorId, SalesRank FROM Authors WHERE SalesRank <= 20' />
  <HeaderTemplate>
    <table>
      <thead>
        <tr>
          <th>Rank</th>
          <th>Author</th>
        </tr>
      </thead>
      <tbody>
      </tbody>
    </table>
  </HeaderTemplate>
  <ItemTemplate>
    <tr>
      <td>{{SalesRank}}</td>
    </tr>
  </ItemTemplate>
</xmod:Feed>
```
Example Feeds -- Top20CrimeBooks

Example - Infinite Paging

This example shows how you can setup "Infinite Paging". The idea behind infinite paging is that the user is presented with an initial list of, say, 10 results. If there are 100 total records, normally the user would page to the next set of 10 and keep paging to browse through the results. Infinite paging leverages the power of AJAX to present all those results on the same page. So, the user is presented with a button that, when clicked, will load the next set of 10 results and add them to the existing list. So, now the list contains a total of 20 results. If the button is pressed again, 10 more results will be added to the list for a total of 30 results. The two feeds are shown after the template example.

NOTE: Infinite Paging will send your feed a parameter called "LastId" that you can use to determine the next set of values to return. However, Infinite Paging needs to you to tel l it where it can get the LastId. You do this by setting the IDSelector property. This should be a jQuery selector that points to the element that contains that value. XMod Pro will grab the innerText of that element and use that as the LastId. In the example below, we're storing the Author's ID in a hidden SPAN tag. Since the last-retrieved record id is going to be in the last record we display, we'll find the LastId in the list item.

Example Feeds -- Top20CrimeBooks

Example - Infinite Paging

This example shows how you can setup "Infinite Paging". The idea behind infinite paging is that the user is presented with an initial list of, say, 10 results. If there are 100 total records, normally the user would page to the next set of 10 and keep paging to browse through the results. Infinite paging leverages the power of AJAX to present all those results on the same page. So, the user is presented with a button that, when clicked, will load the next set of 10 results and add them to the existing list. So, now the list contains a total of 20 results. If the button is pressed again, 10 more results will be added to the list for a total of 30 results. The two feeds are shown after the template example.

NOTE: Infinite Paging will send your feed a parameter called "LastId" that you can use to determine the next set of values to return. However, Infinite Paging needs to you to tell it where it can get the LastId. You do this by setting the IDSelector property. This should be a jQuery selector that points to the element that contains that value. XMod Pro will grab the innerText of that element and use that as the LastId. In the example below, we're storing the Author's ID in a hidden SPAN tag. Since the last-retrieved record id is going to be in the last record we display, we'll find the LastId in the list item.
Example Feed -- Infinite Paging, Authors

This feed is designed to work with the template specified above. There are several important items to implement to ensure successful implementation of infinite paging.

1. The feed must have its ContentType set to "text/html" since we’re sending HTML back to the template.
2. The ListDataSource should use the URL parameter called "LastId" to determine which record(s) to return.
3. The ListDataSource is set to select the first 10 records that match the WHERE clause criteria. This allows you to return the next 10 records to the template. If your template was set to retrieve the next 20 records, the ListDataSource should SELECT TOP 20 ...
4. The ListDataSource has a <Parameter> tag that grabs the URL parameter called "LastId" that is passed automatically by XMod Pro.
5. Since we are injecting HTML into the template's page, we only need to render out the minimum HTML to add List Items to the unordered list. So, we don’t have a HeaderTemplate or FooterTemplate - just an ItemTemplate matches the ItemTemplate in the <xmod:Template> tag above. This is important not only to maintain visual consistency but you must also be sure to include the <span> tag that contains the record’s ID.

Syntax

```xml
<Feed ContentText="text/html">
  <ListDataSource CommandText="SELECT TOP 10 FirstName, LastName, AuthorId FROM Authors WHERE AuthorId > @LastId ORDER BY AuthorId ASC">
    <Parameter Name="LastId" Value='[[Url:LastId]]' />
  </ListDataSource>
  <ItemTemplate>
    <li>[[FirstName]] [[LastName]]<span style="display:none;">[[AuthorId]]</span></li>
  </ItemTemplate>
</Feed>
```

Remarks

The MetaTags tag is used to alter the host page's Title, Description, and Keywords. This is useful for Search Engine Optimization (SEO) as well as being an aid for your users.

Syntax

```xml
<MetaTags>
  <Title append="True|False">...Title Text...</Title>
  <Keywords append="True|False">...Keywords...</Keywords>
  <Description append="True|False">...Description...</Description>
</MetaTags>
```

Remarks

- **Title**: The page’s Title will be replaced with the tag's inner content. If append is specified and is True, then the content will be appended to the page’s Title.
**Keywords**: The page's Keywords will be replaced with the tag's inner content. If append is specified and is True, then the content will be appended to the page's Keywords. Note: for this tag to function correctly, you should set the default Keywords in the DNN Site Settings page for your website.

**Description**: The page's Description will be replaced with the tag's inner content. If append is specified and is True, then the content will be appended to the page's Description. Note: for this tag to function correctly, you should set the default Description in the DNN Site Settings page for your website.

**Example**

```xml
<div>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <!-- EMPLOYEES TEMPLATE -->

        <xmod:template id="Employees">
          <detaildatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
            <parameter name="EmployeeId" alias="EmpID" />
          </detaildatasource>

          <detailtemplate>
            <h1>Employee Profile</h1>
            <h3>[[FirstName]] [[LastName]]</h3>
            <h4>Biography:</h4>
            <div>[[Bio]]</div>
          </detailtemplate>
        </xmod:template>
      </td>
    </tr>
  </table>
</div>
```

**Syntax Remarks Example**

The Redirect tag is a great way to extend XMod Pro's ability to interact with other applications and services. Its sole purpose is to send data to other URL's. You tell it which data to send by adding `<field>` tags. At run-time, it renders as a hyperlink, a push-button, or a clickable image (depending on its settings) and, when clicked will send the information it finds in the `<field>` tags to URL you specify. Data can be sent via HTTP POST (the same method used when you click the Submit button on a standard web form) or HTTP GET (which passes your data as parameters in the URL).
Syntax

<xmod:Redirect>
  Display="button|linkbutton|imagebutton"
  ImageUrl="url"
  ImageAlign="abbottom|absmiddle|baseline|bottom|left|middle|right|texttop|top"
  Method="get|post"
  OnClientClick="javascript"
  Style="string"
  Target="url"
  Text="string"
  ToolTip="string"
  Visible="True|False"
  Width="size">
  <Field Name="string" Value="string" />
  additional fields as needed ...
</xmod:Redirect>

Remarks

- **Display**: Determines how the button will be displayed. The default value is LinkButton. Valid values are:
  - **Button**: displays as a push-button
  - **LinkButton**: displays as a hyperlink
  - **ImageButton**: displays as a clickable image
- **Method**: Determines how the data will be sent. The default value is Post. Valid values are:
  - **Get**: data is sent via the HTTP GET method (i.e. via URL parameters)
  - **Post**: data is sent via the HTTP POST method. NOTE: If you need to POST form data to the target URL, you should use the `<xmod:Redirect>` tag instead.
- **Text**: When Display is set to LinkButton or Button, this will be the text that is displayed to to the user. If Display is set to ImageButton then the text will be used as the image’s alt. text.
- **Target**: A valid URL. This is the destination to which the button’s data will be sent. As with many properties in XMod Pro tags, you can use the tilde (~) character to represent the website root - e.g. ~/mypage.aspx
- **ImageUrl**: A valid URL pointing to an image file. This attribute is only used when Display is set to ImageButton.
- **ImageAlign**: A value determining how the image will be aligned relative to nearby elements in the page. It is ignored unless display is ImageButton. Valid values are:
  - **abbottom** - the lower edge of the image is aligned with the lower edge of the largest element on the same line
  - **absmiddle** - the middle of the image is aligned with the middle of the largest element on the same line
  - **baseline** - the lower edge of the image is aligned with the lower edge of the first line of text
  - **bottom** - the lower edge of the image is aligned with the lower edge of the first line of text
  - **left** - the image is aligned on the left edge with the text wrapping on the right
  - **middle** - the middle of the image is aligned with the lower edge of the first line of text
  - **right** - the image is aligned on the right edge with text wrapping on the left
  - **texttop** - the upper edge of the image is aligned with the upper edge of the highest text on the same line
  - **top** - the upper edge of the image is aligned with the upper edge of the highest element on the same line
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.
Example
<div>
<table width="100%">
<tr>
<td colspan="2">
<!-- DEPARTMENTS TEMPLATE -->
<xmod:template id="Departments">
<listdatasource commandtext="SELECT ProductId, ProductName FROM XMPDemo_Products ORDER BY ProductName" />
<itemtemplate>
<xmod:redirect text="Purchase" target="http://mysite.com/purchase.aspx">
<field name="pid" value='[[ProductId]]' />
</xmod:redirect>
</itemtemplate>
</xmod:template>
</td>
</tr>
</table>
</div>

<xmod:Register>
Syntax Remarks Example
The Register tag enables you to use third-party controls in your XMod Pro templates.

Syntax
<xmod:Register
  TagPrefix="string"
  Namespace="string"
  Assembly="string"
 />

Remarks
- Should you choose to use third-party controls, you’ll need to add a Register tag to your template definition for each collection. Register tags tell XMod Pro where to find the controls you use. You only use the tag once per library. The register tag is declared outside all the <xmod:Template> tags in your template. This allows you to use the library in all your template tags.
- **TagPrefix**: A short series of letter and numbers that you use as part of the control’s tag. It helps XMod Pro determine what library the control belongs to.
- **Namespace**: The namespace in which custom control resides. This information should be supplied by the control developer.
- **Assembly**: This is the name of the assembly (DLL) in which the controls reside. Note, you do not specify the path to the DLL or the ".dll" extension. This information should be supplied by the control developer.

Example
<xmod:template>
...<HeaderTemplate>
<table>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td>
The ReturnButton tag renders as a push-button at run-time. It is used to return to the list view from a detail view.

Syntax

```
<xmod:ReturnButton
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    OnClientClick="javascript"
    Style="string"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size"/>
```

Remarks

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Usage**: The XMod Pro Return controls are used in detail templates. There, they serve to return the user to the list view he/she was previously viewing.

Example

```
<div>
    <table width="100%">
        <tr>
            <td width="250" valign="top">
                <!-- EMPLOYEES TEMPLATE -->
            </td>
        </tr>
    </table>
</div>
```
The ReturnImage tag renders as a clickable image at run-time. It is used to return the user from a detail view to the previously viewed list view.

Syntax

```xml
<xmod:ReturnImage
    AlternateText="string"
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    ImageAlign="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbsBottom|AbsMiddle|TextTop"
    ImageUrl="url"
    OnClientClick="javascript"
    Style="string"
    Text="string"
    ToolTip="string"
```
Remarks

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.
- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application's root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Usage**: The XMod Pro Return controls are used in detail templates. There, they serve to return the user to the list view he/she was previously viewing.

Example

```xml
<directive>
    <table width="100%">
        <tr>
            <td width="250" valign="top">
                <!-- EMPLOYEES TEMPLATE -->

                <xmod:template id="Employees">
                    <listdatasource commandtext="SELECT EmployeeId, FirstName, LastName FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
                        <parameter name="DepartmentId" alias="DepartmentId"/>
                    </listdatasource>
                    <detaildatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
                        <parameter name="EmployeeId" alias="EmpID" />
                    </detaildatasource>
                    <headertemplate>
                        <p>Employees</p>
                    </headertemplate>
                    <itemtemplate>
                        <div style="text-align: middle;">
                            <strong>
                                [[FirstName]] [[LastName]]
                            </strong>
                        </div>
                    </itemtemplate>
                    <detailtemplate>
                        <h1>Employee Profile</h1>
                        <h3>[[FirstName]] [[LastName]]</h3>
                        <h4>Biography:</h4>
                        <div>[[Bio]]</div>
                        <xmod:returnimage text="Go Back" imageurl="~/images/leftarrow.gif" alternatetext="Return"/>
                    </detailtemplate>
                </xmod:template>
            </td>
        </tr>
    </table>
</directive>
```
The ReturnLink tag renders as a clickable image at run-time. It is used to return the user from a detail view to the previously viewed list view.

**Syntax**

```xml
<modx:ReturnLink
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    OnClientClick="javascript"
    Style="string"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size"/>
```

**Remarks**

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Usage**: The XMod Pro Return controls are used in detail templates. There, they serve to return the user to the list view he/she was previously viewing.
Example

```html
<example>
<table width="100%">
<tr>
<td width="250" valign="top">
<!-- EMPLOYEES TEMPLATE -->

<xmod:template id="Employees">
  <listdatasource commandtext="SELECT EmployeeId, FirstName, LastName FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
    <parameter name="DepartmentId" alias="DepartmentId" />
  </listdatasource>
  <detaildatasource commandtext="SELECT * FROM XMPDemo_Employees WHERE EmployeeId = @EmpID">
    <parameter name="EmployeeId" alias="EmpID" />
  </detaildatasource>

  <headertemplate>
    <p>Employees</p>
  </headertemplate>

  <itemtemplate>
    <div style="text-align: middle;">
      <strong>[[FirstName]] [[LastName]]</strong>
      <xmod:detailbutton text="View Profile">
        <parameter name="EmployeeId" value='[[EmployeeId]]' />
      </xmod:detailbutton>
    </div>
  </itemtemplate>

  <detailtemplate>
    <h1>Employee Profile</h1>
    <h3>[[FirstName]] [[LastName]]</h3>
    <h4>Biography:</h4>
    <div>[[Bio]]</div>
    <xmod:returnlink text="Go Back" />
  </detailtemplate>
</xmod:template>
</td>
</tr>
</table>
</example>
```

**Syntax Remarks Example**

The ScriptBlock tag is used to inject HTML `<script>` tags into one of several different locations in the page. Typically this is used to insert JavaScript functions and/or libraries into the page. You can also insert `<style>` tags into the page using this tag.
Syntax

```xml
<xmod:ScriptBlock
    ScriptId="string"
    BlockType="HeadScript|ClientScript|StartupScript|ClientScriptInclude"
    RegisterOnce="True|False"
    Url="url">
    <script type="text/javascript" ...>
        ...Javascript...
    </script>
</xmod:ScriptBlock>
```

Remarks

- **ScriptId**: This is an identifier for your block that uniquely identifies it within the hosting page - across modules. It is used when registering your script block and is required in order to prevent the block from being inserted more than once in the page.

- **BlockType**: This attribute allows you to specify which where in the page the script should be rendered. The default value is ClientScript
  - **HeadScript**: The script block will be inserted between the `<head>` and `</head>` section of the page.
  - **ClientScript**: The script block will be inserted near the top of the page.
  - **ClientScriptInclude**: Use this block type to insert a `<script>` tag that links to an external file. This is useful for including Javascript libraries.
  - **StartupScript**: The script block will be inserted near the bottom of the page.

- **RegisterOnce**: If this value is set to True, the tag will first check to see if a code block with ScriptId has been registered in the page. If not, it will register your block. If it has been registered already, then no action is taken. If this value is False, the default value, then your script block will be inserted, regardless of any previously registered block. The RegisterOnce is only available for ClientScript, ClientScriptInclude, and StartupScript block types.

- **Url**: If the BlockType is set to ClientScriptInclude, this is the path to the Javascript file you wish to include. It is ignored if BlockType is set to a different value. You may optionally use the tilde (~) character in the URL to represent the path to the root of the web application.

Back to top
Example

```html
<div>
<xmod:scriptblock scriptid="AlertScripts" registeronce="true">
  <script language="javascript">
    function helloWorld(){
      alert('Hello World');
    }
    function goodbyeWorld(){
      alert('Goodbye Cruel World');
    }
    function showMessage(sMessage){
      alert(sMessage);
    }
  </script>
</xmod:scriptblock>
<table width="100%">
  <tr>
    <td width="250" valign="top">
      <!-- SCRIPT BLOCK EXAMPLE -->
      <a href="#" onclick="helloWorld();">Hello World</a><br />
      <a href="#" onclick="goodbyeWorld();">Goodbye</a><br />
      <a href="#" onclick="showMessage('Hello and Goodbye')">Show Message</a>
    </td>
  </tr>
</table>
</div>
```

Syntax Remarks Example

The `<xmod:select>` gives you a lot of flexibility in creating your display template. It's similar in function to Visual Basic's Select/Case statement and Javascript's switch/case statement. For non-programmer's you can think of it as a multiple choice tag. The tag can be used both within a template and outside of a template. In fact, it can be used to hide or show templates as well as other text and HTML.

Syntax

```
<xmod:select mode="Standard|Inclusive">
  <case comparetype="Numeric|Float|Date|Text|Regex|Boolean|Role"
    value="string"
    expression="string"
    operator="|>|<|>||<|>="
    ignorecase="true|false"
    culture="LocaleID">
    <case>...</case>
    ...
    <else>...</else>
</xmod:select>
```

Remarks

- The `<xmod:select>` tag contains one or more `<case>` tags and one optional `<else>` tag. At run-time, XMod Pro will evaluate each `<case>` tag beginning with the first one in the list. If the tag doesn't evaluate to True, XMod Pro moves to the next `<case>` tag and evaluates it. It continues this way until it reaches a true `<case>`. If none is found, then nothing will be displayed unless an `<else>` tag has been included. If so, it's contents will be displayed.
Comparison Data Types: Note that when XMod Pro tests to see if the text "1" is the same as the text "1", XMod isn't comparing the values as numbers. It's comparing the values as text. However, that can easily be changed by using the <case>tag's comparetype attribute. Using this, you can tell XMod to compare the values as numeric values, floating point numbers, dates, boolean, text, or to see if the value matches a regular expression pattern. You can also compare whether the current user is/isn't in one or more security roles.

Operators: You're not limited to just testing for equality. Using the operator attribute of the <case>tag, you can test for equality "=" , inequality "<>", less-than "<", greater-than ">", less-than-or-equal "<=" , and greater-than-or-equal ">=". Of course, these operators don't make sense for all types of comparisons. For instance, if you are doing a regular expression comparison, the value can either match or not-match the regular expression pattern. So, the only valid operators in that case are "=" and "<>".

Case-Sensitive Comparisons: If you're doing text comparisons, you can set the <case>tag's ignorecase attribute to "true" or "false" to tell the tag to perform a case-insensitive comparison (true) or a case-sensitive comparison (false).

Culturally-Sensitive Comparisons: The <case>tag includes a culture attribute. This can be set to an LCID (a culture ID) that will instruct the tag that comparisons should be made using the settings of that culture. If no culture attribute is set, the tag will attempt to use the current culture. NOTE: Boolean comparisons do not use a culture setting. Additionally, regular expressions use the system's culture by default. You cannot specify a culture for them. However, as with all comparisons, you can specify "invariant" as the culture to perform a culturally neutral comparison.

Evaluation Modes: The <xmod:select> tag provides one attribute: mode. Valid values for this attribute are "standard" and "inclusive". By default, mode is set to "standard".

- Standard: the tag will evaluate all <case> tags until it finds one that evaluates to true. It will display that tag's content and stop evaluating. If no true <case>tag is found and an <else>tag has been supplied, the <else> tag's content will be displayed.
- Inclusive: ALL <case> tags will be evaluated. Any tags that evaluate to true will have their content displayed. If an <else> tag has been supplied, its content will also be displayed.

Attributes:
- value: This is the value the tag will test. This attribute is REQUIRED unless comparetype is set to "role"
- expression: This is the expression used to test the value. This can be a hard-coded value or field token. When comparetype is set to "regex", expression should be a regular expression pattern. When comparetype is set to "role", expression should be set to a comma-delimited list of security role names. This attribute is REQUIRED.
- operator: Determines what kind of evaluation will be made. Valid values are:
  - = (equality)
  - < (less-than)
  - > (greater-than)
  - <> (not equal)
  - <= (less-than or equal-to)
  - >= (greater-than or equal-to)

  Example: value="1" and expression="2" and operator="<". This would translate to: 1<2. The value property is always on the left side of the operator while expression is always on the right. NOTE: Not all operators function for all types of comparisons. With regular expression comparisons, you can specify "=" for "Is A Match" and "<>" for "Is NOT A Match", but the other operators have no relevance. If "<>" is not specified, then "=" is assumed.
- comparetype: This allows you to tell the tag what type of data is being compared. This attribute is REQUIRED. Valid values are:
  - numeric: value and expression are treated as numbers
  - float: value and expression are treated as floating-point numbers
  - date: value and expression are treated as date/time values
  - text: value and expression are treated as text
  - regex: value is compared against the regular expression pattern in expression. Valid operators are "=" and "<>"
  - boolean: value and expression are treated as True/False values. When comparing numbers as booleans, 0 is False and all other numbers are True. Valid operators are "=" and "<>"
  - role: expression is treated as a comma-delimited list of security role names. value is ignored. Valid operators are "=" and "<>". Note that if you are logged-in as host, the case statement will evaluate to True, even
though the account may not be a member of one of the specified roles. To accurately test the functionality, you should login as a non-host/superuser account.

Example

```xml
<modx:template ...>
...

<itemtemplate>
  Your favorite color is:
  <modx:select>
    <case comparetype="text" operator="=" value='[[FavColor]]'
      expression="blue" ignorecase="true">
      <span color="#0000FF">BLUE</span>
    </case>
    <case comparetype="text" operator="=" value='[[FavColor]]'
      expression="red" ignorecase="true">
      <span color="#FF0000">RED</span>
    </case>
    <case comparetype="text" operator="=" value='[[FavColor]]'
      expression="green" ignorecase="true">
      <span color="#00FF00">GREEN</span>
    </case>
    <else>
      We don't know your favorite color
    </else>
  </modx:select>
<br />
<modx:select>
  <case comparetype="role" operator="=" expression="Administrators">
    (This area reserved for Admins only)
  </case>
</modx:select>
</itemtemplate>
...
</modx:template>
```

<xmod:Slideshow>

Syntax  Remarks  Example

The Slideshow tag, like the Template tag, is a View control that is used for displaying records from your datasource. Unlike the Template tag, the Slideshow is a view with a single purpose- to display images in an eye-pleasing presentation with very little work on your part. All you have to do is supply it with a list of URL's to your images and set a Width and Height for the viewer. It is used in essentially the same manner as the Template tag and can be used together with the Template tag within your templates.

Unlike the Template or DataList tags, the Slideshow doesn't allow any data commands or paging. Additionally, since this is a presentation control, it is assumed that those who can see the module can view the slideshow, so there are no permissions. As a result, the syntax is very simple.

IMPORTANT: Use of this control REQUIRES jQuery 1.3 or later be available on the page.
Syntax

```xml
<xmod:Slideshow
  BasePath="string"
  ConnectionString="string"
  Height="pixels"
  ID="string"
  ImageField="string"
  ResizeImages="True|False"
  Timeout="milliseconds|4000"
  Width="pixels">
  <ListDataSource CommandText="string"
    ConnectionString="string"/>
</xmod:Slideshow>
```

Remarks

- **BasePath**: Provides the option of pre-pending a common path for your image URL's. So, if your image URL's are stored in the data source with their filename only, you can set BasePath to point to the actual directory where the image files reside. You can optionally use the tilde (~) character to represent the application root.

- **ConnectionString**: If you need to specify a SQL Server database other than the current DotNetNuke database, you can supply a connection string here. When specified, it serves as the default connection string for the ListDataSource and DetailDataSource. You can also use a connection string defined in the web.config file. To do so, use the ConnectionString token like so:

  ```xml
  <xmod:DataList ... ConnectionString='[[ConnectionString:connectionName]]' ...>
  ```

- **Height**: Height of the slideshow view port, specified as a number pixels. Do not add "px" to the value. This should be the height of the largest image, generally. For best results all images should be same size. If not, try shrinking the dimensions of the viewport and setting ResizeImages to true.

- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the template file.

- **ImageField**: Required. Specify the name of the column in your data source that contains the URL of the images.

- **ResizeImages**: If set to true, all images will be set to the same size as the slideshow viewport (Height and Width properties). If false, the default, images larger than the viewport size will be partially visible. Setting this to true can be helpful if your images are not the same size. In that case, set the Height and Width to the size of the smallest image and set ResizeImages to true. This will cause your larger images to display in a smaller size (the image is not affected). For best results, all images should be the same size.

- **Timeout**: The time, in milliseconds (1 second = 1000 milliseconds), that an image should be displayed. The default value is 4000 or 4 seconds.

- **Width**: Width of the slideshow view port, specified as a number pixels. Do not add "px" to the value. This should be the width of the largest image, generally. For best results all images should be same size. If not, try shrinking the dimensions of the viewport and setting ResizeImages to true.

- **ListDataSource**: Provides the data for the DatList when it is displaying in list view. It accepts <parameter> tags to pass parameters as part of the command. The "alias" attribute is optional and is used to avoid conflicts or to accept a parameter with a specific name (the "name" attribute) but use a different parameter name in the command (the "alias" attribute). If no "alias" is specified, the "name" will be used.
<xmod:Slideshow ImageField="EmployeePicUrl" Height="250" width="200" Timeout="6000" BasePath="/images/employees/">
  <ListDataSource commandtext="SELECT EmployeePicUrl FROM Employees" />
</xmod:Slideshow>

Syntax Remarks Example

The template tag is the primary workhorse of XModPro. It contains the data commands and layout instructions for your display. Additionally, it enables you to specify who is allowed to add, edit, and delete records as well as who is allowed to view the details of records. You can use multiple Template tags, enabling you to have side-by-side (or however you want to lay them out using HTML) displays within the same module instance, each being fed by different datasources.

Syntax

<xmod:template
  AddRoles="DNNRoleName1,DNNRoleName2"
  Ajax="True|False"
  ConnectionString="string"
  DeleteRoles="DNNRoleName1,DNNRoleName2"
  DetailRoles="DNNRoleName1,DNNRoleName2"
  EditRoles="DNNRoleName1,DNNRoleName2"
  ID="string"
  UsePaging="True|False">
  <ListDataSource CommandText="string"
                   ConnectionString="string"/>
  <DetailDataSource CommandText="string"
                     ConnectionString="string"/>
  <DeleteCommand CommandText="string" />
  <CustomCommands>
    <DataCommand CommandName="string" CommandText="string" ConnectionString="string">
      <Parameter Name="string" Value="string"/>
    </DataCommand>
  </CustomCommands>
  <Pager> ... </Pager>
  <SearchSort>... </SearchSort>
  <HeaderTemplate>...</HeaderTemplate>
  <ItemTemplate>...</ItemTemplate>
  <AlternatingItemTemplate>...</AlternatingItemTemplate>
  <SeparatorTemplate>...</SeparatorTemplate>
  <FooterTemplate>...</FooterTemplate>
  <DetailTemplate>...</DetailTemplate>
  <NoItemsTemplate>...</NoItemsTemplate>
</xmod:template>
Remarks

- **AddRoles/EditRoles/DeleteRoles/DetailRoles**: Each of these properties are lists of DNN role names, separated by commas. Host/SuperUser and Admin accounts always have add/update/delete/detail permissions. By default, anyone who has access to view the module also has detail viewing permissions. Supply the DetailRoles attribute to limit who can view the details of records within the template. Users other than Host/Admin must be explicitly granted add/update/delete rights. Prior to version 4.0, role names were separated by semi-colons. To enhance consistency with other areas of XMod Pro, commas are now used. For backwards compatibility, you can still use semi-colons, but their usage is deprecated as of version 4.0. You should use commas going forward.

  **NOTE**: These permissions apply ONLY to controls within the <xmod:Template> tag. If you have an <xmod:AddButton>, and <xmod:AddLink>, or <xmod:AddImage> tag placed outside of the <xmod:Template> tag, you will need to set permissions at the module level by going to the module's Configure page and choosing the Security tab. There you can choose the roles allowed to add records.

- **Ajax**: New to version 2.6, when this attribute is true, actions within the template tag will be performed without a full page refresh. This value is false by default. **NOTE**: buttons that postback such as the DetailButton (or DetailImage or DetailLink) or AddButton or DeleteButton or CommandButton, etc. must have their ID property set to function correctly. If you have to click twice on a button, for instance, make sure its ID has been set.

  **ConnectionString**: If you need to specify a SQL Server database other than the current DotNetNuke database, you can supply a connection string here. When specified, it serves as the default connection string for the ListDataSource and DetailDataSource. You can also use a connection string defined in the web.config file. To do so, use the ConnectionString token like so:

  ```xml
  <xmod:template ... ConnectionString="[[ConnectionString:connectionName]]" ...
  ```

- **UsePaging**: If set to true, the default, list views will display a pager if more than one page is available for display. The pager can be set using the <Pager> tag.

- **Data Sources**: The data for each template is a SQL command. This could be a SQL SELECT command or a stored procedure. Supply the SQL command as the commandtext attribute. For stored procedures, use `<EXEC <sprocName>`. In the initial release, data sources can only point to the current DotNetNuke database. Also note that Bit columns are returned as True/False - not 1/0. The data source for list views is defined in the <ListDataSource> tag while the data source for the detail view is defined in the DetailDataSource tag.

  In addition to its attributes, the Template tag also contains numerous child tags which define most of the tag's functionality:

    - **ListDataSource**: Provides the data for the template when it is displaying in list view. It accepts <parameter> tags to pass parameters as part of the command. The "alias" attribute is optional and is used to avoid conflicts or to accept a parameter with a specific name (the "name" attribute) but use a different parameter name in the command (the "alias" attribute). If no "alias" is specified, the "name" will be used.

      **Output Parameters**: New to version 1.3, you can retrieve and use the value of SQL Output parameters. To do so, add the Direction="Output" to the <parameter> tag and specify a DataType and Size (required for Strings). See Example #4 and the Data Parameter Tokens Topic.

  Examples:

  ```xml
  #1
  <ListDataSource CommandText="SELECT FirstName, LastName FROM Users" />
  
  #2
  <ListDataSource CommandText="SELECT FirstName, LastName FROM Users
    WHERE ZipCode = @zip">
    <parameter name="zip" alias="zip" value="12345" />
  </ListDataSource>
  ```
You can optionally specify a `ConnectionString` for this data source. If none is specified, the default connection will be used - either the connection specified by the `ConnectionString` property of the parent `<xmod:template>` tag or the current DotNetNuke database if the template doesn't define a connection. As with the template tag, you can use the `[[ConnectionString:connectionName]]` token to use a connection defined in the web.config file.

- **DetailDataSource**: Provides the data for the template when it is displaying a single record. It accepts `<parameter>` tags like the `<ListDataSource>`. You can optionally specify a `ConnectionString` for this data source. If none is specified, the default connection will be used - either the connection specified by the `ConnectionString` property of the parent `<xmod:template>` tag or the current DotNetNuke database if the template doesn't define a connection. As with the template tag, you can use the `[[ConnectionString:connectionName]]` token to use a connection defined in the web.config file.

DetailDataSource tags can also take **Output Parameters**. See the ListDataSource topic above for details.

- **DeleteCommand**: Provides the command to execute when a Delete Button/Image/Link is pressed within the template. It accepts `<parameter>` tags like the `<ListDataSource>`

- **CustomCommands**: New to version 3. Custom commands allow you to execute commands on your data source beyond the standard list/detail/delete commands. For instance, you may use them to create an Approve Record button which would set the approval status on a record. Custom Commands are triggered using `<xmod:CommandButton>`, `<xmod:CommandLink>` and `<xmod:CommandImage>` controls. For the `<Command>` tag, specify the Name of the command and set the Type to Custom. When a custom command is executed, the page will postback to the server and XMod Pro will attempt to reload the result set on the same page the user was on when the command button was clicked. In most cases this should update any values on that page that have been changed. An example is below:

```xml
<CustomCommands>
  <DataCommand CommandName="ChangeLastNameToSmith" CommandText="UPDATE Authors SET LastName='Smith' WHERE AuthorId=@AuthorId">
    <Parameter Name="AuthorId" />
  </DataCommand>
</CustomCommands>
```

```xml
<ItemTemplate>
  ...
</ItemTemplate>
```

```xml
<xmod:CommandButton Text="CustomCommand">
  <Command Name="ChangeLastNameToSmith" Type="Custom">
    <Parameter Name="AuthorId" Value="[[AuthorId]]" />
  </Command>
</xmod:CommandButton>
```
● **HeaderTemplate**: When displaying a list view, HTML, text, and controls that will be rendered once, at the beginning of the list. This template is optional.

● **ItemTemplate**: When displaying a list view, HTML, text, field tokens, and controls that will be rendered for each record in the result set. If the AlternatingItemTemplate is supplied, then the ItemTemplate will be used for all odd-numbered records.

● **AlternatingItemTemplate**: When displaying a list view, HTML, text, field tokens, and controls that will be rendered for the even-numbered records in the result set. This template is optional.

● **SeparatorTemplate**: When displaying a list view, this will render any HTML or text that you specify between each record. One good example is if you’re creating a list of comma-separated records (we’ll use numbers for simplicity). If you put your comma in the ItemTemplate, you would get a trailing comma like this: 1,2,3. But if you put your comma in the SeparatorTemplate, you won’t get that trailing comma: 1,2,3. (Added in version 1.4)

● **FooterTemplate**: When displaying a list view, HTML, text, and controls that will be rendered once, at the end of the list. This template is optional.

● **DetailTemplate**: Defines the layout for viewing a single record. When the user clicks a detail button, this template will be used. This template can contain HTML, text, Field Tokens, and controls. It is required if you will be using XMod Pro’s detail buttons in your list view.

● **NoItemsTemplate**: Defines the layout and text to display when the data source returns no records. This tag is optional. If not specified, nothing will be shown.

● **Pager**: This optional tag allows you to override the default configuration and look of the top and bottom pagers. Additionally, you can optionally override the default layout of the top and bottom pagers using your own HTML. See the `<Pager>` topic for more details.

● **SearchSort**: This allows you to define basic searching and sorting for your template. For more details see the `<SearchSort>` topic.

● **Field Tokens**: In order to display data from your data source, XMod Pro uses "field tokens". These are essentially placeholders that contain the name of the column or field in your data source. At run-time, these tokens are replaced with the value from record. Field Tokens are written in this form: `[[FieldName]]"FieldName"` is the name of the field or column in your data source. This name is case sensitive and must match the field/column's name exactly. It must be surrounded by `[[` and `]]`. You can use the Field Token in many places. However, when you use it as the attribute value for an XMod Pro tag or other third party control, you must delimit the attribute value with single quotes, rather than double quotes. See the `<xmod:detailbutton>` code in the example. When used in the `<parameter>` tag, the attribute is written as: `value='[[UserID]]'` rather than: `value="[[UserID]]"`. When using a Field Token with HTML tag attributes, this is not necessary.
The Pager tag can only be used within a Template or DataList tag. The tag defines how the pagers in the view will look and function. The Pager attributes provide the information necessary to define how the components of the top and bottom pagers will look while the inner content of the Pager allows you to use HTML to determine how the components are arranged. If no inner content is specified, the default layout will be used. If no tag is specified, the pagers will not be shown.

NOTE: To turn off paging for the view, use the UsePaging attribute of the `<xmod:template>` or `<xmod:DataList>` tag.

**Syntax**

```xml
<Pager
    FirstLastCssClass="string|CommandButton"
    FirstPageCaption="string|First"
    LastPageCaption="string|Last"
    MaxPageNumButtons="integer|5"
    NextPageCaption="string|Next"
    PageNumCssClass="string|CommandButton"
    PageSize="integer|10"
    PrevNextCssClass="string|CommandButton"
    PrevPageCaption="string|Prev"
    ScrollToTop="True|False"
    ShowBottomPager="True|False"
    ShowFirstLast="True|False"
    ShowPrevNext="True|False"
    ShowTopPager="True|False">
    ...Display Template (see Remarks)...
</Pager>
```
Remarks

- **FirstLastCssClass**: The Cascading Style Sheet (CSS) class name to associate with the First Page and Last Page navigation links. The default value is CommandButton.
- **FirstPageCaption**: The text to use for the First Page navigation link, when it is visible. The default value is "First".
- **LastPageCaption**: The text to use for the Last Page navigation link, when it is visible. The default value is "Last".
- **MaxPageNumButtons**: Determines the maximum number of page link buttons which will be displayed. Page number buttons enable the user to navigate to a specific page by clicking on them. They appear between the Previous and Next navigation buttons. The default value is 5.
- **NextPageCaption**: The text to use for the Next Page navigation link, when it is visible. The default value is "Next".
- **PageNumCssClass**: The Cascading Style Sheet (CSS) class name to associate with the page number link buttons. The default value is CommandButton.
- **PageSize**: Determines the maximum number of records to display on each page. The default value is 10.
- **PrevNextCssClass**: The Cascading Style Sheet (CSS) class name to associate with the Previous Page and Next Page navigation links. The default value is CommandButton.
- **PrevPageCaption**: The text to use for the Previous Page navigation link, when it is visible. The default value is "Prev".
- **ScrollToTop**: A true/false value which is set to true by default. When true, if the user clicks a link in the Bottom Pager, the page, when it is reloaded, will scroll to the top of the page. This is helpful if you have a long list of items and the user has scrolled down before clicking the Next page (or any pager) link. The default DNN action would maintain his/her position on the new page, which is not what the user would expect. By setting ScrollToTop to true, you can take the user back to the top of the page. **NOTE**: If the user clicks a link in the Top Pager, the page will not scroll to the top - even if this property is true. Instead, it will maintain the user's current position. (new to version 4.0)
- **ShowBottomPager**: A true/false value which determines if the Top paging control will be visible. This optional attribute is True by default. If both ShowBottomPager and ShowTopPager are set to False, no paging controls will be shown but the results will still be paged. In other words, only the first page of results will be shown without any visible mechanism to navigate to other pages.
  
  (Added in version 1.2)
- **ShowFirstLast**: A true/false value which determines if the First Page and Last Page navigation links should be used. The default value is False.
- **ShowPrevNext**: A true/false value which determines if the Previous Page and Next Page navigation links should be used. The default value is True.
- **ShowTopPager**: A true/false value which determines if the Bottom paging control will be visible. This optional attribute is True by default. If both ShowBottomPager and ShowTopPager are set to False, no paging controls will be shown but the results will still be paged. In other words, only the first page of results will be shown without any visible mechanism to navigate to other pages.
  
  (Added in version 1.2)
- **Display Template**: Within the `<Pager> </Pager>` tags, you can use HTML and special pager tokens to arrange and style the components of the pager. See the Example for one possibility. Below are the tokens you can use:
  
  - `{pageNumber}`: Will be replaced with the current page number.
  - `{ pageCount }`: Will be replaced with the total number of pages.
  - `{ pager }`: Will be replaced with the navigation links (First, Previous, Next, Last, and the page number links)

Example

```xml
<modx:Template ...>
...

  <Pager PageSize="15" PageNumCssClass="CommandButton"
    FirstPageCaption="[First]" LastPageCaption="[Last]">
    <table>
      <tr>
        <td>Page <strong>{pageNumber}</strong> of { pageCount}</td>
        <td align="right">{ pager }</td>
      </tr>
    </table>
  </Pager>

Back to top
The SearchSort tag can only be used within a Template tag or DataList tag. The tag defines how the search and sort panel in the view will look and function. The SearchSort tag attributes provide the information necessary to define how the components of the search and sort panel will look and function, while the tag's inner content, if specified, allow you to use HTML and SearchSort tokens to determine how the components of the search and sort panel are arranged.

**Syntax**

```xml
<SearchSort
    BackColor="color name!#dddddd"
    BorderColor="color name!#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    FilterExpression="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
   ForeColor="color name!#dddddd"
    Height="size"
    ReverseSortCssClass="string|Normal"
    ReverseSortText="string|Reverse"
    SearchBoxCssClass="string|NormalTextBox"
    SearchButtonCssClass="string|CommandButton"
    SearchButtonText="string|Search"
    SearchLabelCssClass="string|Normal"
    SearchLabelText="string|Search"
    SortButtonCssClass="string|CommandButton"
    SortButtonText="string|Sort"
    SortFieldLabels="comma-delimited list of sort field labels"
    SortFieldListCssClass="string|NormalTextBox"
    SortFieldNames="comma-delimited list of field names"
    SortLabelCssClass="string|Normal"
    SortLabelText="string|Sort"
    Width="size">
    ...
    </SearchSort>
</xml:Template|DataList>
```

**Remarks**

- **FilterExpression**: Essentially the "WHERE" clause of a SELECT query without the WHERE. It is executed when the user clicks the Search button. You use the placeholder `{0}` to represent the value entered by the user. For example: `FilterExpression="FirstName LIKE '%{0}%"`. In this example, the `{0}` will be replaced by the value entered by the user in the Search Box. If the user enters "John" then the resulting expression will be: "FirstName LIKE 'John%'".
- **ReverseSortCssClass**: The Cascading Style Sheet (CSS) class name to associate with the Reverse Sort checkbox. The default value is "Normal"
- **ReverseSortText**: The text to use for the Reverse Sort checkbox caption. The default value is "Reverse".
- **SearchBoxCssClass**: The Cascading Style Sheet (CSS) class name to associate with the search phrase input box. The default value is "NormalTextBox"
- **SearchButtonCssClass**: The Cascading Style Sheet (CSS) class name to associate with the button that initiates the Search. The default value is "CommandButton"
- **SearchButtonText**: The caption to use for the Search Button.
- **SearchLabelCssClass**: The Cascading Style Sheet (CSS) class name to associate with the Search Label. In most cases, the Search Label precedes the Search Box to identify the purpose of the box. The default value is "Normal"
- **SearchLabelText**: The text to use in the Search Label. In most cases, the Search Label precedes the Search Box to identify the purpose of the control. The default value is "Search"
- **SortButtonCssClass**: The Cascading Style Sheet (CSS) class name to associate with the button that, when pressed, executes the sort. The default value is "CommandButton".
- **SortButtonText**: The text to use for the button that, when pressed, executes the sort. The default value is "Sort"
- **SortFieldLabels**: Many times, the names you use for fields in your data source aren't user-friendly. Supply a comma-delimited list of captions to use in the sort field list control.
- **SortFieldListCssClass**: The Cascading Style Sheet (CSS) class name to associate with the list control containing the list of fields the user can sort on. The default value is "NormalTextBox"
- **SortFieldNames**: The names of the fields in your data source that can be sorted on. This is a comma-delimited list of names.
- **SortLabelCssClass**: The Cascading Style Sheet (CSS) class name to associate with the Sort Label. In most cases, the Sort Label precedes the sort list control to identify the purpose of the control.
- **SortLabelText**: The text to use in the Sort Label. In most cases, the Sort Label precedes the sort list control to identify the purpose of the control.
- **Display Template**: Within the `<SearchSort>` tags, you can use HTML and special tokens to arrange and style the components of the search/sort panel. See the Example for one possibility. Below are the tokens you can use:
  - `{SearchButton}`: Will be replaced with the button used to initiate the search.
  - `{SearchBox}`: Will be replaced with the input box used to enter the search phrase.
  - `{SearchLabel}`: Will be replaced by the label used to identify the purpose of the search box and search button.
  - `{SortLabel}`: Will be replaced by the label used to identify the purpose of the sort list control and button.
  - `{SortFieldList}`: Will be replaced by the list control containing the list of fields the user can sort on.
  - `{ReverseSort}`: Will be replaced by the checkbox that, when ticked, will cause the results to be sorted in reverse order.
  - `{SortButton}`: Will be replaced by the button used to initiate the sort.

```xml
Example
<mod:template ...>
  ...
  <SearchSort FilterExpression="FirstName LIKE '%(0)%'"
    SearchLabelText="Search For:" SearchBoxText="GO"
    SortFieldNames="FirstName,LastName,Zip"
    SortFieldLabels="First Name, Last Name, Zip Code">
    <table>
      <tr>
        <td><strong>{SearchLabel}</strong></td>
        <td align="right">
          <strong>{SortLabel}</strong> {SortFieldList} {SortButton} Reverse {ReverseSort}
        </td>
      </tr>
    </table>
  </SearchSort>
  ...
</mod:template>
```

**Syntax Remarks Example**
The ToggleButton tag renders as a push-button at run-time. It provides you with a simple interface for leveraging jQuery functionality. Simply set a few attributes and you’re done—no scripting necessary.

Note: This tag leverages and assumes the jQuery library is included in the page.

**Syntax**

```xml
<xmod:ToggleButton
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="HotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    Speed="Slow|Normal|Fast|integer"
    Style="string"
    Target="string"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size" />
```

**Remarks**

- **Usage:** The XMod Pro Toggle controls work in conjunction with jQuery. They enable you to leverage jQuery without having to write any Javascript. Because of this, you must ensure that jQuery has been included in the page. If you are using DNN 5 or later, the library is usually included in the page without any effort on your part. If not, use the `<xmod:scriptblock>` tag to include the library. The Target attribute identifies the element in the page whose visibility you’d like to toggle. The optional Speed attribute can be used to achieve a fading effect.

- **AccessKey:** In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)

- **BackColor:** Color of the background of the control.
- **BorderColor:** Color of the border around the control.
- **BorderStyle:** Style of the border around the control.
- **BorderWidth:** Width of the border around the control, specified in units.
- **CssClass:** Name of the Cascading Style Sheets (CSS) class used to style this control.
- **Font Properties:** A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)
- **ForeColor:** Sets the foreground color (typically the color of the text) of the control.
- **Height:** Height of the control, specified in units.
- **Speed:** The rate at which the element becomes visible or is hidden. When specified, you can cause the element to fade in or out when it is toggled. HOWEVER, if speed is defined, the target must either be a block element or have its CSS style "display" set to "inline" explicitly. jQuery 1.2.6 (and maybe later versions) sets "display" to "block" to make the object visible, regardless of whether it is an inline element - unless it has been set to inline beforehand. It’s possible this behavior may change in later versions of jQuery.

Valid values are:
- **Slow**: the element fades in/out at a slow rate of speed - approximately .6 seconds.
- **Normal**: the element fades in/out at a normal (medium) rate of speed - approximately .4 seconds.
- **Fast**: the element fades in/out at a fast rate of speed - approximately .2 seconds.
- **Numeric Value**: You can control the rate of speed more exactly by specifying a number. The number represents the number of milliseconds the transition should take. So, specifying 100 would cause the transition to occur in 1 tenth of a second. 500 would be half a second, and 1000 would be one full second.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Target**: A jQuery "selector" that identifies the element(s) you would like to toggle. To select an element by its ID, use the # selector. So, to select an element with the client ID of "divMyResults" (without the quotes), the Target attribute would be "/divMyResults" (again, without the quotes). To select all DIV elements with the class name of "MyResults", you would use the period selector (. - "div.MyResults")
- **Text**: The caption that will be displayed on the control.
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.

**Example**

```html
<div>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <!-- EMPLOYEES TEMPLATE -->
        <xmod:template id="Employees">
          <listdatasource commandtext="SELECT EmployeeId, FirstName, LastName, Evaluation FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
            <parameter name="DepartmentId" alias="DepartmentId"/>
          </listdatasource>
          <headertemplate>
            <p>Employees</p>
          </headertemplate>
          <itemtemplate>
            <div style="text-align: middle;">
              <strong>[[FirstName]] [[LastName]]</strong>
              <xmod:togglebutton text="View Employee Evaluation" target='[[Join("#divEvaluation_{{0}}",[[EmployeeId]])]]' speed="Fast"/>
            </div>
            <div id="divEvaluation_{{EmployeeId}}">
              <p>[[Evaluation]]</p>
            </div>
          </itemtemplate>
          <detailtemplate>
            <h1>Employee Profile</h1>
            <h3>[[FirstName]] [[LastName]]</h3>
            <h4>Biography: </h4>
            <div>[[Bio]]</div>
          </detailtemplate>
        </xmod:template>
      </td>
    </tr>
  </table>
</div>
```
The <xmod:ToggleImage> tag renders as a clickable image at run-time. It provides you with a simple interface for leveraging jQuery functionality. Simply set a few attributes and you're done - no scripting necessary.

**Syntax**

```xml
<xmod:ToggleImage
  AlternateText="string"
  BackColor="#dddddd"
  BorderColor="#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="#dddddd"
  Height="size"
  ImageAlign="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbsBottom|AbsMiddle|TextTop"
  ImageUrl="url"
  Speed="Slow|Normal|Fast"
  Target="string"
  ToolTip="string"
  Visible="True|False"
  Width="size" />
```

**Remarks**

- **Usage:** The XMod Pro Toggle controls work in conjunction with jQuery. They enable you to leverage jQuery without having to write any Javascript. Because of this, you must ensure that jQuery has been included in the page. If you are using DNN 5 or later, the library is usually included in the page without any effort on your part. If not, use the `<xmod:scriptblock>` tag to include the library. The **Target** attribute identifies the element in the page whose visibility you’d like to toggle. The optional **Speed** attribute can be used to achieve a fading effect.
- **AccessKey:** In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).
- **AlternateText:** Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.
- **BackColor:** Color of the background of the control.
- **BorderColor:** Color of the border around the control.
- **BorderStyle:** Style of the border around the control.
- **BorderWidth:** Width of the border around the control, specified in units.
- **CssClass:** Name of the Cascading Style Sheets (CSS) class used to style this control.
- **Font Properties:** A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor:** Sets the foreground color (typically the color of the text) of the control.
- **Height:** Height of the control, specified in units.
- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application’s root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your local host development machine and "~/images/myimage.gif" on your production server.
- **Speed**: The rate at which the element becomes visible or is hidden. When specified, you can cause the element to fade in or out when it is toggled. HOWEVER, if speed is defined, the target must either be a block element or have its CSS style "display" set to "inline" explicitly. jQuery 1.2.6 (and maybe later versions) sets "display" to "block" to make the object visible, regardless of whether it is an inline element - unless it has been set to inline beforehand. It’s possible this behavior may change in later versions of jQuery.

Valid values are:
- **Slow**: the element fades in/out at a slow rate of speed - approximately .6 seconds.
- **Normal**: the element fades in/out at a normal (medium) rate of speed - approximately .4 seconds.
- **Fast**: the element fades in/out at a fast rate of speed - approximately .2 seconds.
- **Numeric Value**: You can control the rate of speed more exactly by specifying a number. The number represents the number of milliseconds the transition should take. So, specifying 100 would cause the transition to occur in 1 tenth of a second. 500 would be half a second, and 1000 would be one full second.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Target**: The HTML ID of the element that will be toggled. The element can be any standard HTML element on the page. The element does not have to reside in the XMod Pro module instance.
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.

---

### Example

```html
<div>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <!-- EMPLOYEES TEMPLATE -->

        <xmod:template id="Employees">
          <listdatasource commandtext="SELECT EmployeeId, FirstName, LastName, Evaluation FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId" alias="DepartmentId/>
          <parameter name="DepartmentId" alias="DepartmentId"/>
        </listdatasource>
        <headertemplate>
          <p>Employees</p>
        </headertemplate>
        <itemtemplate>
          <div style="text-align: middle;">
            <strong>{{FirstName}} {{LastName}}</strong></div>
            <xmod:ToggleImage alternatetext="New Employee" imageurl="/~/images/add.gif"
              text="View Employee Evaluation" target='[[Join("divEvaluation_",[[EmployeeId]])]]'
              speed="Fast" />
          </div>
          <div id="divEvaluation_{{EmployeeId}}">
            <p>[[Evaluation]]</p>
          </div>
        </itemtemplate>
      </td>
    </tr>
  </table>
</div>
```
The ToggleLink tag renders as a clickable image at run-time. It provides you with a simple interface for leveraging jQuery functionality. Simply set a few attributes and you're done - no scripting necessary.

Note: This tag leverages and assumes the jQuery library is included in the page.

Syntax

```xml
<xmod:ToggleLink
    BackColor="color name|#dddddd"
    BORDERCOLOR="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    Speed="Slow|Normal|Fast|Integer"
    Style="string"
    Target="string"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size"
/>
```

Remarks

- **Usage**: The XMod Pro Toggle controls work in conjunction with jQuery. They allow you to leverage jQuery without having to write any Javascript. Because of this, you must ensure that jQuery has been included in the page. If you are using DNN 5 or later, the library is usually included in the page without any effort on your part. If not, use the `<xmod:scriptblock>` tag to include the library. The Target attribute identifies the element in the page whose visibility you'd like to toggle. The optional Speed attribute can be used to achieve a fading effect.
- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **Speed**: The rate at which the element becomes visible or is hidden. When specified, you can cause the element to fade in or out when it is toggled. HOWEVER, if speed is defined, the target must either be a block element or have its CSS style "display" set to "inline" explicitly. jQuery 1.2.6 sets "display" to "block" to make the object visible, regardless of whether it is an...
inline element - unless it has been set to inline beforehand. It's possible this behavior may change in later versions of jQuery.

Valid values are:
- **Slow**: the element fades in/out at a slow rate of speed - approximately .6 seconds.
- **Normal**: the element fades in/out at a normal (medium) rate of speed - approximately .4 seconds.
- **Fast**: the element fades in/out at a fast rate of speed - approximately .2 seconds.
- **Numeric Value**: You can control the rate of speed more exactly by specifying a number. The number represents the number of milliseconds the transition should take. So, specifying 100 would cause the transition to occur in 1 tenth of a second. 500 would be half a second, and 1000 would be one full second.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Target**: The HTML ID of the element that will be toggled. The element can be any standard HTML element on the page. The element does not have to reside in the XMod Pro module instance.
- **Text**: The caption that will be displayed on the control.
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in [units](#).

**Example**

```xml
<div>
  <table width="100%">
    <tr>
      <td width="250" valign="top">
        <![CDATA[
        !-- EMPLOYEES TEMPLATE -->
        
        <xmod:template id="Employees">
          <listdatasource commandtext="SELECT EmployeeId, FirstName, LastName, Evaluation FROM XMPDemo_Employees WHERE DepartmentId = @DepartmentId">
            <parameter name="DepartmentId" alias="DepartmentId"/>
          </listdatasource>
          <headertemplate>
            <p>Employees</p>
          </headertemplate>
          <itemtemplate>
            <div style="text-align: middle;">
              <strong>[[FirstName]] [[LastName]]</strong>
              <xmod:Togglelink text="View Employee Evaluation"
                  target='[[Join("#divEvaluation_{0}",[[EmployeeId]])]]'
                  speed="Fast"/>
            </div>
            <div id="divEvaluation_{[EmployeeId]}">
              <p>[[Evaluation]]</p>
            </div>
          </itemtemplate>
        </xmod:template>
      </td>
    </tr>
  </table>
</div>
```
Form Controls

<Action>

Syntax Remarks Example

New to Version 4.0! The Action tag allows you to execute custom server-side functionality after the form has been successfully submitted.

Syntax

<Action
  Assembly="Name of DLL containing action - do not include the .dll extension"
  Namespace="Namespace pointing to the action class"
 />

Remarks

- The Action tag is only executed if the form has been successfully submitted. If there is a validation error or an error is thrown from the database, the action will not be performed. Note that the Action is executed after the SubmitCommand is executed, so it cannot impact the database call itself.

- **Order Is Important**: Action tags are executed sequentially, so the order they appear within the form can be important. As an example, if one action fails with an error, all actions prior to the failed action will have executed. Those that occur after the failed action will not be executed. Additionally, some actions may have the ability to modify form values (this modification occurs after any form data has been sent to the database) - i.e. process form values, do calculations on them, transform them, even add and remove values from the list. Those changes will affect any Action tags that are executed downstream that use Field tokens.

- **Using Tokens**: Unlike most form tags, which evaluate their tokens when the form is loaded, Action tags evaluate their tokens when they're executed (after successful form submission). This means that values passed into the form such as URL parameters will need to be stored in a hidden form control (typically a TextBox with its Visibility property set to False). On the other hand, this enables Action tags to use Field tokens as their property values so these tags can use values input by the user in the form.

- **Assembly**: Required. The file name of the DLL in your /bin directory that contains the action class.

- **Namespace**: Required. The full namespace that points to the action class in your DLL.

- **Creating a Custom Action**: If you are a .NET programmer, you can create your custom action by performing the following steps:

  1. Create a Class Library project in Visual Studio
  2. Reference the DotNetNuke.dll, KnowBetter.XModPro.Common.dll files in your project
  3. Create the class that will contain your custom action
  4. Add a method called "Execute" and set it to accept a PortalModuleBase object and KnowBetter.XModPro.Common.XItem object like so:

```csharp
        ' Do something very cool and interesting here.
        ' pmb will contain useful environmental data like the current PortalId and current UserId.
```
' xi will contain a list of values from the form just submitted. Note: They are passed by reference so you can modify the values
' and add values to the list. At this point, the data will have been saved to the database already. However, your modified values
' can be used by subsequent <Action> tags as well as <Email> tags and user redirection.
End Sub

5. Compile and place the DLL in your site's /bin directory.

Example

<AddForm>
<SubmitCommand CommandText="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)"/>
<table>
<tr>
<td>
  <Label For="txtFirstName" Text="First Name"/>
  <TextBox Id="txtFirstName" DataField="FirstName" DataType="string"/>
</td>
</tr>
<tr>
<td>
  <Label For="txtLastName" Text="Last Name"/>
  <TextBox Id="txtLastName" DataField="LastName" DataType="string"/>
</td>
</tr>
<tr>
<td colspan="2">
  <AddButton Text="Add"/> <CancelButton Text="Cancel"/>
</td>
</tr>
</table>
</AddForm>

Syntax

The AddButton tag renders as a push-button at run-time. When clicked, the form executes the <SubmitCommand> associated with the <AddForm>.

Syntax

<AddButton

AccessKey="string"
BackColor="color name|#dddddd"
BorderColor="color name|#dddddd"
BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
BorderWidth="size"
CssClass="string"
Font-Bold="True|False"
Font-Italic="True|False"
Font-Names="string"
Font-Overline="True|False"
Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
Font-Strikeout="True|False"
Remarks

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control.

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in [units](#). (CSS)

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in [units](#).

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing..

- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes. Field tokens may be used in the redirect attribute. However, function tokens such as [[Portal:ID]], [[Join()]], [[User:ID]], etc. cannot be used. When field tokens are used, they are URL Encoded. New to version 4.0: You can use a period (.) for the Redirect property's value. The period acts as shortcut to redirect to the current page.

- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"

  IMPORTANT: When using "Post", the ID that you supply for your form controls determine the name of the field that is posted to the target URL, not the DataField.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Width of the control in **units**.

---

**Example**

```xml
<addform>
    <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
    <table>
        <tr>
            <td>
                <label for="txtFirstName" text="First Name" />
                <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
            </td>
        </tr>
        <tr>
            <td>
                <label for="txtLastName" text="Last Name" />
                <textbox id="txtLastName" datafield="LastName" datatype="string" />
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <addbutton text="Add"/> <cancelbutton text="Cancel"/>
            </td>
        </tr>
    </table>
</addform>
```

---

**Syntax**

**Forms in XMod Pro** are defined using HTML and special XMod Pro tags. XMod Pro provides you with the ability to define separate forms to use for adding a record (using the `<addform>` tag) and for editing a record using the `<editform>` tag. To use third party libraries containing XMod Pro-compatible controls, you use the `<register>` tag. This makes the library available to both your add and edit form.
Syntax

<register ...
<register ...
...
/AddForm ClientName="string" ScrollToTop="True|False">
  <Variable ...
  ...
  <SelectCommand.../>
  <SubmitCommand.../>
  <ControlDataSource .../>
</AddForm>
/EditForm ClientName="string">
  <Variable ...
  ...
  <SelectCommand.../>
  <SubmitCommand.../>
  <ControlDataSource .../>
</EditForm>

/AddSuccessTemplate
  <ItemTemplate>HTML, Field Tokens, <xmod:Redirect> tags, and Continue Button/Image/Link controls</ItemTemplate>
</AddSuccessTemplate>

/EditSuccessTemplate
  <ItemTemplate>HTML, Field Tokens, <xmod:Redirect> tags, and Continue Button/Image/Link controls</ItemTemplate>
</EditSuccessTemplate>

Remarks

Each form definition can contain only one <AddForm> and one <EditForm> tag. Though, it is possible to define just the <AddForm> or <EditForm> should you only need that form type. All your control tags, HTML tags, and text must fall within the <AddForm> and <EditForm> tags. Your HTML and tags should be XHTML compliant for best results.

/AddForm> and <EditForm>

- **ClientName**: This attribute is optional. When used, it enables you to specify the name to use for your form within JavaScript calls. It is used with XMod Pro's Javascript helper function to quickly and easily get the client ID of elements within an XMod Pro form. For more information see the Using Javascript help topic.

- **ScrollToTop**: This attribute is optional. When set to True, when the user clicks the AddButton or UpdateButton, the form will position the browser window at the top of the page rather than maintaining the scroll position. This is helpful when you're using a long form and you want to avoid the visual jump that ensues between the time the page loads (at the top of the page) and the time the browser jumps to the previous scroll position. In the right circumstances, it can mitigate that jarring visual jump. New to version 4.1.

/<SelectCommand> and <SubmitCommand>

Form tags rely on "command" tags (<SelectCommand>, <SubmitCommand>) to perform their data-related functions. The <SelectCommand> when specified, is used to populate the form with data when it first loads. The <SubmitCommand> is executed when the user presses or clicks the Add/Update button in the form. These commands work in conjunction with the "datafield" and "datatype" attributes of the form control tags. The datafield attribute values match the parameter names (i.e. @ParamName) in the commands. The "datatype" attribute enables XMod Pro to pass the control's value properly to your command. The attributes for these tags are described below:
- **CommandText**: The SQL command to execute to return the data. This attribute is required.
- **ConnectionString**: The database connection string for the datasource from which the data will be retrieved. This attribute is optional. XMod Pro will use the connection string for the current DNN database by default. To use connection strings stored in the web.config file, use the [[ConnectionString:connectionStringName]] where connectionStringName is the name assigned to the connection string in the web.config file
- **<Parameter>**: These child tags enable the <SelectCommand> to accept values passed as POST or GET parameters or parameters passed from <xmod:deletebutton>, <xmod:editbutton>, and similar tags. Parameters have the following attributes:
  - **Name**: The name of the parameter. This must match the $name parameter in the CommandText
  - **Value**: The value of the parameter (optional). This attribute allows you to hard-code a value for the parameter or use the value of a token (like [[Url:MyParam]]) - see the example code later in this topic
  - **DefaultValue**: This attribute value will be used if no Value is found for the parameter
  - **DataType**: This attribute that enables you to ensure the parameter's value is of a specific type (as an aid in preventing SQL injection attacks).

**<ControlDataSource>**

Use this optional tag if you want to bind a list-based control to a set of data. [More Information]

**Using [[Field]] Tokens in Your Form**

Beginning in version 1.3, XMod Pro now supports the ability to use data from your <SelectCommand> in the attributes of your form controls. This allows you to retrieve values from your database or from passed-in parameters or hard-coded values in your forms to further enhance their dynamic nature. For instance, if a customer is ordering a product, your <SelectCommand> can look that product up in the inventory database and set the MaximumValue of the Range Validator so that it prevents the user from ordering a quantity greater than the stock you have in the warehouse.

Using [[Field]] tokens is easy. They follow the same rules as [[Field]] tokens in templates. Example:

```xml
<AddForm>
  <SelectCommand CommandText="SELECT StockOnHand FROM Inventory
                      WHERE ProductId = @ProductId">
    <Parameter Name="ProductId" Value='[[Url:pid]]' DataType="Int32" />
  </SelectCommand>
  ...
</AddForm>
```

<TextBox Id="txtQuantity" DataField="Qty" DataType="Int32"/>

```xml
<Validate type="Range" target="txtQuantity" minimumvalue="1"
          maximumvalue="[[StockOnHand]]"
          message='[[Join("You can only order between 1 and [0] tickets",[[StockOnHand]])]]'
          type="Integer"/>
```

**<AddSuccessTemplate>, <EditSuccessTemplate>**

These tags are **optional**. They provide you with the ability to display a "thank you" or a "completed" type of message to your user after successful submission of the form. The AddSuccessTemplate tag is used when the AddForm is submitted. The EditSuccessTemplate tag is used when the EditForm is submitted. When displayed, the content of the tag's <ItemTemplate> tag will be displayed in place of the Add/Edit form.

**Enabling Your Users to Continue After Successful Form Submission**: The AddSuccessTemplate and EditSuccessTemplates tags accept <xmod:ContinueButton>, <xmod:ContinueImage>, and <xmod:continueLink> controls. These render as a button, image, or link, respectively. They act as "OK" buttons. When clicked, these buttons will take the user to the page/view they would ordinarily see after submitting their form without the success message. Of course, you can also send the user to the URL of your choice. Each of these buttons has a Redirect attribute that takes a destination URL. When used, the user will be redirected to that URL. NOTE: if you want to POST values to a destination URL, you should use the <xmod:redirectTo> tag.

You can use text, HTML, and [[Field]] tokens to craft your success message. Note that the [[Field]] tokens will be replaced at run-time by the values in the form's data-bound fields, rather than values actually saved to the database.
Displaying the return value and/or Output parameter value from a stored procedure (new to v.4): If your form's SubmitCommand calls a stored procedure that returns a value or sets the value of an output parameter, you can display or otherwise use that value in your success template.

IMPORTANT: When calling stored procedures in your SubmitCommand, you MUST declare all the parameters you're passing to the stored procedure. Because you can have controls on your form that you may not pass to the database (i.e. using an entered value in an Email or Redirect) XMod Pro cannot automatically generate those stored procedure parameters for you.

1. Modify your <SubmitCommand> to call your stored procedure and set its CommandType property to.StoredProcedure:

   `<SubmitCommand CommandText="addContact" CommandType=".StoredProcedure"/>
   
2. For each form field you want to pass to your stored procedure, add a <Parameter> tag to your <SubmitCommand>:

   `<SubmitCommand CommandText="addContact" CommandType=".StoredProcedure">
   
   <Parameter Name="FirstName"/>
   
   <Parameter Name="LastName"/>
   
   </SubmitCommand>
   
3. Add parameters for your Output and/or Return Value. Be sure to set the Direction property accordingly:

   `<SubmitCommand CommandText="addContact" CommandType=".StoredProcedure">
   
   <Parameter Name="FirstName"/>
   
   <Parameter Name="LastName"/>
   
   <Parameter Name="retVal" Direction="ReturnValue"/>
   
   <Parameter Name="newID" Direction="Output" DataType="int32"/>
   
   </SubmitCommand>
   
NOTES: Your parameter name(s) should be unique among all your form's controls (i.e. their DataField properties). Also, if you set a DataType for your return value/output parameter, ensure that the returned value returned matches it. Otherwise an error will be thrown. Also, if you're passing textual data types like varchar or nvarchar, please ensure that your <Parameter> tags specify a Size property - otherwise, only the 1st character will be returned in an output parameter.

4. If your stored procedure will pass back friendly error messages (like duplicate record messages), set that up. Your stored procedure must have an OUTPUT parameter whose name is @ERROR (all caps). You can handle this by adding a ERROR parameter and adding a <Validate Type="Database"/> and a <ValidationSummary/> tag if your form doesn't already have one. Any errors from the database will be displayed in the ValidationSummary tag.

   `<SubmitCommand CommandText="addContact" CommandType=".StoredProcedure">
   
   <Parameter Name="FirstName"/>
   
   <Parameter Name="LastName"/>
   
   <Parameter Name="retVal" Direction="ReturnValue"/>
   
   <Parameter Name="newID" Direction="Output" DataType="int32"/>
   
   <Parameter Name="ERROR" Direction="Output" DataType="string"/>
   
   </SubmitCommand>
   
   ...<AddButton Text="Add"/>
   
   <Validate Type="Database"/>
   
   <ValidationSummary/>
   
5. In your success template, use your return value as you would a field token:

   `<AddSuccessTemplate>
   
   This is my return value: [[retVal]]<br/>
   
   </AddSuccessTemplate>`
Variable tags, new to version 2.7, are optional tags you can use to provide additional data to your form. Variables are not rendered to the form but are available to your data commands and controls. For instance, you might use a variable to store the portal's ID. Then, in your SubmitCommand, you can use that variable to store the Portal ID in the record along with other form values. Another possibility would be to use the <Text> tag to display information to the user from Variables or, perhaps, to configure a Javascript function in your form.

Syntax: <Variable Name="uniqueName" Value="variableValue" DataType="string|Int32|Int64|boolean|..."/>

The name you provide the variable is used just like column names are used with your data commands. You can bind controls to the variable by specifying the variable's Name as the control's DataField property. The Value can be a hard-coded value or one of XMod Pro's tokens - like [[Portal:ID]]. When using a token, ensure that the Value is delimited with single-quotes rather than double quotes. Finally, the DataType property allows you to specify the type of value the variable contains. The default data type is String.

Variable tags are typically placed at the top of each form. Sample:

```
<AddForm>
  <Variable Name="thePortalId" Value='[[Portal:ID]]' DataType="Int32" />
  <SubmitCommand CommandText="INSERT INTO MyTable (Name,PortalId) VALUES(@Name,@thePortalId)" />
  ...
  <TextBox id="txtName" DataField="Name" DataType="String" />
  ...
</AddForm>
```
### Example

```xml
<addform>
  <selectcommand commandtext="SELECT @FirstName AS FirstName, @LastName AS LastName, 'AZ' AS StateId">
    <parameter name="FirstName" value='[[User:FirstName]]' defaultvalue=""/>
    <parameter name="LastName" value='[[User:LastName]]' defaultvalue=""/>
  </selectcommand>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName, StateId)
VALUES(@FirstName, @LastName, @StateId)"/>
  <controldatasource id="dsStates" connectionstring="(your connection string here)"
    commandtext="SELECT StateName, StateId FROM States ORDER BY StateName ASC" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="FirstName" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="LastName" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="ddlState" text="State" />
        <dropdownlist id="ddlState" datafield="StateId" datatype="int32"
          datasourceid="dsStates" datatextfield="StateName" datavaluefield="StateId"/>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <addbutton text="Add"/>
        <cancelbutton text="Cancel"/>
      </td>
    </tr>
  </table>
  <email to="me@mysite.com" from="server@mysite.com" subject="New User Added" format="html">
    <content>
      A new user has been added:
      <br/>
      FirstName: [[FirstName]]
      <br/>
      LastName: [[LastName]]
    </content>
  </email>
</addform>

<editform>
  <selectcommand commandtext="SELECT * FROM Users WHERE UserId = @UserId AND PortalId = @PID">
    <parameter name="PID" value='[[Portal:ID]]'/>
  </selectcommand>
  <submitcommand commandtext="UPDATE Users SET FirstName=@FirstName,
LastName=@LastName,
StateId=@StateId
WHERE UserId=@UserId"/>
  <controldatasource id="dsStates" connectionstring="(your connection string here)"
    commandtext="SELECT StateName, StateId FROM States ORDER BY StateName ASC" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="FirstName" />
      </td>
    </tr>
  </table>
</editform>
```
The AddImage tag renders as a clickable image at run-time. When clicked, the form executes the <SubmitCommand> associated with the <AddForm>.
Syntax

```
<AddImage
    AccessKey="string"
    AlternateText="string"
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyles="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    ImageAlign="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbsBottom|AbsMiddle|TextTop"
    ImageUrl="url"
    OnClientClick="string"
    Redirect="url"
    RedirectMethod="Get|Post"
    Style="string"
    TabIndex="integer"
    ToolTip="string"
    Visible="True|False"
    Width="size"
/>```

Remarks

- The add image button should only be used in `<AddForm>` tags. It's purpose is to initiate the `<SubmitCommand>` associated with the `<AddForm>`

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in units.

- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.

- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application’s root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing.

- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes. Field tokens may be used in the redirect attribute. However, function tokens such as [[Portal:ID]], [[Join()]], [[User:ID]], etc. cannot be used. When field tokens are used, they are URL Encoded. New to version 4.0: You can use a period (.) for the Redirect property's value. The period acts as shortcut to redirect to the current page.

- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"

  IMPORTANT: When using "Post", the ID that you supply for your form controls determine the name of the field that is posted to the target URL, not the DataField.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Width of the control in units.

---

**Example**

```xml
<addform>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="Last Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <AddImage AlternateText="Add" ImageUrl="~/images/add.gif" />
        <CancelImage AlternateText="Cancel" ImageUrl="~/images/cancel.gif" />
      </td>
    </tr>
  </table>
</addform>
```
The AddLink tag renders as a hyperlink at run-time. When clicked, the form executes the `<SubmitCommand>` associated with the `<AddForm>`,

**Syntax**

```xml
<AddLink
      BackColor="color name|#dddddd"
      BorderColor="color name|#dddddd"
      BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
      BorderWidth="size"
      CssClass="string"
      Font-Bold="True|False"
      Font-Italic="True|False"
      Font-Names="string"
      Font-Overline="True|False"
      Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
      Font-Strikeout="True|False"
      Font-Underline="True|False"
      ForeColor="color name|#dddddd"
      Height="size"
      OnClientClick="javascript"
      Redirect="url"
      RedirectMethod="Get|Post"
      Style="string"
      Text="string"
      ToolTip="string"
      Visible="True|False"
      Width="size"/>
```

**Remarks**

- The AddLink tag should only be used in `<AddForm>` tags. Its purpose is to initiate the `<SubmitCommand>` associated with the `<AddForm>`.

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control.

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units.

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)
• **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

• **Height**: Height of the control, specified in units.

• **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return `true` then the control will perform its normal processing.

• **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes. Field tokens may be used in the redirect attribute. However, function tokens such as `[[Portal:ID]], [[Join()]], [[User:ID]],` etc. cannot be used. When field tokens are used, they are URL Encoded. New to version 4.0: You can use a period (.) for the Redirect property's value. The period acts as shortcut to redirect to the current page.

• **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"
  IMPORTANT: When using "Post", the ID that you supply for your form controls determine the name of the field that is posted to the target URL, not the DataField.

• **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

• **Visible**: Determines if the control is visible (true) or hidden (false)

• **Width**: Width of the control in units.

**Example**

```xml
<addform>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />  
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="Last Name" />  
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <td colspan="2">
      <AddLink Text="Add" /> <CancelLink Text="Cancel" />
    </td>
  </table>
</addform>
```

**AddToRoles**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Remarks</th>
<th>Example</th>
</tr>
</thead>
</table>

Back to top
New to Version 4.0! The AddToRoles tag will add a user to one or more DotNetNuke security roles after the form has been successfully submitted.

Syntax

```xml
<AddToRoles
  Culture="locale-id"
  EndDate="date"
  If="basic conditional equality expression"
  RoleNames="comma-delimited list of DNN roles"
  StartDate="date"
  UserId="integer"
/>
```

Remarks

- The AddToRoles action is only executed if the form has been successfully submitted. If there is a validation error or an error is thrown from the database, the action will not be performed.

- **Order Is Important**: Action tags are executed sequentially, so the order they appear within the form can be important. As an example, if one action fails with an error, all actions prior to the failed action will have executed. Those that occur after the failed action will not be executed. Additionally, some actions may have the ability to modify form values (this modification occurs after any form data has been sent to the database) - i.e. process form values, do calculations on them, transform them, even add and remove values from the list. Those changes will affect any Action tags that are executed downstream that use Field tokens.

- **Using Tokens**: Unlike most form tags, which evaluate their tokens when the form is loaded, Action tags evaluate their tokens when they're executed (after successful form submission). This means that values passed into the form such as URL parameters will need to be stored in a hidden form control (typically a TextBox with its Visibility property set to False). On the other hand, this enables Action tags to use Field tokens as their property values so these tags can use values input by the user in the form.

- **Culture**: optional. Specify a locale id (e.g. en-us, en-gb, de-de, etc.) if the supplied Start/End date properties are being input in that locale. If omitted, the Start/End dates will be parsed in accordance with the system's current locale. Field tokens may be used to populate this property. New to version 4.1

- **EndDate**: optional. When specified, this is the date the user will be removed from the role - i.e. the Expiration Date. Field tokens may be used to populate this property. New to version 4.1

- **If**: optional. This is a basic equality expression. When it evaluates to true, the AddToRoles action will be executed. If it evaluates to false, the AddToRoles action will not be executed. This is a good way to add a user to a role only if some value on the form has been set. For instance, you could add someone to the Newsletter role only if they check a box on the form confirming that they want your newsletter. New to version 4.1

  ```xml
  <AddToRoles If='[[SignMeUp]] = True' RoleNames="Newsletter" UserId='[[uid]]' />
  ```

  **NOTE**: Comparisons are text-only and are not case-sensitive. You can test for equality using the "=" operator or inequality using the "<>" operator.

- **RoleNames**: Required. One or more DotNetNuke security role names you want to add the user to. If more than one role is specified, separate them with commas. Field tokens may be used to populate this property.
**StartDate**: optional. When specified, this is the date on which the user will be placed in the role - i.e. the Effective Date. Field tokens may be used to populate this property. New to version 4.1

**UserId**: Required. The unique numeric user identifier assigned by DotNetNuke to the user you want to add to a role. Field tokens may be used to populate this property.

### Example

In the example below, we're using a `<Variable>` tag to retrieve the current user's ID and make it available to the `<AddToRoles>` tag. The variable has a name of "uid" and we set the UserId property of the `<AddToRoles>` tag to `[[uid]]`.

```
<AddForm>
  <Variable Name="uid" Value='[[User:Id]]' />
  <SubmitCommand CommandText="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <AddToRoles RoleNames="Role1,Editors" UserId='[[uid]]' />
</AddForm>
```

**Syntax**

New to Version 4.0! The AddUser tag will register a user in the DNN site and optionally add that user to one or more DotNetNuke security roles after the form has been successfully submitted.

**IMPORTANT**: You MUST take care to properly validate user input. Additionally you should place this tag only on forms that are properly secured so that only users you intend to have access can use the form.

**Syntax**

```
<AddUser
  Approved="True|False"
  City="string"
  Country="string"
  DisplayName="string"
  Email="string"
  FirstName="string"
  LastName="string"
  Password="string"
```
PostalCode="string"
Region="string"
RoleNames="comma-delimited list of DNN roles"
Street="string"
Telephone="string"
Unit="string"
UpdatePasswordOnNextLogin="True|False"
Username="string">
  (NOTE: Property tags are optional)
  <Property Name="string" Value="string"/>
  ...Additional Property Tags as needed...
</AddUser>

Remarks

- The AddUser action is only executed if the form has been successfully submitted. If there is a validation error or an error is thrown from the database, the action will not be performed.

- **Order Is Important**: Action tags are executed sequentially, so the order they appear within the form can be important. As an example, if one action fails with an error, all actions prior to the failed action will have executed. Those that occur after the failed action will not be executed. Additionally, some actions may have the ability to modify form values (this modification occurs after any form data has been sent to the database) - i.e. process form values, do calculations on them, transform them, even add and remove values from the list. Those changes will affect any Action tags that are executed downstream that use Field tokens.

- **Using Tokens**: Unlike most form tags, which evaluate their tokens when the form is loaded, Action tags evaluate their tokens when they're executed (after successful form submission). This means that values passed into the form such as URL parameters will need to be stored in a hidden form control (typically a TextBox with its Visibility property set to False). On the other hand, this enables Action tags to use Field tokens as their property values so these tags can use values input by the user in the form.

- **Passing the New User ID Downstream**: If you want an action later in the sequence (like an `<AddToRoles>` or `<Redirect>` tag), the AddUser tag now passes a special value along with the other form values: __UserId (that’s 2 underscore characters followed by the word UserId). You can use this as if it were a field token. New to version 4.1.

- **Approved**: Optional. True or False. The default value is False. If set to True, the user will be auto-approved when the user record is created.

- **City**: Optional. The city in which the user resides.

- **Country**: Optional. The country in which the user resides.

- **DisplayName**: Optional. The name to display when displaying the user’s full name. If not supplied the control will attempt to combine the FirstName and LastName.

- **Email**: Required. The user’s email address.

- **FirstName**: Required. The user’s first name.

- **LastName**: Required. The user’s last name.

- **Password**: The password for the user account that will be created.
- **PostalCode**: Optional. The postal code (zip code in the US) associated with the user's address.

- **Property Tags**: These are optional child tags that allow you to specify one or more custom profile properties that will be set when the user is created.

- **Region**: Optional. The region (state in the US) where the user is located.

- **RoleNames**: Optional. One or more DotNetNuke security role names you want to add the user to. If more than one role is specified, separate them with commas.

- **Street**: Optional. The street address of the user.

- **Telephone**: Optional. The user's telephone number.

- **Unit**: Optional. The unit or apartment number associated with the user's address.

- **UpdatePasswordOnNextLogin**: True or False. The default value is False. When true, the user will be prompted to update his/her password when logging in next.

- **Username**: Required. A unique name that will be associated with this user account.

---

Example

```xml
<AddForm>
  <AddUser RoleNames="Role1,Editors" Email='[[Email]]'
    FirstName='[[FName]]' LastName='[[LName]]'
    Username='[[Username]]' Password='[[Password]]' />
</AddForm>
```

```html
<table>
  <tr>
    <td>
      <Label For="txtFirstName" Text="First Name" />
      <TextBox Id="txtFirstName" DataField="FName" DataType="string" />
      <Validate Type="Required" Target="txtFirstName" Text="***" Message="First Name is required." />
    </td>
  </tr>
  <tr>
    <td>
      <Label For="txtLastName" Text="Last Name" />
      <TextBox Id="txtLastName" DataField="LName" DataType="string" />
      <Validate Type="Required" Target="txtLastName" Text="***" Message="Last Name is required." />
    </td>
  </tr>
  <tr>
    <td>
      <Label For="txtEmail" Text="Email" />
      <TextBox Id="txtEmail" DataField="Email" DataType="string" />
      <Validate Type="Required" Target="txtEmail" Text="***" Message="An email address is required." />
      <Validate Type="Email" Target="txtEmail" Text="***" Message="Please enter a valid email address." />
    </td>
  </tr>
  <tr>
    <td>
      <Label For="txtUsername" Text="Username" />
      <TextBox Id="txtUsername" DataField="Username" DataType="string" />
      <Validate Type="Required" Target="txtUsername" Text="***" Message="Please enter a Username." />
    </td>
  </tr>
</table>
```
<AddForm>

<tr><td><Label For="txtPassword" Text="Password" /></td></tr>
<tr><td><Password Id="txtPassword" DataField="Password" DataType="string" /></td></tr>
<tr><td><Validate Type="Required" Target="txtPassword" Text="***" Message="A Password is required." /></td></tr>
<tr><td><Label For="txtReEnterPassword" Text="Password" /></td></tr>
<tr><td><Password Id="txtReEnterPassword" DataField="pw2" DataType="string" /></td></tr>
<tr><td><Validate Type="Required" Target="txtReEnterPassword" Text="***" Message="Please re-enter your password." /></td></tr>
<tr><td><Validate Type="Compare" Target="txtPassword" CompareTarget="txtReEnterPassword" Text="***" Message="Your passwords don't match" /></td></tr>
<tr><td colspan="2">
  <AddButton Text="Add" />
  <CancelButton Text="Cancel"/>
</td></tr>
<tr><td colspan="2">
  <ValidationSummary DisplayMode="BulletList" HeaderText="Errors:" CssClass="NormalRed" />
</td></tr>
</table>

<Back to top

<AjaxButton>

The AjaxButton tag renders as a push-button at run-time that, when clicked, will dynamically insert HTML returned from a URL into an element on the page - without a postback. This is a jQuery based control. It required jQuery be included in the page and that Javascript be enabled in the end-user's browser.
Syntax

```xml
<AjaxButton
    BackColor="color"|#dddddd"
    BorderColor="color"|#dddddd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color"|#dddddd"
    Height="size"
    LoadingCssClass="CSS class name"
    LoadingImageUrl="string"
    OnError="string - JS function to call on error"
    OnSuccess="string - JS function to call on success"
    Style="string"
    Target="jQuery element selector"
    Text="string"
    ToolTip="string"
    Url="url"
    Visible="True|False"
    Width="size" />
```

Remarks

- **Usage**: The XMod Pro Ajax button controls work in conjunction with jQuery. They enable you to leverage jQuery without having to write any Javascript. Because of this, you must ensure that jQuery has been included in the page. If you are using DNN 5 or later, the library is usually included in the page without any effort on your part. If not, use the `<scriptblock>` tag to include the library. You must specify the `Url` property and `Target` property. Optionally, you can specify the `LoadingImageUrl` and `LoadingCssClass`.

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in units.

- **LoadingCssClass**: A CSS class name to assign to the image which appears after the button has been clicked - to indicate content is being loaded. This property is ignored if no `LoadingImageUrl` is specified.
- **LoadingImageUrl**: A URL to the image file that will be displayed after the button is clicked - to indicate content is being loaded. You can use the tilde (~) character as a placeholder for the website root directory. The image will be displayed immediately after the button when the button is clicked. It will be removed from the page after successful completion of the AJAX call. If no image url is specified, no image will be displayed.

- **OnError**: You can optionally specify a Javascript function to call if there is an error in the AJAX call. Your function should accept the following parameters: jqXHR, textStatus, errorThrown. This should only contain a function name. An example would be:

  ```javascript
  OnError="myErrHandler"
  ``

  Elsewhere, you would define your Javascript function like:

  ```javascript
  function myErrHandler(jqXHR, textStatus, errorThrown) {
    alert("The following error occurred: " + textStatus);
  }
  ```

- **OnSuccess**: You can optionally specify a Javascript function to call when data is returned from the AJAX call. This overrides standard default processing of the AJAX call - which normally sets the HTML of the Target element. Instead, your function will be called and the returned data from the AJAX call will be passed to your function. This property should only contain a function name. An example might be:

  ```javascript
  OnSuccess="doSomethingCool"
  ``

  Elsewhere you would define your Javascript function like:

  ```javascript
  function doSomethingCool(data) {
    alert("The following data was returned:" + data);
  }
  ```

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Target**: A jQuery "selector" that identifies the element(s) that will contain the HTML returned from the Url. Note that content of the element will be replaced by the HTML. To select an element by its ID, use the # selector. So, to select an element with the client ID of "divMyResults" (without the quotes), the Target attribute would be "#divMyResults" (again, without the quotes). To select all DIV elements with the class name of "MyResults", you would use the period selector (.) - "div.MyResults".

- **Text**: The caption that will be displayed on the control.

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Width of the control in [units](#).
Example

```xml
<AddForm>
  <SubmitCommand CommandText="Update EmployeeReview SET Rating=@Rating" />
  <Label>Submit Rating for Employee</Label>
  <DropdownList Id="Rating" DataField="Rating" DataType="Int32">
    <ListItem Value="1">Poor</ListItem>
    <ListItem Value="2">Sub-Par</ListItem>
    <ListItem Value="3">Average</ListItem>
    <ListItem Value="4">Above Average</ListItem>
    <ListItem Value="5">Excellent</ListItem>
  </DropdownList>
  <AjaxButton Text="View Employee History" Url="mysite.com/history.aspx?eid=100"
              Target="#divHistory" />
  <div id="divHistory"></div>
  <CancelButton Text="Nevermind" />
</AddForm>
```

<ajaximage>

The `AjaxImage` tag renders as a push-button at run-time that, when clicked, will dynamically insert HTML returned from a URL into an element on the page - without a postback. This is a jQuery based control. It required jQuery be included in the page and that Javascript be enabled in the end-user's browser.

Syntax

```xml
<ajaximage
  AlternateText="string"
  BackColor="color name|#dcdcdd"
  BorderColor="color name|#dcdcdd"
  BorderSide="NotSet,None,Dotted,Dashed,Solid,Double,Grace,Outline"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller,Larger,XX-Small,X-Small,Small,Medium, Large,X-Large,XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dcdcdd"
  Height="size"
  ImageUrl="url"
  LoadingCssClass="CSS class name"
  LoadingImageUrl="string"
  OnError="string - JS function to call on error"
  OnSuccess="string - JS function to call on success"
  Style="string"
  Target="jQuery element selector"
  Text="string"
  ToolTip="string"
  Url="url"
  Visible="True|False"
  Width="size" />
```
Remarks

- **Usage**: The XMod Pro Ajax button controls work in conjunction with jQuery. They enable you to leverage jQuery without having to write any Javascript. Because of this, you must ensure that jQuery has been included in the page. If you are using DNN 5 or later, the library is usually included in the page without any effort on your part. If not, use the `<scriptblock>` tag to include the library. You must specify the Url property and Target property. Optionally, you can specify the LoadingImageUrl and LoadingCssClass.

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control.

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units.

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in units.

- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application's root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.

- **LoadingCssClass**: A CSS class name to assign to the image which appears after the button has been clicked - to indicate content is being loaded. This property is ignored if no LoadingImageUrl is specified.

- **LoadingImageUrl**: A URL to the image file that will be displayed after the button is clicked - to indicate content is being loaded. You can use the tilde (~) character as a placeholder for the website root directory. The image will be displayed immediately after the button when the button is clicked. It will be removed from the page after successful completion of the AJAX call. If no image url is specified, no image will be displayed.

- **OnError**: You can optionally specify a Javascript function to call if there is an error in the AJAX call. Your function should accept the following parameters: jqXHR, textStatus, errorThrown. This should only contain a function name. An example would be:

```javascript
OnError="myErrorHandler"
```

Elsewhere, you would define your Javascript function like:

```javascript
function myErrorHandler(jqXHR, textStatus, errorThrown) {
    alert("The following error occurred: "+ textStatus);
}
```

- **OnSuccess**: You can optionally specify a Javascript function to call when data is returned from the AJAX call. This overrides standard default processing of the AJAX call - which normally sets the HTML of the Target element. Instead, your function will be called and the returned data from the AJAX call will be passed to your function. This property should only contain a function name. An example might be:

```javascript
OnSuccess="doSomethingCool"
```
Elsewhere you would define your Javascript function like:

```javascript
function doSomethingCool(data) {
    alert("The following data was returned:" + data);
}
```

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Target**: A jQuery "selector" that identifies the element(s) that will contain the HTML returned from the URL. Note that content of the element will be replaced by the HTML. To select an element by its ID, use the # selector. So, to select an element with the client ID of "divMyResults" (without the quotes), the Target attribute would be 
  "#divMyResults" (again, without the quotes). To select all DIV elements with the class name of "MyResults", you would use the period selector (.): 
  "div.MyResults".

- **Text**: The caption that will be displayed on the control.

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Width of the control in units.

---

**Example**

```xml
<AddForm>
    <SubmitCommand CommandText="Update EmployeeReview SET Rating=@Rating" />

    <Label>Submit Rating for Employee</Label>
    <DropdownList Id="Rating" DataField="Rating" DataType="Int32">
        <ListItem Value="1">Poor</ListItem>
        <ListItem Value="2">Sub-Par</ListItem>
        <ListItem Value="3">Average</ListItem>
        <ListItem Value="4">Above Average</ListItem>
        <ListItem Value="5">Excellent</ListItem>
    </DropdownList>
    <AjaxImage AlternateText="View Employee History" Url="mysite.com/history.aspx?eid=100" 
        Target="#divHistory" ImageUrl="~/images/history.gif" />

    <div id="divHistory"></div>
    <AddButton Text="Add Rating" />
    <CancelButton Text="Nevermind" />
</AddForm>
```

---

**<AjaxLink>**

The AjaxLink tag renders as a hyperlink at run-time that, when clicked, will dynamically insert HTML returned from a URL into an element on the page - without a postback. This is a jQuery based control. It requires jQuery to be included in the page and that Javascript be enabled in the end-user's browser.
Syntax

<AjaxLink
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
 ForeColor="color name|#dddddd"
  Height="size"
  LoadingCssClass="CSS class name"
  LoadingImageUrl="string"
  OnError="string - JS function to call on error"
  OnSuccess="string - JS function to call on success"
  Style="string"
  Target="jQuery element selector"
  Text="string"
  ToolTip="string"
  Url="url"
  Visible="True|False"
  Width="size" />

Remarks

- **Usage**: The XMod Pro Ajax button controls work in conjunction with jQuery. They enable you to leverage jQuery without having to write any Javascript. Because of this, you must ensure that jQuery has been included in the page. If you are using DNN 5 or later, the library is usually included in the page without any effort on your part. If not, use the `<scriptblock>` tag to include the library. You must specify the Url property and Target property. Optionally, you can specify the LoadingImageUrl and LoadingCssClass.

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control.

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units.

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in units.

- **LoadingCssClass**: A CSS class name to assign to the image which appears after the button has been clicked - to indicate content is being loaded. This property is ignored if no LoadingImageUrl is specified.
- **LoadingImageUrl**: A URL to the image file that will be displayed after the button is clicked - to indicate content is being loaded. You can use the tilde (~) character as a placeholder for the website root directory. The image will be displayed immediately after the button when the button is clicked. It will be removed from the page after successful completion of the AJAX call. If no image url is specified, no image will be displayed.

- **OnError**: You can optionally specify a Javascript function to call if there is an error in the AJAX call. Your function should accept the following parameters: jqXHR, textStatus, errorThrown. This should only contain a function name. An example would be:

  ```javascript
  OnError="myErrHandler"
  ```

  Elsewhere, you would define your Javascript function like:

  ```javascript
  function myErrHandler(jqXHR, textStatus, errorThrown) {
    alert("The following error occurred: " + textStatus);
  }
  ```

- **OnSuccess**: You can optionally specify a Javascript function to call when data is returned from the AJAX call. This overrides standard default processing of the AJAX call - which normally sets the HTML of the Target element. Instead, your function will be called and the returned data from the AJAX call will be passed to your function. This property should only contain a function name. An example might be:

  ```javascript
  OnSuccess="doSomethingCool"
  ```

  Elsewhere you would define your Javascript function like:

  ```javascript
  function doSomethingCool(data){
    alert("The following data was returned: " + data);
  }
  ```

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Target**: A jQuery "selector" that identifies the element(s) that will contain the HTML returned from the Url. Note that content of the element will be replaced by the HTML. To select an element by its ID, use the # selector. So, to select an element with the client ID of "divMyResults" (without the quotes), the Target attribute would be 

  ```javascript
  "#divMyResults" (again, without the quotes).
  ```

  To select all DIV elements with the class name of "MyResults", you would use the period selector (.) - "div.MyResults".

- **Text**: The caption that will be displayed on the control.

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Width of the control in [units](#).
Example

<AddForm>
  <SubmitCommand CommandText="Update EmployeeReview SET Rating=@Rating" />

  <Label>Submit Rating for Employee</Label>
  <DropdownList Id="Rating" DataField="Rating" DataType="Int32">
    <ListItem Value="1">Poor</ListItem>
    <ListItem Value="2">Sub-Par</ListItem>
    <ListItem Value="3">Average</ListItem>
    <ListItem Value="4">Above Average</ListItem>
    <ListItem Value="5">Excellent</ListItem>
  </DropdownList>
  <AjaxLink Text="View Employee History" Url="mysite.com/history.aspx?eid=100" Target="#divHistory" />
  <div id="divHistory"></div>
  <AddButton Text="Add Rating" />
  <CancelButton Text="Nevermind" />
</AddForm>

Syntax Remarks Example

The CalendarButton tag renders as a push-button at run-time. When clicked, a calendar date-picker pops up to enable the user to select a date.

Syntax

<CalendarButton
  AccessKey="string"
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Format="date-formatting expression"
  Height="size"
  Style="string"
  TabIndex="integer"
  Target="ID of control that will receive the selected date"
  Text="string"
  ToolTip="string"
  Visible="True|False"
  Width="size"

Remarks

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their
keyboard (for Windows machines)

- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in **units**.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#).
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Format**: If specified, this overrides the default date format used by the pop-up calendar. If left blank, the web server's default short date format will be used. An example format would be: format="yyyy-MM-dd" where "yyyy" returns the four digit year, "MM" returns a two-digit month, and "dd" returns a two-digit day. If you need the value to stay in that format, consider also using the `<validate type="regex">` tag to validate the target control.
- **Height**: Height of the control, specified in **units**.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Target**: This is the ID of the control where the calendar's selected date will be sent. This should be a text box.
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in **units**.

**Example**

```xml
<addform>
    <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
    <table>
        <tr>
            <td>
                <label for="txtEventDate" text="Event Date" />
                <textbox id="txtEventDate" datafield="EvtDate" datatype="datetime" />
                <calendarbutton text="Select Date" target="txtEventDate" format="yyyy-MM-dd" />
            </td>
        </tr>
        ...
        <td colspan="2">
            <addbutton text="Add"/> <cancelbutton text="Cancel"/>
        </td>
    </table>
</addform>
```

**Syntax**  **Remarks**  **Example**

The CalendarImage tag renders as a push-button at run-time. When clicked, a calendar date-picker pops up to enable the user to select a date.
Syntax

```xml
<CalendarImage
    AccessKey="string"
    AlternateText="string"
    BackColor="#dddddd"
    BorderColor="#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Small|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="#dddddd"
    Format="date-formatting expression"
    Height="size"
    ImageUrl="url"
    Style="string"
    TabIndex="integer"
    Target="ID of control that will receive the selected date"
    ToolTip="string"
    Visible="True|False"
    Width="size"
```

Remarks

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)
- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Format**: If specified, this overrides the default date format used by the pop-up calendar. If left blank, the web server's default short date format will be used. An example format would be: format="yyyy-MM-dd" where "yyyy" returns the four digit year, "MM" returns a two-digit month, and "dd" returns a two-digit day. If you need the value to stay in that format, consider also using the `<validate type="regex">` tag to validate the target control.
- **Height**: Height of the control, specified in units.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Target**: This is the ID of the control where the calendar's selected date will be sent. This should be a text box.
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.

Example

```xml
<addform>
    <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
</addform>
```
The Calendar tag renders as a push-button at run-time. When clicked, a calendar date-picker pops up to enable the user to select a date.

**Syntax**

```xml
<Calendar
   BackColor="color name|#dddddd"
   BorderColor="color name|#dddddd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
   BorderWidth="size"
   CssClass="string"
   Font-Bold="True|False"
   Font-Italic="True|False"
   Font-Names="string"
   Font-Overline="True|False"
   Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
   Font-Strikeout="True|False"
   Font-Underline="True|False"
   ForeColor="color name|#dddddd"
   Format="date-formatted expression"
   Height="size"
   Style="string"
   Target="ID of control that will receive the selected date"
   Text="string"
   ToolTip="string"
   Visible="True|False"
   Width="size"/>
```

**Remarks**

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in **units**.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Target**: This is the ID of the control where the calendar's selected date will be sent. This should be a text box.
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in **units**.

### Example

```xml
<addform>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
        <label for="txtEventDate" text="Event Date" />
        <textbox id="txtEventDate" datafield="EvtDate" datatype="datetime" />
        <CalendarLink text="Select Date" target="txtEventDate" format="yyyy-MM-dd" />
      </td>
    </tr>
    ... 
    <td colspan="2"> 
      <addbutton text="Add"/> <cancelbutton text="Cancel"/>
    </td>
  </tr>
</table>
</addform>
```

### Syntax

The `CancelButton` tag renders as a button at run-time. When clicked, the form closes without performing any actions and returns to the previous view if possible.
Syntax

```xml
<cancelbutton
  AccessKey="string"
 BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  OnClientClick="string"
  Redirect="url"
  RedirectMethod="Get|Post"
  Style="string"
  TabIndex="integer"
  Text="string"
  ToolTip="string"
  Visible="True|False"
  Width="size"
/>```

Remarks

- The cancel button can be used in `<AddForm>` and `<EditForm>` tags. Its purpose is to cancel form processing and hide the form.

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control.

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units.

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in units.

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return
true then the control will perform its normal processing.

- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes. Field tokens may be used in the redirect attribute. However, function tokens such as [[Portal:ID]], [[Join()]], [[User:ID]], etc. cannot be used. When field tokens are used, they are URL Encoded.

- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"

  IMPORTANT: When using "Post", the ID that you supply for your form controls determine the name of the field that is posted to the target URL, not the DataField. New to version 4.0: You can use a period (.) for the Redirect property's value. The period acts as shortcut to redirect to the current page.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Width of the control in units.

**Example**

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label target="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label target="txtLastName" text="Last Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr colspan="2">
      <addbutton text="Add" />
      <cancelbutton text="Cancel" />
    </tr>
  </table>
</addform>
```

**<CancelImage>**

**Syntax Remarks Example**

The CancelImage tag renders as a clickable image at run-time. When clicked, the form closes without performing any actions and returns to the previous view if possible.

**Syntax**

```xml
<CancelImage
  AccessKey="string"
  AlternateText="string"
  BackColor="color name|#dddddd"
```
Remarks

- The cancel button can be used in `<AddForm>` and `<EditForm>` tags. Its purpose is to cancel form processing and hide the form.

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).

- **AlternateText**: Use this attribute’s value will be used as the image’s "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control.

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in **units**.

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in **units**.

- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application’s root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing.

- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes. Field tokens may be used in the redirect attribute. However, function tokens such as [[Portal:ID]], [[Join()]], [[User:ID]], etc. cannot be used. When field tokens are used, they are URL Encoded.

  - New to version 4.0: You can use a period (.) for the Redirect property's value. The period acts as shortcut to redirect to the current page.

- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"

  - IMPORTANT: When using "Post", the ID that you supply for your form controls determine the name of the field that is posted to the target URL, not the DataField.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Width of the control in units.

---

**Example**

```xml
<addform>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="Last Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr colspan="2">
      <AddImage AlternateText="Add" ImageUrl="~/images/add.gif" />
      <CancelImage AlternateText="Cancel" ImageUrl="~/images/cancel.gif" />
    </tr>
  </table>
</addform>
```

---

**Syntax**

- `<submitcommand commandtext=""`
- `<td>
- `<label for="" text="" />
- `<textbox id="" datafield="" datatype="" />
- `<AddImage AlternateText="" ImageUrl="" />
- `<CancelImage AlternateText="" ImageUrl="" />

**Remarks**

- `commandtext` attribute is required.
- `div` attribute is optional.
- `datafield` attribute is optional.
- `datatype` attribute is optional.
- `AlternateText` attribute is optional.
- `ImageUrl` attribute is optional.

**Example**

- `<submitcommand commandtext="" />
- `<td>
- `<label for="" text="" />
- `<textbox id="" datafield="" datatype="" />
- `<AddImage AlternateText="" ImageUrl="" />
- `<CancelImage AlternateText="" ImageUrl="" />

---

**Back to top**
The CancelLink tag renders as a hyperlink at run-time. When clicked, the form closes without performing any actions and returns to the previous view if possible.

**Syntax**

```html
<CancelLink
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  OnClientClick="javascript"
  Redirect="url"
  RedirectMethod="Get|Post"
  Style="string"
  Text="string"
  ToolTip="string"
  Visible="True|False"
  Width="size"/>
```

**Remarks**

- The CancelLink can be used in `<AddForm>` and `<EditForm>` tags. Its purpose is to cancel form processing and hide the form.

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in [units](#)

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in [units](#)

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return
true then the control will perform its normal processing.

- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes. Field tokens may be used in the redirect attribute. However, function tokens such as [[Portal:ID]], [[Join()]], [[User:ID]], etc. cannot be used. When field tokens are used, they are URL encoded. New to version 4.0: You can use a period (.) for the Redirect property’s value. The period acts as shortcut to redirect to the current page.

- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"
  
  IMPORTANT: When using "Post", the ID that you supply for your form controls determine the name of the field that is posted to the target URL, not the DataField.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Determines the width of the control in units.

**Example**

```
<AddForm>
    <SubmitCommand CommandText="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
    <table>
        <tr>
            <td>
                <Label For="txtFirstName" Text="First Name" />
                <TextBox Id="txtFirstName" Datafield="FirstName" DataType="string" />
            </td>
            <td>
                <Label For="txtLastName" Text="Last Name" />
                <TextBox Id="txtLastName" DataField="LastName" DataType="string" />
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <CancelButton Text="Add" /> <CancelButton Text="Cancel" />
            </td>
        </tr>
    </table>
</AddForm>
```

**Captcha**

The Captcha tag renders as a CAPTCHA control at run time.

**NOTE**: Due to a limitation in the underlying DNN CAPTCHA control, this tag can only be used in the FormView module.
Syntax

```xml
<Captcha
    BackColor="color name|#dddddd"
    BackgroundColor="color name|#dddddd"
    BackgroundImage="url"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CaptchaChars="string"
    CaptchaHeight="size"
    CaptchaLength="integer"
    CaptchaWidth="size"
    CssClass="string"
    ErrorMessage="string"
    ErrorStyle-BackColor="color name|#dddddd"
    ErrorStyle-BorderColor="color name|#dddddd"
    ErrorStyle-BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    ErrorStyle-BorderWidth="size"
    ErrorStyle-Font-Bold="True|False"
    ErrorStyle-Font-Italic="True|False"
    ErrorStyle-Font-Names="string"
    ErrorStyle-Font-Overline="True|False"
    ErrorStyle-Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"
    ErrorStyle-Font-Strikeout="True|False"
    ErrorStyle-Font-Underline="True|False"
    ErrorStyle-ForeColor="color name|#dddddd"
    Expiration="integer"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    ID="string"
    Text="string"
    Width="size"
/>
```

Remarks

If your forms are available to the public, chances are you'll get web 'bots filling in those forms with bogus information. To help protect against this, you can add a CAPTCHA control to your form. The control attempts to prove the user is an actual human by asking them to view a series of characters and typing those characters into a box for verification. The characters are rendered as an image and are skewed and obfuscated so that (hopefully) only a human could read them. While it is no guarantee your forms won't get spammed, the control typically reduces the frequency of those attacks.

- **BackColor**: Color of the background of the control.
- **BackgroundImage**: A URL to an image file to use as the background on which the characters will be placed. (optional)
- **BorderColor**: Color of the border around the control
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in **units**
- **CaptchaChars**: If you wish to specify your own characters that will be used to make up the code the user must type, you can specify them in this property.
- **CaptchaHeight**: The height of the area in which the characters will be displayed.
- **CaptchaLength**: The number of characters to use for the code.
- **CaptchaWidth**: The width of the area in which the characters will be displayed.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **ErrorMessage**: The message to display to the user if captcha validation failed.
- **ErrorStyle**: The style to use for displaying the error message. ErrorStyle is specified using the following syntax: ErrorStyle-styleAttributeName where 'styleAttributeName' is the name of the style attribute such as ForeColor or Font-Bold. See the syntax section above for more.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in [units](#).
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **Width**: Width of the control in [units](#).

### Example

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="First Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <captcha captchalength="5" />
      </td>
    </tr>
    <tr colspan="2">
      <addbutton text="Add"/> &nbsp; <cancelbutton text="Cancel"/>
    </tr>
  </table>
</addform>
```

### Remarks

The CheckBox tag renders as a checkbox and associated label at run-time. A series of CheckBox controls provides a useful alternative to the CheckBoxList. With the CheckBoxList, all selected values are placed into a single string value that is sent to the data source. However, a series of individual CheckBox controls provides you with the same basic functionality as the CheckBoxList, but you also have the ability to store the value of each CheckBox in its own data field.
Syntax

<CheckBox
    AccessKey="string"
    BackColor="color name"#dddddd"
    BorderColor="color name"#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    Checked="True|False"
    CssClass="string"
    DataField="string"
    DataType="Boolean"
    DataValueField="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Normal="string"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="#dddddd"
    Height="size"
    ID="string"
    Nullable="True|False"
    Style="string"
    TabIndex="integer"
    Text="string"
    TextAlign="Left|Right"
    ToolTip="string"
    Visible="True|False"
    Width="size"
/>

Remarks

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units.
- **Checked**: Determines whether the checkbox is checked.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.
- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. Valid values are: boolean only. This attribute is required if the control will participate in operations with your form's data commands.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **Nullable**: If this is set to True, the control will return a DBNull value if the control has not been checked. If a DBNull value is passed to this control, regardless of the Nullable setting, the control will be un-checked.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
The CheckboxList tag renders as a series of checkboxes at run-time.

**Syntax**

```html
<CheckBoxList
    AccessKey="string"
    AppendDataBoundItems="True|False"
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CellPadding="integer"
    CellSpacing="integer"
    CssClass="string"
    DataField="string"
    DataSourceID="string"
    DataTextField="string"
    DataTextFormatString="string"
    DataType="string|int32|...." 
    DataValueField="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
```
<ListItem value="string" selected="True|False">Item1</ListItem>
<ListItem value="string">Item2</ListItem>
...
</CheckBoxList>

Remarks

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).
- **AppendDataBoundItems**: If True, items retrieved from a `<controlDataSource>` tag will be appended to the list of items already defined in the control. This only applies if the control is bound to such a tag. The default value is False.
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units.
- **CellPadding**: For table layouts, sets the distance (in pixels) between the border and the content of the cells.
- **CellSpacing**: For table layouts, sets the distance (in pixels) between cells.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.
- **DataField**: Name of the parameter in the `<submitCommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectCommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **DataSourceId**: If this control's data is supplied by a `<controlDataSource>` tag, specify that tag's ID in this attribute. This attribute is required only if the control's data is supplied via a `<controlDataSource>` tag.
- **DataTextField**: When using a `<controlDataSource>` this attribute specifies the column name in that datasource that supplies each list item's display text. This attribute is required if the control's data is supplied via a `<controlDataSource>` tag.
- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. Valid values are: string (default), int32, int64, boolean, . This attribute is required if the control will participate in operations with your form's data commands.
- **DataValueField**: When using a `<controlDataSource>` this attribute specifies the column name in that datasource that supplies each list item's hidden value. This attribute is required only if the control's data is supplied via a `<controlDataSource>` tag.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **Nullabe**: If this is set to True, the control will return a DBNull value if no items have been checked. If a DBNull value is passed to this control, regardless of the Nullabe setting, all items in the control will be un-checked.
- **SelectedItemsSeparator**: If the control enables the selection of multiple items, the control will merge the selected values together using a pipe (|) as a separator. You can change the character used to separate the selected values using...
this property. If, for instance, you wanted to separate them with a comma, you would set $SelectedItemsSeparator$=",". The separator is only used on controls capable of multiple selection and ONLY when more than one item has been selected.

So, for example, if your control had the following values selected: 32, 578, and 38, then the value returned to the database would be: 32|578|38. If only the number 32 was selected, the value would be: 32.

NOTE that if you are using this control to supply email addresses to the $<email>$ tag, it assumes values are delimited with a pipe. However, since email addresses are comma-delimited, you could set $SelectedItemsSeparator$ to a comma and it should still function.

- **RepeatColumns**: Defines the number of columns to use when laying out the checkboxes.
- **RepeatDirection**: Determines if the control displays vertically or horizontally.
- **RepeatLayout**: Use this attribute to specify whether the items in the control are displayed in a table. If this attribute is set to `Table` the items in the list are displayed in a table. If this attribute is set to `Flow`, the items in the list are displayed without a table structure.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- ** TextAlign**: The alignment of the text label with respect to its associated check box. Valid values are Left and Right. Default value is Right.
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.
- **Usage**: The control allows $<listitem>$ child tags which define the items that will appear in the list. The control can also be bound to a $<controldatasource>$ tag. To do so, specify the ID of the $<controldatasource>$ tag in the $DataSourceId$ attribute, the name of the column in the data source that should supply the display text for each list item in the $DataTextField$ attribute, and the column in the data source that should supply the hidden value of each list item in the $DataValueField$ attribute.

### Example

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label target="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label target="lstColors" text="Favorite Color" />
        <checkboxlist id="cblColors" datafield="FavoriteColors" datatype="string">
          <listitem value="#00FF00">Green</listitem>
          <listitem value="#FF0000" selected="true">Red</listitem>
          <listitem value="#0000FF">Blue</listitem>
        </checkboxlist>
      </td>
    </tr>
    <tr col="2">
      <addbutton text="Add" /> <cancelbutton text="Cancel"/>
    </tr>
  </table>
</addform>
```

### Syntax Remarks Example
The `ContinueButton` tag renders as a push-button at run-time. It is only valid within an `<AddSuccessTemplate>` or `<EditSuccessTemplate>`. When clicked, the user is returned either to the page that would have been displayed after successfully submitting the form or to the URL specified in the `Redirect` attribute.

### Syntax

```
<modx:ContinueButton
    AccessKey="string"
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    OnClientClick="string"
    Redirect="url"
    RedirectMethod="Get|Post"
    Style="string"
    TabIndex="integer"
    Text="string"
    ToolTip="string"
    Visible="True|False"
    Width="size"
/>
```

### Remarks

- The `ContinueButton` should only be used in `<AddSuccessTemplate>` and `<EditSuccessTemplate>` tags. It’s purpose is to return the user to the page he/she would have seen if no success template was displayed or to send the user to the URL of your choice via the `Redirect` attribute.
- Unlike other form controls, the `ContinueButton` control is really a template tag and, thus, begin with the "xmod:" prefix like so: `<modx:ContinueButton>` rather than `<ContinueButton>`.
- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return `true` then the control will perform its normal processing.
• **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes.

• **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"

• **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

• **Visible**: Determines if the control is visible (true) or hidden (false)

• **Width**: Width of the control in units.

**Example**

```xml
<addform>...
</addform>

<AddSuccessTemplate>
  <h1>Thanks for Signing Up</h1>
  <p>Click the button below to go to your profile page</p>
  <xmod:ContinueButton Text="View Your Profile" Redirect="http://mysite.com/profile" RedirectMethod="Get" />
</AddSuccessTemplate>
```

**Syntax Remarks Example**

The ContinueImage tag renders as a clickable image at run-time. It is only valid within an `<AddSuccessTemplate>` or `<EditSuccessTemplate>` tag. When clicked, the user is returned either to the page that would have been displayed after successfully submitting the form or to the URL specified in the Redirect attribute.

**Syntax**

```xml
<xmod:ContinueImage
  AccessKey="string"
  AlternateText="string"
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Faces="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  ImageAlign="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbsBottom|AbsMiddle|TextTop"
  ImageUrl="url"
  OnClientClick="string"
  Redirect="url"
  RedirectMethod="Get|Post"
  Style="string"
  TabIndex="integer"
  Text="string"
  ToolTip="string"
  Visible="True|False"
```
Remarks

- The ContinueImage tag should only be used in `<AddSuccessTemplate>` and `<EditSuccessTemplate>` tags. It's purpose is to return the user to the page he/she would have seen if no success template was displayed or to send the user to the URL of your choice via the Redirect attribute.
- Unlike other form controls, the ContinueImage control is really a template tag and, thus, begins with the "xmod:" prefix like so: `<xmod:ContinueImage>` rather than `<ContinueImage>`.
- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).
- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines.
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in [units](#).
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#).
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in [units](#).
- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application's root directory. For instance: `ImageUrl="~/images/myimage.gif"` might map to "/dnntestsites/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.
- **OnClick**: If you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes.
- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post".
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;").
- **Visible**: Determines if the control is visible (true) or hidden (false).
- **Width**: Width of the control in [units](#).

Example

```
<addform>...
</addform>

<AddSuccessTemplate>
  <h1>Thanks for Signing Up!</h1>
  <p>Click the button below to go to your profile page</p>
  <xmod:ContinueImage AlternateText="View Your Profile" Redirect="http://mysite.com/profile" RedirectMethod="Get" ImageUrl="~/images/profile.gif"/>
</AddSuccessTemplate>
```

Back to top
Syntax Remarks Example

The ContinueLink tag renders as a hyperlink at run-time. It is only valid within an `<AddSuccessTemplate>` or `<EditSuccessTemplate>` tag. When clicked, the user is returned either to the page that would have been displayed after successfully submitting the form or to the URL specified in the Redirect attribute.

Syntax

```xml
<xmod:ContinueLink
   BackColor="color name"#dddddd"
   BorderColor="#dddddd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
   BorderWidth="size"
   CssClass="string"
   Font-Bold="True|False"
   Font-Italic="True|False"
   Font-Names="string"
   Font-Overline="True|False"
   Font-Size="string|Small|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
   Font-Strikeout="True|False"
   Font-Underline="True|False"
   ForeColor="#dddddd"
   Height="size"
   OnClientClick="javascript"
   Redirect="url"
   RedirectMethod="Get|Post"
   Style="string"
   Text="string"
   ToolTip="string"
   Visible="True|False"
   Width="size"/>
```

Remarks

- The ContinueLink tag should only be used in `<AddSuccessTemplate>` and `<EditSuccessTemplate>` tags. Its purpose is to return the user to the page he/she would have seen if no success template was displayed or to send the user to the URL of your choice via the Redirect attribute.
- Unlike other form controls, the ContinueLink control is really a template tag and, thus, begin with the "xmod:" prefix like so: `<xmod:ContinueLink>` rather than `<ContinueLink>`.
- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.
- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes.
- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in **units**.

**Example**

```xml
<addform>...
</addform>

<AddSuccessTemplate>
  <h1>Thanks for Signing Up</h1>
  <p>Click the button below to go to your profile page</p>
  <xmod:ContinueLink Text="View Your Profile" Redirect="http://mysite.com/profile" RedirectMethod="Get"/>
</AddSuccessTemplate>
```

**<ControlDataSource>**

**Syntax**

```xml
<ControlDataSource
  CommandText="string"
  ConnectionString="string"
  Id="string"
  Source="string">
  Optionally add Parameter tags for any parameters you need to send to the database.
  <Parameter Name="string" Value="string" DataType="data type"/>
  ...
</ControlDataSource>
```

**Remarks**

- **Id**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form. This attribute is required as it provides the link between the data source and the controls to which it is bound.

- **CommandText**: The SQL command to execute to return the data. This attribute is required. Optionally, when **Source** has been set to *dnn*, you can specify special values for CommandText to enable retrieval of data from the DotNetNuke database via its API rather than direct database queries (see the Source property description for more details).

- **ConnectionString**: The database connection string for the datasource from which the data will be retrieved. This attribute is optional. XMod Pro will use the connection string for the current DNN database by default. To use connection strings stored in the web.config file, use the `[[ConnectionString:connectionStringName]]` where connectionStringName is the
name assigned to the connection string in the web.config file

- **Source**: New to version 4.0. You can optionally specify "dnn" (without quotes) as this property's value. When Source is set to dnn, you can retrieve different sets of data using the DNN API rather than querying the database directly. You determine what data you want to retrieve by specifying it in the CommandText property. Currently supported values are:

  - **Users**: Returns a list of users from the current portal
  - **Roles**: Returns a list of security roles in the current portal
  - **Pages**: Returns a list of pages in the current portal
  - **ListCountries**: Returns the list of countries defined in the DotNetNuke Lists table.

- **<Parameter>**: Beginning with version 1.2, the ControlDataSource tag now recognizes `<parameter>` child tags:

  - **Name**: This is the name of the parameter and should be the same as the @param used in your CommandText.
  - **Value**: This is the value of the parameter. You can hard-code the value or you can use function tokens like `[[Url:paramName]]` or `[[Portal:ID]]`, etc.

For each control that will use the data, you need to specify:

- **DataSourceId**: This is the Id of the `<ControlDataSource>` and serves to link the control to the data source.
- **DataTextField**: This is the name of the field in the data source that contains the value to use for the control's display text.
- **DataValueField**: This is the name of the field in the data source that contains the value to use for the control's hidden text.

**Example**

```xml
<AddForm>
  <SelectCommand CommandText="SELECT @FirstName AS FirstName, @LastName AS LastName, 'AZ' AS StateId"
  <Parameter Name="FirstName" Value='[[User:FirstName]]' DefaultValue=""/>
  <Parameter Name="LastName" Value='[[User:LastName]]' DefaultValue="" />
  </selectcommand>
  <SubmitCommand CommandText="INSERT INTO Users(FirstName, LastName, StateId)
VALUES(@FirstName, @LastName, @StateId)" />
</selectcommand>
  <ControlDataSource Id="dsStates" ConnectionString="(your connection string here)"
  CommandText="SELECT StateName, StateId FROM States ORDER BY StateName ASC" />
</AddForm>
```

```xml
<table>
  <tr>
    <td>
      <Label For="txtFirstName" Text="First Name" />
      <TextBox Id="txtFirstName" DataField="FirstName" DataType="string" />
    </td>
  </tr>
  <tr>
    <td>
      <Label For="txtLastName" Text="Last Name" />
      <TextBox Id="txtLastName" DataField="LastName" DataType="string" />
    </td>
  </tr>
  <tr>
    <td>
      <Label For="ddlState" Text="State" />
      <DropdownList Id="ddlState" DataField="StateId" DataType="int32"
DatasourceId="dsStates" DataTextField="StateName" DataValueField="StateId"/>
    </td>
  </tr>
  <tr>
    <td colspan="2">
```
Back to top

<DateInput>

Syntax Remarks Example

The DateInput tag renders as a single-line textbox at run time that accepts dates and, optionally, time.

Syntax

<DateInput
   AccessKey="string"
   BackColor="color name | #dddddd"
   BorderColor="color name | #dddddd"
   BorderStyle="NotSet | None | Dotted | Dashed | Solid | Double | Groove | Ridge | Inset | Outset"
   BorderRadius="size"
   CssClass="string"
   Culture="locale id"
   DataField="String"
   DataType="datetime | date"
   DateOnly="True | False"
   Font-Bold="True | False"
   Font-Italic="True | False"
   Font-Names="string"
   Font-Strikeout="True | False"
   Font-Underline="True | False"
   ForeColor="color name | #dddddd"
   Format="date/time formatting expression"
   Height="size"
   ID="string"
   Nullable="True | False"
   ReadOnly="True | False"
   Style="string"
   TabIndex="integer"
   ToolTip="string"
   Visible="True | False"
   Width="size"
/>

Remarks

When you need to enter dates in your forms, the DateInput tag will help. It allows you to specify what culture XMod Pro should use when evaluating the input so it can better convert it to the appropriate datetime data type. Additionally, you can specify a format for the input that will be used when editing a record. The "datatype" attribute defaults is datetime.
- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control
- **BorderStyle**: Style of the border around the control
- **BorderWidth**: Width of the border around the control, specified in units.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **Culture**: A locale ID identifying the culture that should be used when converting the control’s input to a date for use by the database. Some examples: en-GB, es-MX, es-SP, en-US. If no culture is specified the system’s current culture is used.
- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control’s data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control’s data when the form is loaded. This attribute is required if the control will participate in operations with your form’s data commands.
- **DateOnly**: (new to version 3.1) When set to True, the control only processes the date portion of the value entered. While a time may be entered by the user, it will not be sent to the database. When a date if retrieved from the database for editing, only the date will be displayed. It’s important to remember that when the DataType is set to DateTime, there is always a time component - even if it isn’t visible in the control. So, when DataType is DateTime and DateOnly is true, the time component will always be set to 12:00:00 AM. When DataType is set to Date, only the date component is processed and stored in the database and he DateOnly property does not need to be set.
- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. . This attribute is required if the control will participate in operations with your form’s data commands. The DateInput uses datetime by default.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Determines how it will be displayed. You can use standard date formatting expressions here:
  - d: Represents the day of the month as a number from 1 through 31
  - dd: Represents the day of the month as a number from 01 through 31
  - M: Represents the month as a number from 1 to 12
  - MM: Represents the month as number from 01 to 12
  - yy: Represents the year as a two digit number. 2008 would be 08, 1999 would be 99
  - yyyy: Represents the year as a four digit number. 2008 would be 2008, 1999 would be 1999
  - h: Represents the hour as a number from 1 to 12, using a twelve hour clock
  - hh: Represents the hours as a number from 01 to 12 using a twelve hour clock
  - H: Represents the hour as a number from 1 to 24 using a 24 hour clock
  - HH: Represents the hour as a number from 01 to 24 using a 24 hour clock
  - m: Represents the minute as a number from 0 to 59
  - mm: Represents the minute as a number from 00 to 59
  - s: Represents the second as a number from 0 to 59
  - ss: Represents the second as number from 00 to 59
- **Height**: Height of the control, specified in units.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **Nullable**: If True (the default is False), the control will return a DBNull value if the control is blank or contains just whitespace. If a DBNull value is passed to the control, the control will be set to an empty string.
- **ReadOnly**: If True, the contents of the control cannot be changed. The default value is False.
- **MaxLength**: The maximum number of characters allowed in the text box.
- **Style**: Same as the HTML style attribute.It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **TabIndex**: Sets the tab index for the control
- **Tooltip**: In browsers that support it, sets the text to display when the mouse pointer hovers over the control
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.
Example

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtEventDate" text="Event Date" />
        <DateInput id="txtEventDate" datafield="EventDate" datatype="datetime"
                   culture="fr-FR" format="dd-MM-yyyy"/>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <addbutton text="Add"/> &nbsp; <cancelbutton text="Cancel"/>
      </td>
    </tr>
  </table>
</addform>
```

Back to top

**<DropDownList>**

- **Syntax**
- **Remarks**
- **Example**

The DropdownList tag renders as a drop-down list control at run-time.
Syntax

```xml
<DropDownList
   AccessKey="string"
   AppendDataBoundItems="True|False"
   BackColor="color name|#dddddd"
   BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
   BorderWidth="size"
   CssClass="string"
   DataField="string"
   DataSourceID="string"
   DataTextField="string"
   DataTextFormatString="string"
   DataType="string|int32|...."
   DataValueField="string"
   Font-Bold="True|False"
   Font-Italic="True|False"
   Font-Names="string"
   Font-Overline="True|False"
   Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
   Font-Strikeout="True|False"
   Font-Underline="True|False"
   ForeColor="color name|#dddddd"
   Height="size"
   ID="string"
   Nullable="True|False"
   ParameterName="string"
   Style="string"
   TabIndex="integer"
   TargetControlId="string"
   TargetDataSourceId="string"
   ToolTip="string"
   Visible="True|False"
   Width="size">
   <listitem value="string" selected="True|False">Item1</listitem>
   <listitem value="string">Item2</listitem>
   ...
</DropDownList>
```

Remarks

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).

- **AppendDataBoundItems**: If True, items retrieved from a `<controldatasource>` tag will be appended to the list of items already defined in the control. This only applies if the control is bound to such a tag. The default value is False.

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units
• **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control

• **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.

• **DataSourceId**: If this control's data is supplied by a `<controldatasource>` tag, specify that tag's ID in this attribute. This attribute is required only if the control's data is supplied via a `<controldatasource>` tag.

• **DataTextField**: When using a `<controldatasource>` this attribute specifies the column name in that datasource that supplies each list item's display text. This attribute is required if the control's data is supplied via a `<controldatasource>` tag.

• **DataType**: The type of data this control is supplying to the data commands. This is a [Database type](#). Valid values are: string (default), int32, int64, boolean, . This attribute is required if the control will participate in operations with your form's data commands.

• **DataValueField**: When using a `<controldatasource>` this attribute specifies the column name in that datasource that supplies each list item's hidden value. This attribute is required only if the control's data is supplied via a `<controldatasource>` tag.

• **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)

• **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

• **Height**: Height of the control, specified in [units](#).

• **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.

• **Nullable**: Since an item is always selected in a drop-down list, if Nullable is set to True (the default value is False), the control will return a DBNull value if the selected item has a hidden value of an empty string. If a DBNull value is passed to this control, regardless of the Nullable setting, XMod Pro will attempt to select the first item that has an empty string as its hidden value. If no item is found, it will attempt to select the first item in the list.

• **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

• **TargetControlID**: One or more control ID's (separated by commas - new to v.4.0). This property is used when creating dependent list controls and reflects the control(s) which should be updated when this control's value has changed.

• **TargetDataSourceId**: One or more ControlDataSource control ID's (separated by commas - new to v.4.0). This property is used when creating dependent list controls and reflects the data sources that should be updated with this control's newly selected value.

• **Visible**: Determines if the control is visible (true) or hidden (false)

• **Width**: Width of the control in [units](#).

• **Usage** The drop-down list allows `<listitem>` child tags which define the items that will appear in the list. The control can also be bound to a `<controldatasource>` tag. To do so, specify the ID of the `<controldatasource>` tag in the control's "datasourceid" attribute, the name of the column in the data source that should supply the display text for each list item, and the column in the data source that should supply the hidden value of each list item.
Creating Dependent Lists/Cascading Lists

New in Version 1.5: Two drop-down list controls can be bound together so that the selected value in the first control determines the available values in the second control. To do this, you'll need to set a few properties on the 1st control and bind the 2nd control to a `<ControlDataSource>`.

Example

For this example, we'll be using the DNN Lists table. This table contains numerous values - among them are a list of Countries and, for some of those countries, a list of regions. We will set up two drop-down list controls. The 1st control will load a list of countries from the Lists table. The 2nd control will be dynamically loaded with a list of regions (if any) for the selected country. Not all countries will have regions.

```
<AddForm>
    <SubmitCommand CommandText="INSERT INTO Contacts(ContactName,Country,Region) VALUES (@ContactName,@Country,@Region)" />
    <ControlDataSource id="dsCountries" CommandText="SELECT Text, EntryID FROM Lists WHERE ListName='Country' ORDER BY Text"/>
    <ControlDataSource id="dsRegions" CommandText="SELECT Text, Value FROM Lists WHERE ParentID=@ParentID" />
    <ControlDataSource id="dsRegions" ParameterName="ParentID" TargetControlId="ddlRegions" />
    ...
    <AddButton text="Add" />
    <CancelButton text="Cancel" />
</AddForm>
```

In the example above, there are two `<ControlDataSource>` tags. The first loads the list of countries from the DNN Lists table and is linked to the "ddlCountries" drop-down list. The second one will be used to lookup the list of regions in the selected country.

The DNN Lists table is setup so that each record can be the "parent" of one or more other records. If a record has a parent, its "ParentID" column will contain the EntryID of its parent. So, for our example, to get a list of regions in a given country, we look for all records that have a ParentID that matches the EntryID of our country. So, we've setup the ControlDataSource to accept a "ParentID" parameter.

Next, we need to setup the Country drop-down list to send its value to the Regions ControlDataSource. It is set to load its data from the "dsCountries" ControlDataSource using the DataSourceId, DataTextField, and DataValueField attributes. To enable the control to cause the Regions drop-down list to reload its data, we:

1. Specify the TargetControlId. This is the control we want to be dependent on this control's value. For the example, that is "ddlRegions".
2. Specify the TargetDataSourceId. This is the ID of the ControlDataSource used to fill the target control. In this example, it is "dsRegions".
3. Specify the ParameterName. This is the name of the parameter the target ControlDataSource is expecting in order to retrieve its data. In the example, this is "ParentID".

So, when a country is selected, a parameter will be created with the name "ParentID". It's value will be set to the value (not the display text) of the currently selected country. That parameter will then be passed to the target ControlDataSource (dsRegions) and the target control (ddlRegions) will be re-bound to the data.

Back to top
**Example 1 - General Usage**

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="ddlColors" text="Favorite Color" />
        <dropdownlist id="ddlColors" datafield="FavoriteColor" datatype="string">"
          <listitem value="#00FF00">Green</listitem>
          <listitem value="#FF0000" selected="true">Red</listitem>
          <listitem value="#0000FF">Blue</listitem>
        </dropdownlist>
      </td>
    </tr>
    <tr colspan="2">
      <addbutton text="Add" />
      <cancelbutton text="Cancel" />
    </tr>
  </table>
</addform>
```

**Example 2 - Binding to a Data Source**

```xml
<AddForm>
  ...
  <ControlDataSource Id="dsColors" CommandText="SELECT ColorId, ColorName FROM MyColorsTable"/>
  ...
  <DropDownList Id="ddlColors" DataSourceId="dsColors" DataTextField="ColorName" DataValueField="ColorId"
    DataField="FavoriteColor" DataType="Int32" AppendDataBoundItems="True" />
</AddForm>
```

**Example 3 - Adding Items to a Data-Bound List**

This example shows how to use the AppendDataBoundItems property to add a "None Selected" item to a list that is being populated from a table.

```xml
<AddForm>
  ...
  <ControlDataSource Id="dsColors" CommandText="SELECT ColorId, ColorName FROM MyColorsTable"/>
  ...
  <DropDownList Id="ddlColors" DataSourceId="dsColors" DataTextField="ColorName" DataValueField="ColorId"
    DataField="FavoriteColor" DataType="Int32"/>
  <ListItem Value="-1">(None Selected)</ListItem>
</DropDownList>
</AddForm>
```

**Example 4 - How to Require an Item Be Selected**

This example shows how you can require that the user choose an item from your data-bound list. Notice the Value for the "None Selected" item is set to an empty string. This will be interpreted by the Required Field Validator as not having a value. Presumably the colors you're retrieving from the MyColorsTable table will have values. This type of approach will work for hard-coded lists as well.
<AddForm>

...<ControlDataSource Id="dsColors" CommandText="SELECT ColorId, ColorName FROM MyColorsTable"/>

...<DropDownList Id="ddlColors" DataSourceId="dsColors" DataTextField="ColorName" DataValueField="ColorId"
              DataField="FavoriteColor" DataType="Int32" AppendDataBoundItems="True">
    <ListItem Value="">(None Selected)</ListItem>
</DropDownList>

<Validate Type="Required" Target="ddlColors" Text="***" Message="Please select a color" />

...<ValidationSummary Id="vsSummary" DisplayMode="BulletList" />

...</AddForm>

Back to top

<DualList>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Remarks</th>
<th>Example</th>
</tr>
</thead>
</table>

New to version 1.5. The DualList tag renders as two listboxes at run-time. Each box is separated by a series of buttons that enable moving items from the first listbox to the second listbox and vice versa. NOTE: The DualList requires Javascript to function correctly.
<DualList
AppendDataBoundItems="True|False"
BackColor="color name|#dddddd"
BorderColor="color name|#dddddd"
BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
BorderWidth="size"
ButtonStyle-BackColor="color name|#dddddd"
ButtonStyle-BorderColor="color name|#dddddd"
ButtonStyle-BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
ButtonStyle-BorderWidth="size"
ButtonStyle-Font-Bold="True|False"
ButtonStyle-Font-Italic="True|False"
ButtonStyle-Font-Names="string"
ButtonStyle-Font-Overline="True|False"
ButtonStyle-Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
ButtonStyle-Font-Strikeout="True|False"
ButtonStyle-Font-Underline="True|False"
ButtonStyle-ForeColor="color name|#dddddd"
CssClass="string"
DataField="string"
DataSourceID="string"
DataTextField="string"
DataTextFormatString="string"
DataType="string|int32|...."
DataValueField="string"
Font-Bold="True|False"
Font-Italic="True|False"
Font-Names="string"
Font-Overline="True|False"
Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
Font-Strikeout="True|False"
Font-Underline="True|False"
ForeColor="color name|#dddddd"
Height="size"
ID="string"
ListStyle-BackColor="color name|#dddddd"
ListStyle-BorderColor="color name|#dddddd"
ListStyle-BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
ListStyle-BorderWidth="size"
ListStyle-CssClass="string"
ListStyle-Font-Bold="True|False"
ListStyle-Font-Italic="True|False"
ListStyle-Font-Names="string"
ListStyle-Font-Overline="True|False"
ListStyle-Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
ListStyle-Font-Strikeout="True|False"
ListStyle-Font-Underline="True|False"
ListStyle-ForeColor="color name|#dddddd"
ListStyle-Width="size"
SelectedItemsSeparator="string"|
Style="string"
TabIndex="integer"
ToolTip="string"
Visible="True|False"
Width="size"/>
Remarks

- **AppendDataBoundItems**: If True, items retrieved from a `<controldatasource>` tag will be appended to the list of items already defined in the control. This only applies if the control is bound to such a tag. The default value is False. (new in version 3.0)
- **BackColor**: Color of the background of the control.
- **ButtonStyle**: The style to use for displaying the 'movement' buttons. ButtonStyle is specified using the following syntax: ButtonStyle-styleAttributeName where 'styleAttributeName' is the name of the style attribute such as ForeColor or Font-Bold or CssClass. See the syntax section above for more.
- **BorderColor**: Color of the border around the control
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **DataSourceId**: If this control's data is supplied by a `<controldatasource>` tag, specify that tag's ID in this attribute. This attribute is required only if the control's data is supplied via a `<controldatasource>` tag.
- **DataTextField**: When using a `<controldatasource>` this attribute specifies the column name in that datasource that supplies each list item's display text. This attribute is required if the control's data is supplied via a `<controldatasource>` tag.
- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. Valid values are: string (default), int32, int64, boolean, . This attribute is required if the control will participate in operations with your form's data commands. NOTE: If this is a multi-select listbox, you MUST set the DataType to "string" because the value returned from the control will be a string.
- **DataValueField**: When using a `<controldatasource>` this attribute specifies the column name in that datasource that supplies each list item's hidden value. This attribute is required only if the control's data is supplied via a `<controldatasource>` tag.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **ListStyle**: The style to use for displaying the both of the list controls. ListStyle is specified using the following syntax: ListStyle-styleAttributeName where 'styleAttributeName' is the name of the style attribute such as ForeColor or Font-Bold or CssClass. See the syntax section above for more.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **Nullable**: If Nullable is set to True (the default value is False), the control will return a DBNull value if no item has been selected. If a DBNull value is passed to this control, regardless of the Nullable setting, the control will de-select all items in the control.
- **SelectedItemsSeparator**: If the control enables the selection of multiple items, the control will merge the selected values together using a pipe (|) as a separator. You can change the character used to separate the selected values using this property. If, for instance, you wanted to separate them with a comma, you would set SelectedItemsSeparator="". The separator is only used on controls capable of multiple selection and ONLY when more than one item has been selected.

So, for example, if your control had the following values selected: 32, 578, and 38, then the value returned to the database would be: 32|578|38. If only the number 32 was selected, the value would be: 32. When dealing with multiple selections, remember to set the DataType to "string" because while "32|578|38" is a series of numbers, for the database, it is first and foremost a string containing a numeric text and the pipe character.

NOTE that if you are using this control to supply email addresses to the `<email>` tag, it assumes values are delimited with a pipe. However, since email addresses are comma-delimited, you could set SelectedItemsSeparator to a comma and it should still function.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in **units**.
- **Usage** The Listbox can operate in one of two modes: single selection, where only one item is allowed to be selected at a time, and multiple selection, which allows more than one item to be selected. The Listbox allows `<listitem>` child tags which define the items that will appear in the list. The control can also be bound to a `<controldatasource>` tag. To do so, specify the ID of the `<controldatasource>` tag in the Listbox's "datasourceid" attribute, the name of the column in the data source that should supply the display text for each list item, and the column in the data source that should supply the hidden value of each list item.

**Example**

```xml
<addform>
...
<table>
<tr>
<td>
<label for="txtFirstName" text="First Name" />
<textbox id="txtFirstName" datafield="FirstName" datatype="string" />
</td>
</tr>
</table>
</addform>
```

**Email**

**Syntax**

```
<Email
  To="comma-delimited list of email addresses"
  From="email address"
  CC="string"
  BCC="string"
  ReplyTo="string"
  Subject="string"
  Format="Text|Html"
  Attachment="mapped path and filename of attachment"
  SendIf="expression">
...
</Email>
```
Remarks

- You may include more than one Email tag in each form.
- The Email tag can be used to send multiple emails but it is not intended as a bulk email mechanism.
- Field Tokens may be used in the email attributes and in the body of the email.
- **Attachment**: Attaches a file in the file system of the website to the email. The value must be a file system-based path (mapped path) and filename.
- **BCC**: Blind Carbon Copy - email addresses will receive a copy of the email but will not show up in the list of recipients. You may specify a single email address or a list of addresses. Addresses can be separated by commas. NOTE, you may use email addresses derived from list controls as well. The list control must use the pipe (|) separator to separate its values for this to work correctly. (added in version 2.1)
- **CC**: Carbon Copy - email addresses will receive a copy of the email and will show up in the list of recipients as having been CC’ed. You may specify a single email address or a list of addresses. Addresses can be separated by commas. NOTE, you may use email addresses derived from list controls as well. The list control must use the pipe (|) separator to separate its values for this to work correctly. (added in version 2.1)
- **Format**: "Text" to send a plain text email or "HTML" to send an HTML-based email.
- **From**: The email address that should be displayed in the "From" field of the email.
- **ReplyTo**: (New to version 4.0) The email address that will be listed as the Reply-To address when the email recipient clicks "Reply" in their email client. Use this if you want replies sent to an email address that is different than the From email address.
- **SendIf**: (new to version 2.6) An expression that, when it evaluates to True, indicates the email should be sent. Typically you'll use this attribute if you want to only send an email if a user selects a certain value in your form.

  Example 1: SendIf='[[Department]] = Sales'. In this example we are taking the value of the "Department" column and comparing it to "Sales" If they are equal, the email will be sent. If they are not, the email will not be sent.

  Example 2: SendIf='[[Department]] <> Sales'. In this example, if "Department" doesn't equal "Sales", the email will be sent. Otherwise, it won't be sent.

  NOTE: Comparisons are not text-only and are not case-sensitive. You can test for equality using the "=" operator or inequality using the "<>" operator.
- **To**: A single email address or comma-delimited list of email addresses that should receive the email. NOTE, you may use email addresses derived from list controls as well. The list control must use the pipe (|) separator to separate its values for this to work correctly.

Back to top

Example

```xml
<addform>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
  </table>
</addform>
```
The FileUpload tag allows your users to upload a file to your web server. If a file has been uploaded, its filename is displayed.

Syntax

```xml
<FileUpload
    ID="string"
    DataField="string"
    DataType="string"
    DisplayMode="FilePicker|FilePickerNoUpload|UploadAndSelect"
    Extensions="comma-delimited list of extensions"
    FileNameLabelCssClass="string|Normal"
    MessageLabelCssClass="string|Normal"
    NewFileButtonCssClass="string|CommandButton"
    NewFileButtonText="string|Upload File"
    Nullable="True|False"
    Path="string"
    UploadButtonCssClass="string|CommandButton"
    UploadButtonText="string|Upload"
    UseUniqueFileName="True|False"
    Visible="True|False"
/>
```

Remarks

If your users need to upload files like images or documents to your web server, you can use the `<fileupload>` tag. If a file has been previously uploaded, the control displays the file's name. The user can click the "New File" button to show the HTML file upload control that allows them to browse to their file on their local machine. Once they've selected the file, they click the "Upload" button to begin the upload process. If the file's extension does not match those that have been specified in the "extensions" attribute or if there is some other error, it is displayed for the user. The "datatype" attribute is always `string`.

Validating the FileUpload Control: The nature of the FileUpload control does not allow it to be validated on the client. If you use a validator with this control, set the "enableclientscript" attribute to "false".

The FileUpload control has the following attributes:
**DisplayMode:** (New to v.2.7) This allows you to determine how the control behaves. You can configure it to be a file picker, where the files in the specified directory are displayed in a drop-down list; a file picker with the ability to upload a new file; or, the default, provide the user with the ability to upload a file and have that file be the "selected" file for the control. The UploadAndSelect method is the same method that earlier versions of this control used.

**Extensions:** A comma-delimited list of file extensions that define the file types that are allowed to be uploaded. File extensions should not include the period. For instance: gif,jpg,tiff is a valid value for Extensions. This attribute is required.

**FileNameLabelCssClass:** The Cascading Style Sheets (CSS) class name to associate with the label that displays the selected/uploaded file name. The default value is "Normal"

**MessageLabelCssClass:** The Cascading Style Sheets (CSS) class name to associate with the label that displays any error messages that result from operation of the control. The default value is "Normal"

**NewFileButtonCssClass:** The Cascading Style Sheets (CSS) class name to associate with the link button the user clicks to show the file upload control. The default value is "CommandButton"

**NewFileButtonText:** The text to use for the link button the user clicks to show the file upload control. The default value is "Upload File"

**Nullable:** If True (the default is False), the control will return a DBNull value when no file has been uploaded. If a DBNull value is passed to the control, regardless of the Nullable setting, no value will be set for the uploaded file.

**Path:** The path to the file on the web server where the file should be saved. This path must be accessible to the web application and permissions must be set to allow file operations. The path should be written relative to the web application root and may use the tilde (~) character to represent the root. Examples: ~/Portals/0/images/" or "/~/Portals/0/images/"

**UploadButtonCssClass:** The Cascading Style Sheets (CSS) class name to associate with the button the user clicks to initiate the file upload process. The default value is "CommandButton"

**UploadButtonText:** The text to use for the link button the user clicks to initiate the file upload process. The default value is "Upload"

**UseUniqueFileName:** False by default, when this attribute is set to True, the control will generate a unique GUID-based name for the uploaded file. If the attribute is False, the original file name will be used.

**NOTE:** Because of Javascript's security mechanisms, the FileUpload control will NOT function if Partial Page Rendering has been enabled for the XMod Pro control. This is true of the ASP.NET FileUpload control and is not specific to XMod Pro's FileUpload control.

**Example**

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="Last Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="uplMugShot" text="Mug Shot" />
        <fileupload id="uplMugShot" path="/images/" extensions="gif,jpg" datafield="UserImage" datatype="string" />
      </td>
    </tr>
  </table>
  <addbutton text="Add="/cancelbutton text="Cancel"/>
</addform>
```
The HtmlInput tag renders as a rich text editor at run time. It uses the current HTML Editor provider for your DotNetNuke site.

### Syntax

```html
<HtmlInput
   DataField="string"
   DataType="string"
   Height="size"
   ID="string"
   Nullable="True|False"
   Visible="True|False"
   Width="size"
/>
```

### Remarks

The HtmlInput tag enables you to provide your users with an easy-to-use rich text editor. This allows them to enter and format text as they would in a word processor. Behind the scenes, the HtmlInput tag uses the default rich text editor setup for use with your site. The "datatype" attribute is always string.

Validating the HtmlInput Control: The nature of the HtmlInput control does not allow it to be validated on the client. If you use a validator with this control, set the "enableclientscript" attribute to "false".

The HtmlInput control has the following attributes:

- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. The datatype for the HtmlInput is always string.
- **Height**: Height of the control, specified in units.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **Nullable**: If True (the default is False), the control will return a DBNull value if the control is blank or contains just white-space. If a DBNull value is passed to the control, the control will be set to an empty string. NOTE: Different HTML editors may function differently. XMod Pro will set/read the "StringValue" property on the underlying editor provider. What the individual editor does with that is up to the editor.
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.

### Example

```html
<addform>
  ...
  <table>
    ...
    <tr>
      <td>
        <label for="txtBio" text="Bio" />
        <htmlinput id="txtBio" datafield="Bio" datatype="string" width="600" height="400"/>
      </td>
    </tr>
  ...
</addform>
```
<tr>  <td colspan="2">    <addbutton text="Add"/> &nbsp;<cancelbutton text="Cancel"/>  </td> </tr>  </table>  </addform>

**jQueryReady**

- **Syntax**

  `<jQueryReady>`  
  `[jQuery and/or Javascript script]`  
  `</jQueryReady>`

- **Remarks**

  - **Usage:** Use this tag to quickly insert Javascript and/or jQuery code that should only be run after the document’s DOM has been loaded. The jQuery(document).ready() function is standard fare when working with jQuery. This tag allows you to forget about the boilerplate script and focus on your script. The tag will place your script near the bottom of the page - standard practice for improving page load times. Importantly, the tag automatically creates a "closure" for your script allowing you to use the "$" shortcut instead of "jQuery" in your code. Use of the closure also helps ensure your script operates in its own scope and will be not be impacted by other Javascript on the page. This tag requires jQuery be included in the page.

- **Example**

  In the example below, added a "help" image and some help text just after the ListBox. We've set the help SPAN tag containing the help text to initially be hidden (style="display:none;"). Then, we used the jQueryReady tag to attach some code to the image tag's click event. It simply makes the help SPAN tag visible.

  `<AddForm>`  
  ...  
  `<table>`  
  `<tr>`  
  `<td>`  
  `<label for="txtFirstName" text="First Name" />`  
  `<textbox id="txtFirstName" datafield="FirstName" datatype="string" />`  
  `</td>`  
  `</tr>`  
  `<tr>`  
  `<td>`  
  `<label for="lstColors" text="Favorite Color" />`  
  `<listbox id="lstColors" datafield="FavoriteColors" datatype="string" selectionmode="single">`  
  `<listitem value="#00FF00" >Green</listitem>`  
  `<listitem value="#FF0000" selected="true">Red</listitem>`  
  `<listitem value="#0000FF" >Blue</listitem>`  
  `</listbox>`  
  `<img src="/images/help_icon.gif" class="help-icon" />`  
  `<span class="help-text" style="display:none;"">Choose your absolute favorite color</span>`  
  `</td>`  
  `</tr>`  
  `<tr>`  
  `<td colspan="2">`  
  `<addbutton text="Add"/> <cancelbutton text="Cancel"/>`  
  `</td>`
The Label tag renders as a static text at run-time.

**Syntax**

```xml
<Label
    ID="string"
    For="string"
    Text="string"
    CssClass="string"
    Visible="True|False"
/>
```

**Remarks**

The Label tag provides the ability to offer better accessibility to your users by using the "target" attribute to identify a form control for which the label provides the caption. This makes it much easier for screen reader applications to understand your form. When supplying text for the Label, you can either place it in the `text` attribute or you can place it between the opening `<label>` and closing `</label>` tags.

**Example**

```xml
<addform>
    ...
    <table>
        <tr>
            <td>
                <label id="lblFirstName" for="txtFirstName" text="First Name"/>
                <textbox id="txtFirstName" datafield="FirstName" datatype="string"/>
            </td>
        </tr>
        <tr>
            <td>
                <label id="lblLastName" for="txtLastName" text="Last Name"/>
                <textbox id="txtLastName" datafield="LastName" datatype="string"/>
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <addbutton text="Add"/> <cancelbutton text="Cancel"/>
            </td>
        </tr>
    </table>
</addform>
```
The Listbox tag renders as a single or multi-select listbox at run-time.

**Syntax**

```xml
<ListBox
    AccessKey="string"
    AppendDataBoundItems="True|False"
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    DataField="string"
    DataSourceID="string"
    DataTextField="string"
    DataTextFormatString="string"
    DataType="string|int32|...."
    DataValueField="string"
    FontStyle="True|False"
    FontStyle="True|False"
    FontNames="string"
    FontOverline="True|False"
    FontSize="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    FontStrikeout="True|False"
    FontUnderline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    ID="string"
    Rows="integer"
    SelectionItemsSeparator="string|""
    SelectionMode="Single|Multiple"
    Style="string"
    TabIndex="integer"
    ToolTip="string"
    Visible="True|False"
    Width="size">
    <ListItem value="string" selected="True|False">Item1</ListItem>
    <ListItem value="string">Item2</ListItem>
    ...
</ListBox>
```

**Remarks**

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).
- **AppendDataBoundItems**: If True, items retrieved from a `<controldatasource>` tag will be appended to the list of items already defined in the control. This only applies if the control is bound to such a tag. The default value is False.
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **DataSourceId**: If this control's data is supplied by a `<controldatasource>` tag, specify that tag's ID in this attribute. This attribute is required only if the control's data is supplied via a `<controldatasource>` tag.
- **DataTextField**: When using a `<controldatasource>` this attribute specifies the column name in that datasource that supplies each list item's display text. This attribute is required if the control's data is supplied via a `<controldatasource>` tag.
- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. Valid values are: string (default), int32, int64, boolean, . This attribute is required if the control will participate in operations with your form's data commands. NOTE: If this is a multi-select listbox, you MUST set the DataType to "string" because the value returned from the control will be a string.
- **DataValueField**: When using a `<controldatasource>` this attribute specifies the column name in that datasource that supplies each list item's hidden value. This attribute is required only if the control's data is supplied via a `<controldatasource>` tag.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **Nullable**: If Nullable is set to True (the default value is False), the control will return a DBNull value if no item has been selected. If a DBNull value is passed to this control, regardless of the Nullable setting, the control will de-select all items in the control.
- **Rows**: A numeric value greater than or equal to 1 that determines the number of rows to display in a list box.
- **SelectionMode**: Determines whether the listbox allows a maximum of one selection or more than one selection. Valid values are: single and multiple. This attribute is optional. The default value is single.
- **SelectedItemsSeparator**: If the control enables the selection of multiple items, the control will merge the selected values together using a pipe (|) as a separator. You can change the character used to separate the selected values using this property. If, for instance, you wanted to separate them with a comma, you would set SelectedItemsSeparator="," The separator is only used on controls capable of multiple selection and ONLY when more than one item has been selected.

So, for example, if your control had the following values selected: 32, 578, and 38, then the value returned to the database would be: 32|578|38. If only the number 32 was selected, the value would be: 32. When dealing with multiple selections, remember to set the DataType to "string" because while "32|578|38" is a series of numbers, for the database, it is first and foremost a string containing a numeric text and the pipe character.

NOTE that if you are using this control to supply email addresses to the `<email>` tag, it assumes values are delimited with a pipe. However, since email addresses are comma-delimited, you could set SelectedItemsSeparator to a comma and it should still function.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.
- **Usage** The Listbox can operate in one of two modes: single selection, where only one item is allowed to be selected at a time, and multiple selection, which allows more than one item to be selected. The Listbox allows `<listitem>` child tags which define the items that will appear in the list. The control can also be bound to a `<controldatasource>` tag. To do so, specify the ID of the `<controldatasource>` tag in the Listbox's "datasourceid" attribute, the name of the column in the data source that should supply the display text for each list item, and the column in the data source that should supply the hidden value of each list item.

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
      </td>
      <td>
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
  </table>
</addform>
```
Example 2 - Binding to a Data Source

```xml
<AddForm>
...
<ControlDataSource Id="dsColors" CommandText="SELECT ColorId, ColorName FROM MyColorsTable"/>
...
<ListBox Id="lstColors" DataSourceId="dsColors" DataTextField="ColorName" DataValueField="ColorId"
    DataField="FavoriteColor" DataType="Int32" AppendDataBoundItems="True">
    <ListItem Value="-1">(None Selected)</ListItem>
</ListBox>
</AddForm>
```

Example 3 - Adding Items to a Data-Bound List

This example shows how to use the AppendDataBoundItems property to add a "None Selected" item to a list that is being populated from a table.

```xml
<AddForm>
...
<ControlDataSource Id="dsColors" CommandText="SELECT ColorId, ColorName FROM MyColorsTable"/>
...
<ListBox Id="lstColors" DataSourceId="dsColors" DataTextField="ColorName" DataValueField="ColorId"
    DataField="FavoriteColor" DataType="Int32" AppendDataBoundItems="True">
    <ListItem Value="-1">(None Selected)</ListItem>
</ListBox>
</AddForm>
```

New to Version 4.0! The Login tag will log the specified user into the DNN website.
## Syntax

```xml
<Login
  FirstNameField="string"
  LastNameField="string"
  Password="string"
  RememberMe="True|False"
  UserIdField="string"
  Username="string"
</Login>
```

## Remarks

- The Login action is only executed if the form has been successfully submitted. If there is a validation error or an error is thrown from the database, the action will not be performed.

- **Order Is Important**: Action tags are executed sequentially, so the order they appear within the form can be important. As an example, if one action fails with an error, all actions prior to the failed action will have executed. Those that occur after the failed action will not be executed. Additionally, some actions may have the ability to modify form values (this modification occurs after any form data has been sent to the database) - i.e. process form values, do calculations on them, transform them, even add and remove values from the list. Those changes will affect any Action tags that are executed downstream that use Field tokens. In this manner, actions can be chained together with previous actions providing new or updated data for actions downstream.

- **Using Tokens**: Unlike most form tags, which evaluate their tokens when the form is loaded, Action tags evaluate their tokens when they’re executed (after successful form submission). This means that values passed into the form such as URL parameters will need to be stored in a hidden form control (typically a TextBox with its Visibility property set to False). On the other hand, this enables Action tags to use Field tokens as their property values so these tags can use values input by the user in the form.

- **FirstNameField**: Optional. If you would like the tag to pass the user’s FirstName along to subsequent actions, specify the field name to use here. Later actions can then use that name in their field tokens. If not specified, this information will not be inserted into the action stream for use by later action tags.

- **LastNameField**: Optional. If you would like the tag to pass the user’s LastName along to subsequent actions, specify the field name to use here. Later actions can then use that name in their field tokens. If not specified, this information will not be inserted into the action stream for use by later action tags.

- **Password**: REQUIRED: The password for the user account that will be logged-in. Typically this will be a Field Token that is bound to a `<Password>` form control.

- **RememberMe**: 

- **UserIdField**: Optional. If you would like the tag to pass the user’s UserId along to subsequent actions, specify the field name to use here. Later actions can then use that name in their field tokens. If not specified, this information will not be inserted into the action stream for use by later action tags.

- **Username**: REQUIRED: The username of the account to be logged in. Typically this will be a Field Token bound to a `<TextBox>` form control.

- **UsernameField**: Optional. If you would like the tag to pass the user's Username along to subsequent actions, specify the field name to use here. Later actions can then use that name in their field tokens. If not specified, this information will not...
be inserted into the action stream for use by later action tags.

Example

```xml
<AddForm>
  <Login Username='[[Uname]]' Password='[[Pword]]' UsernameField="UsrName" UserIdField="UsrId" />
  <Redirect Target="~/Home.aspx?uid=[[UsrId]]&un=[[UsrName]]" />
</AddForm>
```

In the example above note how we're using Field Tokens ("[[Uname]]" and "[[Pword]]") to grab the data from the form controls. Additionally, we're telling the Login tag to add the user's Username and UserID to any tags "downstream". The field names that will be used are "UsrName" for the Username and "UsrId" for the UserID. There is one "downstream" action - the <Redirect> tag. It's using those fields in its Target property to feed the data into the URL. NOTE that in this specific case (and unlike most other XMod Pro forms) we can simply put the field tokens "[[UsrName]]" and "[[UsrId]]" directly into the target property without having to use the JOIN() function.

Syntax

```xml
<Panel AccessKey="string" BackColor="color name|#dddddd" BackImageUrl="uri" BorderColor="color name|#dddddd" BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset" BorderWidth="size" CssClass="string" DefaultButton="string" Font-Bold="True|False" Font-Italic="True|False" Font-Names="string" Font-Overline="True|False" Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large" Font-Strikeout="True|False" Font-Underline="True|False"ForeColor="color name|#dddddd" Height="size" HorizontalAlign="NotSet|Left|Center|Right|Justify" ID="string" ScrollBars="None|Horizontal|Vertical|Both|Auto"
```
ShowRoles="Role1Name,Role2Name"
Style="string"
Visible="True|False"
Width="size"
Wrap="True|False">

...HTML, Text, and Control Tags...

</Panel>

Remarks

This tag can be used solely as a container (much like the DIV tag in HTML). More often, it will be used to only show portions of a form to members of particular roles. To do this, use the ShowRoles attribute.

- **AccessKey**: Gets or sets the access key that allows you to quickly navigate to the control.
- **BackColor**: Color of the background of the control.
- **BackImageUrl**: Sets the URL of the background image for the panel control.
- **BorderColor**: Color of the border around the control
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **DefaultButton**: Set this to the ID of a push-button <AddButton>, <UpdateButton> or <CancelButton> on your form and it will be "clicked" when the user presses the ENTER key. Link and Image buttons may work for this as well, but are not supported. Link buttons, for instance, work on IE but not in Firefox.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **HorizontalAlign**: Sets the horizontal alignment of controls within the panel.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **ScrollBars**: Sets the visibility and position of scroll bars for the control:
  - **None**: No scroll bars are shown (default)
  - **Horizontal**: Only a horizontal scroll bar is shown
  - **Vertical**: Only a vertical scroll bar is shown
  - **Both**: Both horizontal and vertical scroll bars are shown
  - **Auto**: Horizontal and/or vertical scroll bars are show - only if necessary
- **ShowRoles**: A comma-delimited list of security role names. When specified, only members of the security roles will be shown the content of the panel. Note that if you are logged-in as host, you will see the panel contents even though the account may not be a member of one of the specified roles. To accurately test the functionality, you should login as a non-host/superuser account.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.
- **Wrap**: If true (default), content within the panel wraps. If false, content does not wrap.
Example

<addform>
  ...
  <Panel ShowRoles="Editor">
    <checkbox id="chkApproved" datafield="Approved" datatype="boolean" text="Approved?" />
  </Panel>
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="Last Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <addbutton text="Add"/> &nbsp; <cancelbutton text="Cancel"/>
      </td>
    </tr>
  </table>
</addform>

<Password>

Syntax Remarks Example

The Password tag renders as a single-line text input box at run time that masks user input so that passwords and other sensitive data isn’t viewable by others looking at the screen.
**Syntax**

```xml
<Password
   AccessKey="string"
   BackColor="color name|#dddddd"
   BorderColor="color name|#dddddd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
   BorderRadius="size"
   CssClass="string"
   DataType="string|int32|int64|boolean|...."
   Font-Bold="True|False"
   Font-Italic="True|False"
   Font-Names="string"
   Font-Overline="True|False"
   Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"
   Font-Strikeout="True|False"
   Font-Underline="True|False"
   ForeColor="color name|#dddddd"
   Height="size"
   ID="string"
   MaxLength="integer"
   Nullable=""True|False"
   ReadOnly=""True|False"
   Style="string"
   TabIndex="integer"
   ToolTip="string"
   Visible="True|False"
   Width="size"
/>
```

**Remarks**

If no value is supplied for the "datatype" attribute, a data type of `string` is assumed.

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.
- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **DataType**: The type of data this control is supplying to the data commands. This is a **Database type**. Valid values are: string (default), int32, int64, boolean, . This attribute is required if the control will participate in operations with your form's data commands.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **Nullable**: If True (the default is False), the control will return a DBNull value if the control is blank or contains just whitespace. If a DBNull value is passed to the control, the control will be set to an empty string.
- **ReadOnly**: If True, the contents of the control cannot be changed. The default value is False.
- **MaxLength**: The maximum number of characters allowed in the text box.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.

**Example**

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtPassword" text="Enter your password" />
        <password id="txtPassword" datafield="Password" datatype="string" />
      </td>
    </tr>
    <tr colspan="2">
      <addbutton text="Add"/> <cancelbutton text="Cancel"/>
    </tr>
  </table>
</addform>
```

**<RadioButton>**

The RadioButton tag renders as a radio button and associated label at run-time. Using a series of RadioButton controls, along with the Group-Name property, you can create a list of mutually exclusive options. This can be a useful alternative to using a RadioButtonList. With the RadioButtonList, only one value is sent to the datasource. If you use RadioButton controls that are grouped, you get the same mutually-exclusive options, but you also have the ability to store the value of each RadioButton in its own data field.
Syntax

<RadioButton
   AccessKey="string"
   AccessKey="string"
   BackColor="#dddddd"
   BorderColor="#dddddd"
   BorderStyle="None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
   BorderWidth="size"
   Checked="True|False"
   CssClass="string"
   DataField="string"
   DataType="Boolean"
   DataValueField="string"
   Font-Bold="true|false"
   Font-Italic="true|false"
   Font-Names="string"
   Font-Overline="true|false"
   Font-Size="small|small|medium|small|medium|large|large|xx-large|xx-large"
   Font-Strikeout="true|false"
   Font-Underline="true|false"
   ForeColor="#dddddd"
   GroupName="string"
   Height="size"
   ID="string"
   Nullable="true|false"
   Style="string"
   TabIndex="integer"
   TextAlign="left|right"
   ToolTip="string"
   Visible="true|false"
   Width="size"
/>

Remarks

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units.
- **Checked**: Determines whether the radio button is checked.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.
- **DataField**: Name of the parameter in the <submitcommand> which will be filled with this control's data when the form is submitted and/or the parameter in the <selectcommand> which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. Valid values are: boolean only. This attribute is required if the control will participate in operations with your form's data commands.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More.
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **GroupName**: The name of the group the radio button belongs to. Use this to create a grouping of radio buttons that form a mutually exclusive list of options. When this property is specified, only one radio button may be selected within that group at any given time.
- **Height**: Height of the control, specified in units.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
**Nullable**: If this is set to True, the control will return a DBNull value if the control has not been checked. If a DBNull value is passed to this control, regardless of the Nullable setting, the control will be un-checked.

**Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

**Text**: The caption to display next to the radio button.

**TextAlign**: The alignment of the text label with respect to its associated check box. Valid values are Left and Right. Default value is Right.

**Visible**: Determines if the control is visible (true) or hidden (false)

**Width**: Width of the control in [units](https://example.com/units).

---

**Example**

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label target="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <radiobutton id="optAgree" datafield="AgreeToTerms" datatype="boolean" text="I agree to be bound by the terms of the contract" />
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <addbutton text="Add"/> <cancelbutton text="Cancel"/>
      </td>
    </tr>
  </table>
</addform>
```

---

**RadioButtonList**

**Syntax**

- `<RadioButtonList>` tag renders as a series of mutually-exclusive option button at run-time.
Syntax

<RadioButtonList
    AccessKey="string"
    AppendDataBoundItems="True|False"
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge | Inset|Outset"
    BorderWidth="size"
    CellPadding="integer"
    CellSpacing="integer"
    CssClass="string"
    DataSourceID="string"
    DataTextField="string"
    DataTextFormatString="string"
    DataValueField="string"
    Enabled="True|False"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    ID="string"
    Nullable="True|False"
    RepeatColumns="integer"
    RepeatDirection="Horizontal|Vertical"
    RepeatLayout="Table|Flow"
    Style="string"
    TabIndex="integer"
    TextAlign="Left|Right"
    ToolTip="string"
    Visible="True|False"
    Width="size">
    <listitem value="string" selected="True|False">Item1</listitem>
    <listitem value="string">Item2</listitem>
    ...
</RadioButtonList>

Remarks

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)
- **AppendDataBoundItems**: If True, items retrieved from a <controldatasource> tag will be appended to the list of items already defined in the control. This only applies if the control is bound to such a tag. The default value is False.
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units.
- **CellPadding**: For table layouts, sets the distance (in pixels) between the border and the content of the cells.
- **CellSpacing**: For table layouts, sets the distance (in pixels) between cells.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.

- **DataSourceId**: If this control's data is supplied by a `<controldatasource>` tag, specify that tag's ID in this attribute. This attribute is required only if the control's data is supplied via a `<controldatasource>` tag.

- **DataTextField**: When using a `<controldatasource>` this attribute specifies the column name in that datasource that supplies each list item's display text. This attribute is required if the control's data is supplied via a `<controldatasource>` tag.

- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. Valid values are: string (default), int32, int64, boolean, . This attribute is required if the control will participate in operations with your form's data commands.

- **DataValueField**: When using a `<controldatasource>` this attribute specifies the column name in that datasource that supplies each list item's hidden value. This attribute is required only if the control's data is supplied via a `<controldatasource>` tag.

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More]

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in units.

- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.

- **Nullable**: If True (the default is False), this control returns a DBNull value if no item has been selected. If a DBNull value is passed to the control, regardless of the Nullable setting, all items will be de-selected.

- **RepeatColumns**: Defines the number of columns to use when laying out the checkboxes.

- **RepeatDirection**: Determines if the control displays vertically or horizontally.

- **RepeatLayout**: Use this attribute to specify whether the items in the control are displayed in a table. If this attribute is set to Table the items in the list are displayed in a table. If this attribute is set to Flow, the items in the list are displayed without a table structure.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **TextColor**: The alignment of the text label with respect to its associated radio button. Valid values are Left and Right. Default value is Right.

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Width of the control in units.

- **Usage**: The control allows `<listitem>` child tags which define the items that will appear in the list. The control can also be bound to a `<controldatasource>` tag. To do so, specify the ID of the `<controldatasource>` tag in the DataSourceId attribute, the name of the column in the data source that should supply the display text for each list item in the DataTextField attribute, and the column in the data source that should supply the hidden value of each list item in the DataValueField attribute.

**Example**

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label target="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label target="rblColors" text="Favorite Color" />
        <radiobuttonlist id="rblColors" datafield="FavoriteColors" datatype="string">
          <listitem value="#00FF00">Green</listitem>
          <listitem value="#FF0000" selected="true">Red</listitem>
          <listitem value="#0000FF">Blue</listitem>
        </radiobuttonlist>
      </td>
    </tr>
  </table>
</addform>
```
The Register tag enables you to use third-party controls in your XMod Pro forms. Note that only controls created specifically for XMod Pro can be used.

### Syntax

```xml
<Register
    TagPrefix="string"
    Namespace="string"
    Assembly="string"
/>
```

### Remarks

- Should you choose to use third-party controls, you'll need to add a Register tag to your form definition for each collection. Register tags tell XMod Pro where to find the controls you use. You only use the tag once per library. The register tag is declared outside the `<AddForm>` and `<EditForm>` tags. This allows you to use the library in both forms without having to duplicate the register tag.
- **TagPrefix**: A short series of letter and numbers that you use as part of the control's tag. It helps XMod Pro determine what library the control belongs to.
- **Namespace**: The namespace in which custom control resides. This information should be supplied by the control developer.
- **Assembly**: This is the name of the assembly (DLL) in which the controls reside. Note, you do not specify the path to the DLL or the ".dll" extension. This information should be supplied by the control developer.
Example

```xml
<addform>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="FirstName" />
        <ctb:CoolTextBox id="txtFirstName" makeitcool="true" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="LastName" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr colspan="2">
      <addbutton text="Add" />
      <cancelbutton text="Cancel" />
    </tr>
  </table>
</addform>
```

Syntax Remarks Example

New to Version 4.0! The Redirect action tag will send the user to the specified URL (Target) at run-time. In previous versions you could set a redirect target on the Add and Update button tags and you still can. However, the Redirect tag allows you to perform conditional redirects based on form data.

Syntax

```xml
<Redirect
  If="comma-delimited list of Redirect addresses"
  Target="Redirect address"
  Method="string" />
```

Remarks

- The Redirect action is only executed if the form has been successfully submitted. If there is a validation error or an error is thrown from the database, the action will not be performed.

- **Order Is Important**: Action tags are executed sequentially, so the order they appear within the form can be important. As an example, if one action fails with an error, all actions prior to the failed action will have executed. Those that occur after the failed action will not be executed. Additionally, some actions may have the ability to modify form values (this modification occurs after any form data has been sent to the database) - i.e. process form values, do calculations on them, transform them, even add and remove values from the list. Those changes will affect any action tags that are executed downstream that use Field tokens.

- **Using Tokens**: Unlike most form tags, which evaluate their tokens when the form is loaded, Action tags evaluate their tokens when they’re executed (after successful form submission). This means that values passed into the form such as URL parameters will need to be stored in a hidden form control (typically a TextBox with its Visibility property set to False). On the other hand, this enables Action tags to use Field tokens as their property values so these tags can use values input by
the user in the form.

- **If**: An expression that, when it evaluates to True, indicates the Redirect should be executed. No other redirects will be processed because a user can only be sent to one destination at a time. The If property allows you to perform conditional redirection. All redirect tags will be evaluated in the order they appear on the form. If the first tag's If evaluates to False, XMod Pro will move on to the 2nd Redirect tag. It will continue to iterate through the Redirect tags until one evaluates to True or doesn't specify an If property. If no Redirects evaluate to True, normal processing takes place. This could mean the redirect on the clicked button is executed or, if there is not, normal redirection back to the calling page.

Typically you'll use this attribute if you want to only send an Redirect if a user selects a certain value in your form.

**Example**: `<Redirect If='[[Department]] = Sales' Target="/mysite.com/sales.aspx" />

In this example we are taking the value of the "Department" column and comparing it to "Sales" If they are equal, the user will be sent to the sales.aspx page. If they are not, XMod Pro will look for the next Redirect tag and evaluate it.

**NOTE**: Comparisons are text-only and are not case-sensitive. You can test for equality using the "=" operator or inequality using the "<>" operator.

- **Method**: Determines how the redirection will be sent. The default value is Get. Valid values are:
  - **Get**: The user is redirected via the HTTP GET method (i.e. the normal method when navigating between web pages)
  - **Post**: The user is redirected via the HTTP POST method.

- **Target**: This is the URL to which the user will be sent if the IF property evaluates to True or there is no IF property specified. You can use a period (.) for the Target property's value. The period acts as shortcut to redirect to the current page.

**Example**

```xml
<AddForm>
  <SubmitCommand CommandText="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
        <Label For="txtFirstName" Text="First Name" />
        <TextBox Id="txtFirstName" DataField="FirstName" DataType="string" />
      </td>
    </tr>
    <tr>
      <td>
        <Label For="txtLastName" Text="Last Name" />
        <TextBox Id="txtLastName" DataField="LastName" DataType="string" />
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <AddButton Text="Add" Redirect="/Find.aspx" />
        <CancelButton Text="Cancel" />
      </td>
    </tr>
  </table>
  <Redirect Target="/Find.aspx?ln-Smith" Method="Get" If='"[[LastName]] = Smith" />
  <Redirect Target="/Find.aspx?ln-Jones" Method="Get" If='"[[LastName]] = Jones" />
</AddForm>
```
In the example above, there are 3 possible redirections that can occur based on user input. If the user enters a last name of "Smith" or "smith" she will be redirected to the URL: /Find.aspx?ln=Smith. If the user enters "Jones" or "jones" then they will be redirected to the URL: /Find.aspx?ln=Jones. If the user doesn't enter anything or enters some other name or value, they will be sent to the default URL specified on the AddButton tag: /Find.aspx where, presumably, all records - regardless of last name - would be listed.

Back to top

SilentPost

Syntax Remarks Example

New to Version 4.0! The SilentPost action tag will create an HTTP POST request and send it to the specified URL (Target) at run-time. This happens behind the scenes (i.e. silently) without user interaction.

Syntax

<SilentPost
    Url="URL that should receive the POST request">

    Optional 1 or more Field child tags that contain data to pass to the URL
    <Field Name="string" Value="string" />

</SilentPost>

Remarks

- The SilentPost action is only executed if the form has been successfully submitted. If there is a validation error or an error is thrown from the database, the action will not be performed.

- **Order Is Important**: Action tags are executed sequentially, so the order they appear within the form can be important. As an example, if one action fails with an error, all actions prior to the failed action will have executed. Those that occur after the failed action will not be executed. Additionally, some actions may have the ability to modify form values (this modification occurs after any form data has been sent to the database) - i.e. process form values, do calculations on them, transform them, even add and remove values from the list. Those changes will affect any action tags that are executed downstream that use Field tokens.

- **Using Tokens**: Unlike most form tags, which evaluate their tokens when the form is loaded, Action tags evaluate their tokens when they're executed (after successful form submission). This means that values passed into the form such as URL parameters will need to be stored in a hidden form control (typically a TextBox with its Visibility property set to False). On the other hand, this enables Action tags to use Field tokens as their property values so these tags can use values input by the user in the form.

- **Url**: The URL that will receive the POST request and data.

- **Fields**: You can optionally add one or more <Field> child tags to the SilentPost. These enable you to send data to the URL, specifying a Name and Value for each.

Back to top
Example

```xml
<AddForm>
  <SilentPost Url="http://mysite.com/PostTest.aspx">
    <Field Name="param1" Value="1" />
    <Field Name="param2" Value='[[P2]]' />
  </SilentPost>

  <TextBox Id="txtParam2" DataField="P2" DataType="String"/>
</AddForm>
```

In the example above, we've setup a SilentPost that will send a POST request to http://mysite.com/PostTest.aspx. The request will include field called "param1" with a value of "1" and a field with a name of "param2" and a value that is pulled from the txtParam2 TextBox.

Back to top

SyntaxBlock

Syntax | Remarks | Example
--- | --- | ---

The ScriptBlock tag is used to inject HTML `<script>` tags into one of several different locations in the page. Typically this is used to insert JavaScript functions and/or libraries into the page. You can also insert `<style>` tags into the page using this tag.

Syntax

```xml
<ScriptBlock
  ScriptId="string"
  BlockType="HeadScript|ClientScript|StartupScript|ClientScriptInclude"
  RegisterOnce="True|False"
  Url="url"
>
  <script type="text/javascript" ...>
    ...Javascript...
  </script>
</ScriptBlock>
```

Remarks

- **scriptId**: This is an identifier for your block that uniquely identifies it within the hosting page - across modules. It is used when registering your script block and is required in order to prevent the block from being inserted more than once in the page.
- **blocktype**: This attribute allows you to specify which where in the page the script should be rendered. The default value is ClientScript
  - **HeadScript**: The script block will be inserted between the `<head>` and `</head>` section of the page.
  - **ClientScript**: The script block will be inserted near the top of the page.
  - **ClientScriptInclude**: Use this block type to insert a `<script>` tag that links to an external file. This is useful for including Javascript libraries.
  - **StartupScript**: The script block will be inserted near the bottom of the page.
- **registeronce**: If this value is set to True, the tag will first check to see if a code block with ScriptId has been registered in the page. If not, it will register your block. If it has been registered already, then no action is taken. If this value is False, the default value, then your script block will be inserted, regardless of any previously registered block. The RegisterOnce is only available for ClientScript, ClientScriptInclude, and StartupScript block types.
- **Url**: If the BlockType is set to ClientScriptInclude, this is the path to the Javascript file you wish to include. It is ignored if BlockType is set to a different value. You may optionally use the tilde (~) character in the URL to represent the path to the root of the web application.

---

**Example**

```xml
<AddForm>
  <ScriptBlock ScriptId="AlertScripts" RegisterOnce="true">
    <script type="text/javascript">
      function helloWorld()
      {
        alert('Hello World');
      }
      function goodbyeWorld()
      {
        alert('Goodbye Cruel World');
      }
      function showMessage(sMessage)
      {
        alert(sMessage);
      }
    </script>
  </ScriptBlock>
  <table width="100%">
      <tr>
        <td width="250" valign="top">
            <!-- SCRIPT BLOCK EXAMPLE -->
            <a href="#" onclick="helloWorld();">Hello World</a><br />
            <a href="#" onclick="goodbyeWorld();">Goodbye</a><br />
            <a href="#" onclick="showMessage('Hello and Goodbye')">Show Message</a>
        </td>
      </tr>
  </table>
</AddForm>
```

---

**Tabstrip**

The Tabstrip tag renders as a series of tabs that, when clicked, shows the corresponding content within that tab. This control requires that Javascript be enabled in the browser to function correctly.

**Syntax**

```xml
<Tabstrip

  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  HoverBackColor="color name|#dddddd"
  HoverForeColor="color name|#dddddd"
  SelectedBackColor="color name|#dddddd"
  SelectedForeColor="color name|#dddddd"

  >
```

---

**Remarks**

- **BackColor**: Color of the tab background.
- **BorderColor**: Color of the border around the tab.
- **Font-Bold**: True if the tab text is bold, False otherwise.
- **Font-Italic**: True if the tab text is italicized, False otherwise.
- **Font-Names**: Font names or family.
- **Font-Overline**: True if the tab text has an overline, False otherwise.
- **Font-Size**: Size of the tab text.
- **Font-Strikeout**: True if the tab text is struck out, False otherwise.
- **Font-Underline**: True if the tab text is underlined, False otherwise.
- **ForeColor**: Color of the text in the tab.
- **Height**: Height of the tab.
- **HoverBackColor**: Color of the tab background when the mouse hovers over it.
- **HoverForeColor**: Color of the tab text when the mouse hovers over it.
- **SelectedBackColor**: Color of the tab background when it is selected.
- **SelectedForeColor**: Color of the tab text when it is selected.
Remarks

This tag allows you to create a tabbed user interface for your form much like a Rolodex or set of manila folders. When a tab is clicked, it becomes the selected tab and its associated content is shown, while any previously visible tab content is hidden. The tabstrip relies on and requires that Javascript be enabled in your users' browsers. Using the <tabstrip> control you can quickly create an attractive interface to turn larger forms into paged forms, presenting data to the user in smaller, more manageable chunks.

- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **HoverBackColor**: Color of the background of each tab when the mouse hovers over it.
- **HoverForeColor**: Color of the text in the tab when the mouse hovers over it.
- **SelectedBackColor**: Color of the background of the currently selected tab.
- **SelectedForeColor**: Color of the text of the currently selected tab.
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.

Example

```
<addform>
  ...
  <Tabstrip Font-Bold="True" Font-Names="Arial, Helvetica, sans-serif"
            HoverBackColor="Black" HoverForeColor="White">
    <Tab Text="Customer Info">
      <table>
        <tr>
          <td>
            <label for="txtFirstName" text="First Name" />
          </td>
          <td>
            <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
          </td>
        </tr>
        <tr>
          <td>
            <label for="txtLastName" text="Last Name" />
          </td>
          <td>
            <textbox id="txtLastName" datafield="LastName" datatype="string" />
          </td>
        </tr>
      </table>
    </Tab>
    <Tab Text="Notes">
      ...
    </Tab>
  </Tabstrip>
</addform>
```
The Text tag renders as un-decorated text at run-time.

**Syntax**

```xml
<Text>
  DataField="string"
  Nullable="True|False"
</Text>
```

**Remarks**

The main purpose of the Text tag is to render the value of a column from the form's SelectCommand in the form. For instance, you may be retrieving the user's last login date and want display that to the user on the form. So, you use the Text tag along with some HTML and text to your form like:

```html
<strong>Last Login Date: <Text datafield="LastLogin" /></strong>
```

You may also find the Text tag useful when combined with Javascript. The Text tag utilizes one-way binding. It can receive data from a SelectCommand, but it does not participate in the SubmitCommand. In other words, data in a Text tag is not saved to the database.

**Attributes**

- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **Nullable**: If True (the default is False), the control will return a DBNull value if the control is blank or contains just whitespace. If a DBNull value is passed to the control, the control will be set to an empty string.
Example

```html
<addform>
  <a href="#" onclick="alert('<Text datafield="LastLogin"/>')">Click for Last Login Date</a>
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <Textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    ...
    <tr>
      <td colspan="2">
        <addbutton text="Add"/>
        <cancelbutton text="Cancel"/>
      </td>
    </tr>
  </table>
</addform>
```

Textarea

**Syntax**

```html
<Textarea 
  AccessKey="string" 
  BackColor="color name|#dddddd" 
  BorderColor="color name|#dddddd" 
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset" 
  BorderWidth="size" 
  CharacterCount="None|CountDown|CountUp" 
  CharacterCountClass="CSS Class Name" 
  CharacterCountLabel="string" 
  Columns="integer" 
  CssClass="string" 
  DataField="string" 
  DataType="string|int32|int64|boolean|...." 
  Font-Bold="True|False" 
  Font-Italic="True|False" 
  Font-Names="string" 
  Font-Overline="True|False" 
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large" 
  Font-Strikeout="True|False" 
  Font-Underline="True|False" 
 ForeColor="color name|#dddddd" 
  Height="size" 
  HtmlEncode="True|False" 
  ID="string" 
  MaxLength="integer" 
  Nullable="True|False" 
  ReadOnly="True|False" 
  Rows="integer" 
  Style="string" 
  TabIndex="integer" 
  ToolTip="string" 
```
Remarks

If no "datatype" is supplied, the Textarea control defaults to string.

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control.
- **BorderStyle**: Style of the border around the control.
- **BorderWidth**: Width of the border around the control, specified in units.
- **CharacterCount**: Defaults to None. When set to CountUp, the number of characters the user has typed into the control will be displayed just after the control. If the value is set to CountDown, the number of characters remaining will be displayed. Remaining characters are calculated based on the MaxLength property. *This feature requires Javascript and jQuery.* (New to version 2.1)
- **CharacterCountLabel**: Text to be displayed next to the number displayed in CharacterCount. This label is only displayed when CharacterCount is set to CountUp or CountDown. NOTE: You should precede your label text with a space. Otherwise the text will butt up against the character count number. (New to version 2.1)
- **CharacterCountClass**: When CharacterCount is CountUp or CountDown, you can style the the displayed number and the CharacterCountLabel (if specified) by specifying a CSS class name for this property. (New to version 2.1)
- **Columns**: The display width, in characters, of the control.
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.
- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. Valid values are: string (default), int32, int64, boolean, . This attribute is required if the control will participate in operations with your form's data commands.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **HtmlEncode**: When set to true, the content of the control will have any HTML encoded before sending it to the SubmitCommand for processing. Note that setting this to true can help protect against scripting attacks but it will also enlarge the size of the text that is saved (<) becomes (&lt;), for instance.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **MaxLength**: For Use ONLY With CharacterCount attribute. Specifies the maximum number of characters the CharacterCount feature should allow. If not specified, the user’s input will not be limited. IMPORTANT: Unlike the TextBox's MaxLength attribute, this does not ensure that no more than the specified number of characters will be entered. You should still use normal validation methods for that. (New to version 2.1)
- **Placeholder**: New to version 4.2. If this property is set, the value will be displayed in the textbox when it is empty. It is an HTML5 feature that will only function on capable browsers (all other browsers will ignore it). The purpose is to provide the user with some explanatory text in the control itself prior to the user entering any text. For instance, a contact form's About Me textarea might have "Tell us about yourself" as the Placeholder. When the user tabs into the control, the text will disappear. If the user tabs out without entering anything, the "Tell us about yourself" text will re-appear. The Placeholder value is not a default value for the control and will not be sent to the database.
- **Nullable**: If True (the default is False), the control will return a DBNull value if the control is blank or contains just whitespace. If a DBNull value is passed to the control, the control will be set to an empty string.
- **ReadOnly**: If True, the contents of the control cannot be changed. The default value is False.
- **Rows**: The number of rows displayed in the textarea control.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **TabIndex**: Sets the tab index for the control
- **Tooltip**: In browsers that support it, sets the text to display when the mouse pointer hovers over the control
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in **units**.
- **Wrap**: If True, text in the textarea wraps. If False, text does not wrap. The default is True.

**Example**

```html
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtBio" text="Bio" />
        <textarea id="txtBio" datafield="Bio" datatype="string" />
      </td>
    </tr>
    <tr colspan="2">
      <addbutton text="Add" />
      <cancelbutton text="Cancel" />
    </tr>
  </table>
</addform>
```

**<Textbox>**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Remarks</th>
<th>Example</th>
</tr>
</thead>
</table>

The Textbox tag renders as a single-line text input box at run time.
Syntax

```xml
<Textbox
   AccessKey="string"
   BackColor="color name|#ddddd"
   BorderColor="color name|#ddddd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
   BorderWidth="size"
   CssClass="string"
   DataField="string"
   DataType="string|int32|int64|boolean|...."
   Font-Bold="True|False"
   Font-Italic="True|False"
   Font-Names="string"
   Font-Overline="True|False"
   Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
   Font-Strikeout="True|False"
   Font-Underline="True|False"
   ForeColor="color name|#ddddd"
   Height="size"
   HtmlEncode="True|False"
   ID="string"
   MaxLength="integer"
   Nullable="True|False"
   Placeholder="string"
   ReadOnly="True|False"
   Style="string"
   TabIndex="integer"
   ToolTip="string"
   Visible="True|False"
   Width="size"
/>
```

Remarks

You will probably use the Textbox tag most often in your forms. It is the standard method for input within most forms and is perfect for entering names, addresses, and other single-line input. The "datatype" attribute defaults to string.

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)
- **BackColor**: Color of the background of the control.
- **BorderColor**: Color of the border around the control
- **BorderStyle**: Style of the border around the control
- **BorderWidth**: Width of the border around the control, specified in units
- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control
- **DataField**: Name of the parameter in the `<submitcommand>` which will be filled with this control's data on when the form is submitted and/or the parameter in the `<selectcommand>` which will supply this control's data when the form is loaded. This attribute is required if the control will participate in operations with your form's data commands.
- **DataType**: The type of data this control is supplying to the data commands. This is a Database type. This attribute is required if the control will participate in operations with your form's data commands.
- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **HtmlEncode**: When set to true, the content of the control will have any HTML encoded before sending it to the SubmitCommand for processing. Note that setting this to true can help protect against scripting attacks but it will also enlarge the size of the text that is saved (<) becomes (&lt;), for instance.
- **ID**: Name, consisting of letters and numbers, beginning with a letter, that uniquely identifies the control within the form.
- **Nullable**: If True (the default is False), the control will return a DBNull value if the control is blank or contains just whitespace. If a DBNull value is passed to the control, the control will be set to an empty string.
- **Placeholder**: New to version 4.2. If this property is set, the value will be displayed in the textbox when it is empty. It is an HTML5 feature that will only function on capable browsers (all other browsers will ignore it). The purpose is to provide the user with some explanatory text in the control itself prior to the user entering any text. For instance, a contact form's Email textbox might have "Enter your email address" or simple "Email" as the Placeholder. When the user tabs into the control, the text will disappear. If the user tabs out without entering anything, the "Email" text will re-appear. The value here is not a default value for the control and will not be sent to the database.
- **ReadOnly**: If True, the contents of the control cannot be changed. The default value is False.
- **MaxLength**: The maximum number of characters allowed in the text box.
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **TabIndex**: Sets the tab index for the control
- **Tooltip**: In browsers that support it, sets the text to display when the mouse pointer hovers over the control
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.

**Example**

```xml
<addform>
    ...
    <table>
        <tr>
            <td>
                <label for="txtFirstName" text="First Name" />
                <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
            </td>
        </tr>
        <tr>
            <td>
                <label for="txtLastName" text="Last Name" />
                <textbox id="txtLastName" datafield="LastName" datatype="string" />
            </td>
        </tr>
        <tr colspan="2">
            <addbutton text="Add"/&nbsp;<cancelbutton text="Cancel"/>
        </tr>
    </table>
</addform>
```

**<UpdateButton>**

**Syntax**

The UpdateButton tag renders as a push-button at run-time that, when clicked, initiates the update process, executing the `<SubmitCommand>` of the associated `<EditForm>`. 
Syntax

```xml
<UpdateButton
  AccessKey="string"
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  FontStyle="True|False"
  Font-Italic="True|False"
  Font-Face="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  OnClientClick="string"
  Redirect="url"
  RedirectMethod="Get|Post"
  Style="string"
  TabIndex="integer"
  ToolTip="string"
  Visible="True|False"
  Width="size" />
```

Remarks

- The update button should only be used in `<EditForm>` tags. It's purpose is to initiate the `<SubmitCommand>` associated with the `<EditForm>`.

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control.

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units.

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. More.

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in units.

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return
true then the control will perform its normal processing.

- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes. Field tokens may be used in the redirect attribute. However, function tokens such as [[Portal:ID]], [[Join()]], [[User:ID]], etc. cannot be used. When field tokens are used, they are URL Encoded. New to version 4.0: You can use a period (.) for the Redirect property’s value. The period acts as shortcut to redirect to the current page.

- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post" IMPORTANT: When using "Post", the ID that you supply for your form controls determine the name of the field that is posted to the target URL, not the DataField.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Determines the width of the control in units.

---

**Example**

```xml
<editform>
  <selectcommand commandtext="SELECT * FROM Users WHERE UserId = @UserId" />
  <submitcommand commandtext="UPDATE Users SET FirstName=@FirstName, LastName=@LastName WHERE UserId=@UserId"/>
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="Last Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr colspan="2">
      <updatebutton text="Update"/> <cancelbutton text="Cancel"/>
    </tr>
  </table>
  <textbox id="txtUserId" datafield="UserId" datatype="int32" visible="false"/>
</editform>
```

---

**<UpdateImage>**

The UpdateImage tag renders as a clickable image at run-time. When clicked, the form executes the `<SubmitCommand>` associated with the `<EditForm>`.
Syntax

```xml
<UpdateImage
  AccessKey="string"
  AlternateText="string"
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  ImageAlign="NotSet|Left|Right|Baseline|Top|Middle|Bottom|AbsBottom|AbsMiddle|TextTop"
  ImageUrl="url"
  OnClientClick="string"
  Redirect="url"
  RedirectMethod="Get|Post"
  Style="string"
  TabIndex="integer"
  Text="string"
  ToolTip="string"
  Visible="True|False"
  Width="size" />
```

Remarks

- The `UpdateImage` tag should only be used in `<EditForm>` tags. Its purpose is to initiate the `<SubmitCommand>` associated with the `<EditForm>`

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines)

- **AlternateText**: Use this attribute's value will be used as the image's "alt" text. The "alt" text is generally used by screen reader software used by visually impaired users to identify the content of an image. It may also be used by search engines

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)
- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.
- **Height**: Height of the control, specified in units.
- **ImageAlign**: This attribute determines how the image will be aligned with respect to the other elements in its context.
- **ImageUrl**: Specify a URL to the image. You may use the tilde (~) character to represent the application’s root directory. For instance: ImageUrl="~/images/myimage.gif" might map to "/dnntestsite/images/myimage.gif" on your localhost development machine and "/images/myimage.gif" on your production server.
- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns false the control will not perform its normal processing. If you return true then the control will perform its normal processing..
- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes. Field tokens may be used in the redirect attribute. However, function tokens such as [[Portal:ID]], [[Join()]], [[User:ID]], etc. cannot be used. When field tokens are used, they are URL Encoded. New to version 4.0: You can use a period (.) for the Redirect property's value. The period acts as shortcut to redirect to the current page.
- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"
- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")
- **Visible**: Determines if the control is visible (true) or hidden (false)
- **Width**: Width of the control in units.

**Example**

```xml
<editform>
  <selectcommand commandtext="SELECT * FROM Users WHERE UserId = @UserId" />
  <submitcommand commandtext="UPDATE Users SET FirstName=@FirstName, LastName=@LastName WHERE UserId=@UserId" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="FirstName" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <td colspans="2">
      <UpdateImage AlternateText="Update" ImageUrl="~/images/update.gif" />
      <CancelImage AlternateText="Cancel" ImageUrl="~/images/cancel.gif" />
    </td>
  </table>
</editform>
```
The UpdateLink tag renders as a hyperlink at run-time. When clicked, the form executes the `<SubmitCommand>` associated with the `<EditForm>`.

**Syntax**

```xml
<UpdateLink>
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderRadius="size"
  CssClass="string"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  OnClientClick="javascript"
  Redirect="url"
  RedirectMethod="Get|Post"
  Style="string"
  Text="string"
  ToolTip="string"
  Visible="True|False"
  Width="size"/>
</UpdateLink>
```
**Remarks**

- **The UpdateLink tag should only be used in `<EditForm>` tags.** It's purpose is to initiate the `<SubmitCommand>` associated with the `<EditForm>`.

- **AccessKey**: In browsers that support it, this property can be set to a character on the keyboard that can be used to set focus to the control. For instance, setting the value to F allows the user to access the control by pressing Alt+F on their keyboard (for Windows machines).

- **BackColor**: Color of the background of the control.

- **BorderColor**: Color of the border around the control.

- **BorderStyle**: Style of the border around the control.

- **BorderWidth**: Width of the border around the control, specified in units.

- **CssClass**: Name of the Cascading Style Sheets (CSS) class used to style this control.

- **Font Properties**: A series of attributes such as font-bold, font-size, etc. that allow you to control how the text in the control is displayed. [More](#)

- **ForeColor**: Sets the foreground color (typically the color of the text) of the control.

- **Height**: Height of the control, specified in units.

- **OnClientClick**: Should you wish to perform some action on the client when the control is clicked, add your Javascript function call or script in this attribute. If your script returns `false` the control will not perform its normal processing. If you return true then the control will perform its normal processing.

- **Redirect**: Enables you to redirect the user to an alternative URL after the button is clicked. The redirection occurs after any form processes initiated by the button click completes. Field tokens may be used in the redirect attribute. However, function tokens such as `[[Portal:ID]], [[User:ID]], etc. cannot be used. When field tokens are used, they are URL Encoded. New to version 4.0: You can use a period (.) for the Redirect property's value. The period acts as shortcut to redirect to the current page.

- **RedirectMethod**: Determines the HTTP method by which the user is redirected: "Get" or "Post"

  IMPORTANT: When using "Post", the ID that you supply for your form controls determine the name of the field that is posted to the target URL, not the DataField.

- **Style**: Same as the HTML style attribute. It allows you to apply CSS styling to the control (e.g. "color: red; border: solid 1px black;")

- **Visible**: Determines if the control is visible (true) or hidden (false)

- **Width**: Width of the control in units.

**Back to top**

**Example**

```xml
<addform>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
```
<label for="txtFirstName" text="FirstName" />
<textbox id="txtFirstName" datafield="FirstName" datatype="string" />
</td>
</tr>
<tr>
<label for="txtLastName" text="FirstName" />
<textbox id="txtLastName" datafield="LastName" datatype="string" />
</td>
</tr>
<tr>
<td colspan="2">
<UpdateLink Text="Add" />
<CANCELLink Text="Cancel" />
</td>
</tr>
</table>
</addform>

<UpdateUser>

Syntax

New to Version 4.0! The UpdateUser tag will register a user in the DNN site and optionally add that user to one or more DotNetNuke security roles after the form has been successfully submitted.

IMPORTANT: You MUST take care to properly validate user input. Additionally you should place this tag only on forms that are properly secured so that only users you intend to have access can use the form.

Syntax

<UpdateUser
  Approved="True|False"
  City="string"
  Country="string"
  DisplayName="string"
  Email="string"
  FirstName="string"
  LastName="string"
  NewPassword="string"
  OldPassword="string"
  PostalCode="string"
  Region="string"
  Street="string"
  Telephone="string"
  Unit="string"
  UpdatePasswordOnNextLogin="True|False"
  UserId="integer">
  (NOTE: Property tags are optional)
  <Property Name="string" Value="string" />
  ...Additional Property Tags as needed...
</UpdateUser>

Remarks

- The UpdateUser action is only executed if the form has been successfully submitted. If there is a validation error or an error is thrown from the database, the action will not be performed.
• **Order Is Important**: Action tags are executed sequentially, so the order they appear within the form can be important. As an example, if one action fails with an error, all actions prior to the failed action will have executed. Those that occur after the failed action will not be executed. Additionally, some actions may have the ability to modify form values (this modification occurs after any form data has been sent to the database) - i.e. process form values, do calculations on them, transform them, even add and remove values from the list. Those changes will affect any Action tags that are executed downstream that use Field tokens.

• **Using Tokens**: Unlike most form tags, which evaluate their tokens when the form is loaded, Action tags evaluate their tokens when they're executed (after successful form submission). This means that values passed into the form such as URL parameters will need to be stored in a hidden form control (typically a TextBox with its Visibility property set to False). On the other hand, this enables Action tags to use Field tokens as their property values so these tags can use values input by the user in the form.

• **Approved**: Optional. True or False. If set to True, the user will be auto-approved when the user record is updated. If False, the user will be un-approved when the record is updated. If the property is not set, no change will take place.

• **City**: Optional. The city in which the user resides. If the property is not set, no change will take place.

• **Country**: Optional. The country in which the user resides. If the property is not set, no change will take place.

• **DisplayName**: Optional. The name to display when displaying the user's full name. Unlike the AddUser tag which attempts to create a DisplayName from the first and last names of the user when the DisplayName is not set, in the UpdateUser tag, if the property is not set, no change will take place.

• **Email**: Required. The user's email address. If the property is not set or if the property evaluates to an empty string, no change will take place.

• **FirstName**: Required. The user's first name. If the property is not set or if the property evaluates to an empty string, no change will take place.

• **LastName**: Required. The user's last name. If the property is not set or if the property evaluates to an empty string, no change will take place.

• **NewPassword**: The new password for the user account that will be updated. If the property is not set or if the property evaluates to an empty string or if OldPassword is not supplied (or is not correct), no change will take place.

• **OldPassword**: Required if NewPassword is supplied. The existing password for the user account that will be updated.

• **PostalCode**: Optional. The postal code (zip code in the US) associated with the user’s address. If the property is not set, no change will take place.

• **Property Tags**: These are optional child tags that allow you to specify one or more custom profile properties that will be set when the user is created.

• **Region**: Optional. The region (state in the US) where the user is located. If the property is not set, no change will take place.

• **Street**: Optional. The street address of the user.

• **Telephone**: Optional. The user’s telephone number. If the property is not set, no change will take place.

• **Unit**: Optional. The unit or apartment number associated with the user’s address. If the property is not set, no change will take place.
- **UpdatePasswordOnNextLogin**: Optional. True or False. When true, the user will be prompted to update his/her password when logging in next. If the property is not set, no change will take place.

- **UserId**: Required. A numeric value that uniquely identifies the user account to be updated. This must be the UserID assigned to the user when the account was created. You can use the `[[User:ID]]` token to get the current user’s ID or you can use a field token like `[[User]]` to retrieve a valid user ID from a form control (perhaps a drop-down list that contains users and their ID’s).

---

**Example**

```xml
<AddForm>
  <SelectCommand CommandText="SELECT @UserId AS UserId">
    <Parameter Name="UserId" Value='[[User:ID]]'DataType="Int32" DefaultValue="-1"/>
  </SelectCommand>
  <UpdateUser Email='[[Email]]' FirstName='[[FName]]' LastName='[[LName]]' UserId='[[User:ID]]'/>
</AddForm>
```

---

**Syntax**

**Remarks**

**Example**
The Validate tag whose type is set to "action" is referred to as an Action Validator and is used to display special error messages thrown from the action tags in your form.

**Syntax**

```xml
<Validate
  CssClass="string"
  Text="string"
  Type="Action"
/>
```

**Remarks**

The action validator is a special type of `<validate>` tag. When the "type" attribute is set to `action`, the control will display the error message thrown by an action tag such as `<AddUser>`. An example would be if the user attempts to register with a username that already exists in the site. A duplicate username error will be thrown and an appropriate message will be displayed by this validator. If you have a `<ValidationSummary>` tag on your form then the error message will be displayed there. If this validator has not been placed in your form, the message will still be displayed using the default error reporting mechanism.

**Example**

```xml
<AddForm>
  <AddUser RoleNames="Role1,Editors" Email='[[Email]]'
    FirstName='[[FName]]' LastName='[[LName]]'
    Username='[[Username]]' Password='[[Password]]' />
</AddForm>
```

```xml
<table>
  <tr>
    <td>
      <Label For="txtFirstName" Text="First Name" />
      <TextBox Id="txtFirstName" DataField="FName" DataType="string" />
      <Validate Type="Required" Target="txtFirstName" Text="***" Message="First Name is required." />
    </td>
  </tr>
  <tr>
    <td>
      <Label For="txtLastName" Text="Last Name" />
      <TextBox Id="txtLastName" DataField="LName" DataType="string" />
      <Validate Type="Required" Target="txtLastName" Text="***" Message="Last Name is required." />
    </td>
  </tr>
  <tr>
    <td>
      <Label For="txtEmail" Text="Email" />
      <TextBox Id="txtEmail" DataField="Email" DataType="string" />
      <Validate Type="Required" Target="txtEmail" Text="***" Message="An email address is required." />
      <Validate Type="Email" Target="txtEmail" Text="***" Message="Please enter a valid email address." />
    </td>
  </tr>
  <tr>
    <td>
      <Label For="txtUsername" Text="Username" />
      <TextBox Id="txtUsername" DataField="Username" DataType="string" />
      <Validate Type="Required" Target="txtUsername" Text="***" Message="Please enter a Username." />
    </td>
  </tr>
  <tr>
    <td>
      <Label For="txtPassword" Text="Password" />
    </td>
  </tr>
</table>
```
In the example above, we've highlighted the three key components at work here - the AddUser tag, the Validate tag, and the ValidationSummary tag. As you can see, there isn't much you have to do to use the action validator. Just place it on your form and, ideally, also have a ValidationSummary tag on the form for displaying any errors.

Syntax Remarks Example

The Validate tag whose type is set to "checkbox" is referred to as a CheckBox Validator and is used to ensure the user either checks or does not check a checkbox control.
Syntax

```xml
<Validate
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Display="Static|Dynamic"
    EnableClientScript="True|False"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
   ForeColor="color name|#dddddd"
    Height="size"
    Message="string"
    MustBeChecked="True|False"
    Target="string"
    Text="string"
    Type="Checkbox"
    Width="size"
/>```

Remarks

The checkbox validator is one type of the `<validate>` tag. When the "type" attribute is set to `checkbox`, the control prevents the form from being submitted if its associated checkbox control is checked or un-checked (depending on settings). You can associate a control with the `<validate>` tag by setting its "target" attribute to the ID of the control you wish to validate. The "message" attribute is the text that will be displayed to the user when validation fails. If you are using the `<validationsummary>` tag, then you can also supply a "text" attribute. When validation fails, the "text" will be displayed where your `<validate>` tag is and the "message" will be displayed in the `<validationsummary>`. The "display" attribute determines if the the `<validate>` tag will reserve space for its message in the page layout - typically resulting in blank space in your form - or whether it will dynamically allocate the space for the message when validation fails. The `<validate>` tag defaults to Dynamic display.

- **MustBeChecked**: True/False, default value is True. Determines what constitutes a valid state for the control. If set to True, then the user must check the box for the control to be valid. If the property is set to False, the user must un-check the control for it to be valid.
Example

```xml
<addform>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="Last Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="chkAgree" text="I Agree to the Terms of Service" />
        <checkbox id="chkAgree" datafield="Agree" datatype="boolean" />
        <validate type="checkbox" target="chkAgree" mustbechecked="True" message="You must enter a First Name" />
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <addbutton text="Add" /><cancelbutton text="Cancel" />
      </td>
    </tr>
  </table>
</addform>
```

<Validate type="checkboxlist">

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Remarks</th>
<th>Example</th>
</tr>
</thead>
</table>

The Validate tag whose type is set to "checkboxlist" is referred to as a CheckBoxList Validator and is used to ensure the user checks at least one box in the list. NOTE: This validator runs on the server only. A form must pass all client-side validation and be submitted to the server before this validation will trigger.
Syntax

<Validate
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge| Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  Display="Static|Dynamic"
  EnableClientScript="True|False"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
 ForeColor="color name|#dddddd"
  Height="size"
  Message="string"
  Target="string"
  Text="string"
  Type="CheckboxList"
  Width="size"
/>

Remarks

The checkboxlist validator is one type of the <validate> tag. When the "type" attribute is set to checkboxlist, the control prevents the form from being submitted if its associated control does not have any boxes ticked. You associate a control with the <validate> tag by setting its "target" attribute to the ID of the control you wish to validate. The "message" attribute is the text that will be displayed to the user when validation fails. If you are using the <validationsummary> tag, then you can also supply a "text" attribute. When validation fails, the "text" will be displayed where your <validate> tag is and the "message" will be displayed in the <validationsummary>. The "display" attribute determines if the <validate> tag will reserve space for its message in the page layout - typically resulting in blank space in your form -or whether it will dynamically allocate the space for the message when validation fails. The <validate> tag defaults to Dynamic display.
<addform>
  <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
  <table>
    <tr>
      <td>
        <label for="txtFirstName" text="First Name" />
        <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtLastName" text="Last Name" />
        <textbox id="txtLastName" datafield="LastName" datatype="string" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="cblColors" text="Favorite Color(s)" />
        <checkbox id="cblColors" datafield="FavColors" datatype="string" />
        <validate type="checkboxlist" target="cblColors" text="***" message="Everyone has a favorite color. What's yours? (Select at least one)" />
      </td>
    </tr>
  </table>
  <td colspan="2">
    <addbutton text="Add"/> &nbsp; <cancelbutton text="Cancel"/>
    <validationsummary />
  </td>
</addform>

<Validatetype="compare"/>

**Syntax Remarks Example**

The Validate tag whose type is set to "compare" is referred to as a Comparison Validator and is used to ensure the value of the target control is the same as a hard-coded value or the same as the value in a second control. It is useful when prompting the user for an email address or password.
Syntax

```xml
<Validate
    BackColor="color name"|#dddddd"
    BorderColor="color name"|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge| Inset|Outset"
    BorderWidth="size"
    CompareTarget="string"
    CompareValue="string"
    CssClass="string"
    DataType="String|Integer|Double|Date|Currency"
    Display="Static|Dynamic"
    EnableClientScript="True|False"
    Message="string"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
   ForeColor="color name"|#dddddd"
    Height="size"
    Operator="Equal|NotEqual|GreaterThan|GreaterThanEqual|LessThan|LessThanEqual|DataTypeCheck"
    Target="string"
    Text="string"
    Type="Compare"
    Width="size"
/>
```

Remarks

When the "type" attribute is set to compare, the control prevents the form from being submitted if its associated control does not match a value - either a hard-coded value or the value in a second control. You associate a control with the <validate> tag by setting its "target" attribute to the ID of the control you wish to validate. To compare the target control with the value of a second control, place the second control's ID in the "comparetarget" attribute and do not define the "comparevalue" attribute. To compare it with a hard-coded value, place that value in the "comparevalue" attribute and do not define the "comparetarget" attribute. If you set the "operator" attribute to DataTypeCheck then choose the datatypeto check-for using the "DataType" attribute. The "message" attribute is the text that will be displayed to the user when validation fails. If you are using the <validationsummary> tag, then you can also supply a "text" attribute. When validation fails, the "text" will be displayed where your <validate> tag is and the "message" will be displayed in the <validationsummary>.

Example

```xml
<addform>

  ...
  <table>
    <tr>
      <td>
        <label for="txtEmailOne" text="Email" />
        <textbox id="txtEmailOne" />
      </td>
    </tr>
    <tr>
      <td>
        <label for="txtEmail" text="Email" />
        <textbox id="txtEmail" datafield="Email" datatype="string" />
        <validate type="compare" target="txtEmail" comparetarget="txtEmailOne" message="The email addresses don't match" />
      </td>
    </tr>

  </table>

</addform>
```
The Validate tag whose type is set to "database" is referred to as a Database Validator and is used to display error messages thrown from the database.

**Syntax**

```xml
<Validate
    CssClass="string"
    Text="string"
    Type="Database"
/>
```

**Remarks**

The database validator is a special type of `<validate>` tag. When the "type" attribute is set to `database`, the control prevents the form from being submitted if the database throws an error or a value is returned via a specially-named output parameter. Unlike other `<validate>` tags, the database validator is not associated with a specific form control and only one is allowed per form. The database validator is used to display error messages returned from the database. These can be actual errors thrown by the database or a friendly error message returned using the specially named ERROR output parameter.

**Passing a friendly error message back to the form**

In some cases your stored procedure may want to inform the user that the data they submitted is invalid in some way. A good example is if a user is choosing a Team Name for a sports league and you want to ensure that no two Team Names are the same. If the user submits a name that already exists, you'd want to inform them of that, allowing them to choose a different name. Here's how you'd do that:

1. Set the `<SubmitCommand>` tag's `CommandType` property to: `StoredProcedure`
2. Add an OUTPUT parameter to the `<SubmitCommand>`. It **must** be named `ERROR` and its direction must be set to `Output` like so:
   ```xml
   <Parameter
        Name="ERROR"
        DataType="String"
        Size="250"
        Direction="Output"
    />
   ```
3. Optionally (though you will usually do this), add a `<Validate Type="Database" />` tag to your form.
4. Optionally add a `<ValidationSummary>` tag to your form if you don’t already have one.
5. If you don’t use the Validate/ValidationSummary combination of tags, the error will be reported to the end user using the standard XMod Pro reporting mechanisms.
6. In your stored procedure, set the @ERROR parameter to be an OUTPUT parameter and set its value to whatever message you want to return.

**Example**

```xml
<AddForm>
    <SubmitCommand
        CommandText="XMP_ReturnValueTester"
        CommandType="StoredProcedure">
    </SubmitCommand>
</AddForm>
```
Here's a sample stored procedure that is guaranteed to throw an error:

```sql
CREATE PROCEDURE [dbo].[XMP_DBThrownError_Tester]
   @FirstName nvarchar(255),
   @LastName nvarchar(255)
AS
BEGIN
   RAISERROR('Example Error Thrown',18,1)
END
```

Here's what the example form looks like after the user has submitted the form and the stored procedure has thrown the error:
The Validate tag whose type is set to "email" is referred to as an Email Validator and is used to ensure the value of the target control matches the form of a valid email address. It does not validate the email account is valid or active.

Syntax

```xml
<Validate
    BackColor="color|#dddddd"
    BorderColor="color|name|#dddddd"
    BorderStyle="None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Display="Static|Dynamic"
    EnableClientScript="True|False"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    Message="string"
    Target="string"
    Text="string"
    Type="Email"
    Width="size"
/>```

Remarks

When the "type" attribute is set to email, the control prevents the form from being submitted if its associated control's value does not match the pattern of a properly formed email address. It does not validate the email account is active or valid. The Email Validator is a handy short-cut. It uses a built-in pattern that should validate most forms of email address. If you find it insufficient for specific situations, you can always use the RegularExpressionValidator and use your own pattern.

You associate a control with the `<validate>` tag by setting its "target" attribute to the ID of the control you wish to validate. The "message" attribute is the text that will be displayed to the user when validation fails. If you are using the `<validationsummary>` tag, then you can also supply a "text" attribute. When validation fails, the "text" will be displayed where your `<validate>` tag is and the "message" will be displayed in the `<validationsummary>`.
Example

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtEmail" text="Email" />
        <textbox id="txtEmail" datafield="Email" datatype="string" />
        <validate type="email" target="txtEmail" message="Please enter a valid email address" />
      </td>
    </tr>
    <tr colspan="2">
      <addbutton text="Add"/>
      <cancelbutton text="Cancel"/>
    </tr>
  </table>
</addform>
```

<Validate type="range">

Syntax

The Validate tag whose type is set to "range" is referred to as a Range Validator and is used to ensure the value of the target control falls within the specified range of values. This can be used, for example, to ensure that only a limited number of tickets can be purchased - that the number of tickets ordered is at least one but not more than five.

Syntax

```xml
<Validate
  BackColor="color name|#dddddd"
  BorderColor="color name|#dddddd"
  BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
  BorderWidth="size"
  CssClass="string"
  DataType="String|Integer|Double|Date|Currency"
  Display="Static|Dynamic"
  EnableClientScript="True|False"
  Font-Bold="True|False"
  Font-Italic="True|False"
  Font-Names="string"
  Font-Overline="True|False"
  Font-Size="String|Smaller|Larger|XX-Small|X-Small|Small|Medium| Large|X-Large|XX-Large"
  Font-Strikeout="True|False"
  Font-Underline="True|False"
  ForeColor="color name|#dddddd"
  Height="size"
  MaximumValue="string"
  Message="string"
  MinimumValue="string"
  Target="string"
  Text="string"
  Type="Range"
  Width="size"
/>```
Remarks

When the "type" attribute is set to **range**, the control prevents the form from being submitted if its associated control's value does not match the regular expression pattern specified in the "validationexpression" attribute. You associate a control with the `<validate>` tag by setting its "target" attribute to the ID of the control you wish to validate. The "message" attribute is the text that will be displayed to the user when validation fails. If you are using the `<validationsummary>` tag, then you can also supply a "text" attribute. When validation fails, the "text" will be displayed where your `<validate>` tag is and the "message" will be displayed in the `<validationsummary>`.

Example

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtQuantity" text="Number of Tickets" />  
        <textbox id="txtQuantity" datafield="Quantity" datatype="int32" />
        <validate type="range" target="txtQuantity" minimumvalue="1" maximumvalue="5" message="You can only order between 1 and 5 tickets" datatype="Integer"/>  
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <addbutton text="Place Order"/> &nbsp; <cancelbutton text="Cancel"/>
      </td>
    </tr>
  </table>
</addform>
```

Syntax Remarks Example

The Validate tag whose type is set to "regex" is referred to as a Regular Expression Validator and is used to ensure the value of the target control matches the specified regular expression pattern. This can be used, for example, to ensure a phone number matches the format you specify or that an email address is properly formatted.
Syntax

<Validate
  BackColor="color name|#ddd|ddd" 
  BorderColor="color name|#ddd|ddd" 
  BorderStyle="None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset" 
  BorderRadius="size" 
  CssClass="string" 
  Display="Static|Dynamic" 
  EnableClientScript="True|False" 
  Font-Bold="True|False" 
  Font-Italic="True|False" 
  Font-Names="string" 
  Font-Overline="True|False" 
  Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large" 
  Font-Strikeout="True|False" 
  Font-Underline="True|False" 
  ForeColor="color name|#ddd|ddd" 
  Height="size" 
  Message="string" 
  Target="string" 
  Text="string" 
  Type="RegEx" 
  ValidationExpression="string" 
  Width="size"
/>

Remarks

When the "type" attribute is set to regex, the control prevents the form from being submitted if its associated control's value does not match the regular expression pattern specified in the "validationexpression" attribute. You associate a control with the <validate> tag by setting its "target" attribute to the ID of the control you wish to validate. The "message" attribute is the text that will be displayed to the user when validation fails. If you are using the <validationsummary> tag, then you can also supply a "text" attribute. When validation fails, the "text" will be displayed where your <validate> tag is and the "message" will be displayed in the <validationsummary>.

Example

<addform>
  ... 
  <table>
    <tr>
      <td>
        <label for="txtPhone" text="Phone"/>
        <textbox id="txtPhone" datafield="Phone" datatype="string"/>
        <validate type="regex" target="txtPhone" validationexpression="^\d{3}\s+\d{3}-\d{4}$" message="The phone number must be in the format: (999) 999-9999"/>
      </td>
    </tr>
    ... 
  </table>
</addform>
The Validate tag whose type is set to "required" is referred to as a Required Field Validator and is used to ensure the user provides some input for a given control.

**Syntax**

```xml
<Validate
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
    BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    Display="Static|Dynamic"
    EnableClientScript="True|False"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    Height="size"
    Message="string"
    Target="string"
    Text="string"
    Type="Required"
    Width="size"
/>```

**Remarks**

The required field validator is one type of the `<validate>` tag. When the "type" attribute is set to **required**, the control prevents the form from being submitted if its associated control does not have a value. You associate a control with the `<validate>` tag by setting its "target" attribute to the ID of the control you wish to validate. The "message" attribute is the text that will be displayed to the user when validation fails. If you are using the `<validationsummary>` tag, then you can also supply a "text" attribute. When validation fails, the "text" will be displayed where your `<validate>` tag is and the "message" will be displayed in the `<validationsummary>`. The "display" attribute determines if the the `<validate>` tag will reserve space for its message in the page layout - typically resulting in blank space in your form - or whether it will dynamically display allocate the space for the message when validation fails. The `<validate>` tag defaults to **Dynamic** display.

**Example**

```xml
<addform>
    <submitcommand commandtext="INSERT INTO Users(FirstName, LastName) VALUES(@FirstName, @LastName)" />
    <table>
        <tr>
            <td>
                <label for="txtFirstName" text="FirstName" />
                <textbox id="txtFirstName" datafield="FirstName" datatype="string" />
                <validate type="required" target="txtFirstName" message="You must enter a First Name" />
            </td>
        </tr>
        <tr>
            <td>
                <label for="txtLastName" text="LastName" />
                <textbox id="txtLastName" datafield="LastName" datatype="string" />
            </td>
        </tr>
    </table>
</addform>
```
The Validate tag whose type is set to "xml" is referred to as an XML Validator and is used to ensure the user provides a well-formed XML snippet as input for a given control. **NOTE:** that this provides very basic verification the XML is well-formed. Deeper checks are not performed.

### Syntax

<Validate
   BackColor="color name|#dddddd"
   BorderColor="color name|#dddddd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
   BorderWidth="size"
   CssClass="string"
   Display="Static|Dynamic"
   EnableClientScript="True|False"
   Font-Bold="True|False"
   Font-Italic="True|False"
   Font-Names="string"
   Font-Overline="True|False"
   Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
   Font-Strikeout="True|False"
   ForeColor="color name|#dddddd"
   Height="size"
   Message="string"
   Target="string"
   Text="string"
   Type="XML"
   Width="size"
/>

### Remarks

The XML validator is one type of the <validate> tag. When the "type" attribute is set to xml, the control prevents the form from being submitted if its associated control does not contain well-formed XML. You associate a control with the <validate> tag by setting its "target" attribute to the ID of the control you wish to validate. The "message" attribute is the text that will be displayed to the user when validation fails. If you are using the <validationsummary> tag, then you can also supply a "text" attribute. When validation fails, the "text" will be displayed where your <validate> tag is and the "message" will be displayed in the <validationsummary>. The "display" attribute determines if the the <validate> tag will reserve space for its message in the page layout - typically resulting in blank space in your form - or whether it will dynamically display allocate the space for the message when validation fails. The <validate> tag defaults to Dynamic display.

### Example

<addform>
   <submitcommand commandtext="INSERT INTO Users(FirstName, LastName, MyXml) VALUES(@FirstName, @LastName, @MyXml)" />
   <table>
     <tr>
       <td>

```html
<tr>
  <td colspan="2">
    <addbutton text="Add"/>
    <cancelbutton text="Cancel"/>
  </td>
</tr>
</table>
</addform>

Back to top

<Validate type="xml"/>

---

**Syntax**

BackColor="color name|#dddddd"
BorderColor="color name|#dddddd"
BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
BorderWidth="size"
CssClass="string"
Display="Static|Dynamic"
EnableClientScript="True|False"
Font-Bold="True|False"
Font-Italic="True|False"
Font-Names="string"
Font-Overline="True|False"
Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
Font-Strikeout="True|False"
ForeColor="color name|#dddddd"
Height="size"
Message="string"
Target="string"
Text="string"
Type="XML"
Width="size"
/

**Remarks**

The XML validator is one type of the <validate> tag. When the "type" attribute is set to xml, the control prevents the form from being submitted if its associated control does not contain well-formed XML. You associate a control with the <validate> tag by setting its "target" attribute to the ID of the control you wish to validate. The "message" attribute is the text that will be displayed to the user when validation fails. If you are using the <validationsummary> tag, then you can also supply a "text" attribute. When validation fails, the "text" will be displayed where your <validate> tag is and the "message" will be displayed in the <validationsummary>. The "display" attribute determines if the the <validate> tag will reserve space for its message in the page layout - typically resulting in blank space in your form - or whether it will dynamically display allocate the space for the message when validation fails. The <validate> tag defaults to Dynamic display.

**Example**

<addform>
   <submitcommand commandtext="INSERT INTO Users(FirstName, LastName, MyXml) VALUES(@FirstName, @LastName, @MyXml)" />
   <table>
     <tr>
       <td>

```html
<tr>
  <td colspan="2">
    <addbutton text="Add"/>
    <cancelbutton text="Cancel"/>
  </td>
</tr>
</table>
</addform>

Back to top
The ValidationSummary tag is used to group the display of all validation messages generated by <validate> tags. The tags’ message will be displayed in the validation summary, while the tags’ text value will be displayed in place of the <validate> tag.

Syntax

<ValidationSummary
    BackColor="color name|#dddddd"
    BorderColor="color name|#dddddd"
   BorderStyle="NotSet|None|Dotted|Dashed|Solid|Double|Groove|Ridge|Inset|Outset"
    BorderWidth="size"
    CssClass="string"
    DisplayMode="List|BulletList|SingleParagraph"
    EnableClientScript="True|False"
    Font-Bold="True|False"
    Font-Italic="True|False"
    Font-Names="string"
    Font-Overline="True|False"
    Font-Size="string|Smaller|Larger|XX-Small|X-Small|Small|Medium|Large|X-Large|XX-Large"
    Font-Strikeout="True|False"
    Font-Underline="True|False"
    ForeColor="color name|#dddddd"
    HeaderText="string"
    Height="size"
    Width="size" />

Remarks

In the example below, a Range Validator has been placed on the form in addition to a Validation Summary. The Range Validator has its message set to "You can only order between 1 and 5 tickets" and has its text set to ". When validation fails, the Range Validator will display ".
next to the text box, while the Validation Summary, below the form's button, will display the error message. If there were additional validation errors, they would also be displayed in the Validation Summary.

**Example**

```xml
<addform>
  ...
  <table>
    <tr>
      <td>
        <label for="txtQuantity" text="Number of Tickets" />
        <textbox id="txtQuantity" datafield="Quantity" datatype="int32" />
        <validate type="range" target="txtQuantity" minimumvalue="1" maximumvalue="5" message="You can only order between 1 and 5 tickets" type="Integer" text="**"/>
      </td>
    </tr>
    <tr colspan="2">
      <addbutton text="Place Order"/> <cancelbutton text="Cancel"/><br />
      <validationsummary displaymode="BulletList" cssclass="NormalRed" headertext="Errors:"/>
    </tr>
  </table>
</addform>
```

Back to top
Tokens

Data Parameter Tokens

Syntax

[[TemplateID_list@ParameterName]]
[[TemplateID_detail@ParameterName]]

Remarks

- **Forming Your Parameter Token**: The token is formed by combining the ID of your `<xmod:template>` tag, some text that identifies if the parameter is from the ListDataSource (_list) or DetailDataSource (_detail), the @ symbol and finally the name of the parameter.

So, if your `<xmod:template>` has an ID of "MyTemplate" and you want to use the value of the parameter "MyParam" in the `<ListDataSource>` tag, your token would look like this:

```
[[MyTemplate_list@MyParam]]
```

All components of the token are required: The template's ID, the "_list" or "_detail", the "@" symbol, and the parameter's name.

- **Tokens Are Not Case-Sensitive**:
  `[[MyTemplate_list@MyParam]]`
  `[[mytemplate_list@MyParam]]`
  `[[mytemplate_list@MyParam]]`

  All of the above are equivalent.

- **Valid in Templates Only**: Currently, use of parameters is limited to the `<ListDataSource>` and `<DetailDataSource>` tags in templates. Within `<xmod:template>` tags, you can use these Data Parameter tokens inside the following tags:
  - `<HeaderTemplate>`
  - `<ItemTemplate>`
  - `<AlternatingItemTemplate>`
  - `<FooterTemplate>`
  - `<DetailTemplate>`
  - `<NoItemsTemplate>`

- **Only Available Within Current Template**: You cannot use parameter tokens to display data from other templates. Likewise you cannot use parameter tokens for the ListDataSource in the DetailTemplate and DetailDataSource parameters cannot be used in the components the list view (Header/Item/AlternatingItem/Footer templates). While no error will be thrown, no data will be returned.

  There is one exception: You can use parameter values from the `<ListDataSource>` and `<DetailDataSource>` in the `<NoItemsTemplate>` tag. The value returned will depend on which data source has no items. So, if you click a button to show the details of the record and the record is not found, `<DetailDataSource>` parameters will be accessible and `<ListDataSource>` parameters will not. The reverse holds true if no records are returned for the `<ListDataSource>`.

Tokens are the mechanism used to display OUTPUT and standard parameter values from your datasources in your templates. At run-time, XMod Pro will replace that token with the value stored in that parameter.

There are some additional usage rules you'll need to adhere to. They are described in the Remarks section.
**Standard Token Usage Rules:**
- Tokens must begin with double-brackets and end with double-brackets (]])
- Tokens can be used as the value of an HTML attribute or in standard text. For example:
  ```html
  <img src="[[Employees_list@PictureUrl]]" align="left" />
  ```
- In many cases, tokens can also be used as the attribute value for an XMod Pro tag. However, when using them in this manner, you MUST delimit the attribute value with single quotes, not double quotes. For example:
  ```html
  CORRECT: <xmod:detailbutton text='[[Employees_list@UserFullName]]' />
  INCORRECT: <xmod:detailbutton text="[[Employees_list@UserFullName]]" />
  ```

**Example**

Stored Procedure in the Database:

```
01: CREATE PROCEDURE GetEmployeesByDepartment
02:   @DepartmentId int,
03:   @DepartmentName nvarchar(100) OUTPUT
04: AS
05: BEGIN
06:   SELECT @DepartmentName = Name FROM Departments WHERE ID = @DepartmentId
07:   SELECT DisplayName, Address1, Address2, City, State, Zip
08:     FROM Employees
09:   WHERE DepartmentId = @DepartmentId
10: END
```

Template Definition:

```
01:<xmod:template id="Employees">
02:  <ListDataSource CommandText="GetEmployeesByDepartment"
                   CommandType="StoredProcedure">
03:    <parameter name="DepartmentId" value="32" />
04:    <parameter name="DepartmentName" direction="Output" datatype="string"
                   size="100" />
05:  </ListDataSource>
06:  <HeaderTemplate>
07:    <h1>Employees in [[Employees_list@DepartmentName]] Department</h1>
08:  </HeaderTemplate>
09:  <ItemTemplate>
10:    <strong>Full Name: [[DisplayName]]</strong><br />
11:    <strong>Address: [[Address1]]</strong><br />
12:      [[Address2]]<br />
13:      [[City]], [[State]] [[Zip]]<br />
14:  </ItemTemplate>
15:  ...
16:  <NoItemsTemplate>
17:    No employees listed in the [[Employees_list@DepartmentName]] department
18: </xmod:template>
```

In the example above, we've created a stored procedure in the database. This stored procedure's main purpose is to return all the employees in a particular department of the company. Additionally, it declares (line #3) and fills an OUTPUT parameter with the name of the department for use by our template (line #6).

In the template definition, we use the name as the stored procedure as the CommandText, and specify "StoredProcedure" as the CommandType. This is a critical step if you are going to use OUTPUT parameters. (Line #2)

Next, in addition to the DepartmentId parameter (line 3), we've defined a second parameter for the OUTPUT parameter (line 4). This must be the same name as that identified in the stored procedure (DepartmentName in our example). You must also define the Direction attribute with a value of Output. It is good practice to also define the DataType attribute with the appropriate datatype that will be returned (String in this
case, which handles all textual values - for an "int" database data type, you might use Int32 or Int64 as the DataType). Finally, since this is textual data, it is required that we specify the Size of the parameter. If not Size is specified, an error will be thrown. It is good practice to match the Size specified in our stored procedure (100), but it is legal to specify a larger or smaller number. If you specify a smaller number and the returned value exceeds that length, the value will be truncated.

Finally, we can use the parameter in our template. In this example, we've placed it in the Header (line 7), to display the department's name above the list of employees. We've also placed it in the <NoItemsTemplate> to provide a more user-friendly message if no records are found (line 17). It would also be legal to place the parameter token in the <ItemTemplate> and <AlternatingItemTemplate> to display with each record as well as the <FooterTemplate>.

**DateAdd Token**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Remarks</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Remarks</td>
<td>Example</td>
</tr>
</tbody>
</table>

The DateAdd token provides you with the ability to generate a date, relative to the current date. This allows you to create a date that is, say, use information about the current module instance at run-time such as the module's ID. Additional Module Tokens will be added to this topic as they become available.

**Syntax**

```
[[DateAdd: number, interval, format]]
```

**Remarks**

- These tokens can be used in templates and forms. Standard token rules apply. See discussion of Field Tokens.

  - **number**: Required if any arguments are specified. This is an integer that specifies how many of `interval` to add to the current date. The number may be positive (greater than 0) to get to a date in the future. A negative number (a number less than zero) can be used to get a date in the past.

  - **interval**: Required if number is specified. One of the following characters to identify which interval to add to the current date:
    - d: Day
    - w: Week
    - m: Month
    - y: Year

  - **format**: Optional. If you would like to determine exactly how the calculated date is returned, you can use standard date formatting expressions here. One example would be `yyyy-MM-dd` to display August 1st, 2012 would display `2012-08-01`. Whereas MM/dd/yyyy will display 08/01/2012 and dd.MM.yy would display 01.08.12.
Example

```xml
<modx:template>
...
<modx:field>

  <h1>This current date is: [[DateAdd:]]</h1>
  <h1>Next week the date will be: [[DateAdd:1,w]]</h1>
  <h1>Last month the date was: [[DateAdd:-1,m]]</h1>
  <h1>In five years the date will be: [[DateAdd:5,y,yyyy-MM-dd]]</h1>

</modx:field>
...
</modx:template>

<ClientName='[[Join("MyForm", [[DateAdd:0,d,yyyyMMdd]])]]'>
</ClientName>
```

Field Tokens

**Syntax**

```
[[fieldName]]
```

**Remarks**

- **Valid in Templates and Form Controls**: Beginning with version 1.3, Field tokens can be used in forms as well as templates. Within form controls, they render values returned in the `<SelectCommand>` in much the same way the field tokens in templates retrieve values from the `<ListDataSource>` or `<DetailDataSource>` commands.

- **Usage In Forms**: Form Field Tokens will retrieve different data depending on the context in which they are used.

  **Initializing the Form**: When a form first loads, it will execute the `<SelectCommand>`.
  Normally this data is bound to your form controls (the Text property of a TextBox, for instance) using the "DataField" attribute. All of the other attributes of the control are hard-coded. Beginning in version 1.3, if you want to use that data to set control properties, you can use the `[[Field]]` token syntax.

  **Form Results**: When a user submits a form, the data they enter is sent to your `<SubmitCommand>` so that it can be saved to the database. You can use `[[Field]]` token syntax to use that data to 1) Customize emails that are sent; 2) Display a custom form-submission acknowledgement (via the `<AddSuccessTemplate>` and `<EditSuccessTemplate>` tags); or 3) Dynamically redirect the user to a custom URL (via the command button tags like AddButton, CancelLink, UpdateImage,
So, while the two token types share the same syntax, they operate on different data depending on the context in which they are used.

Additionally, some control properties may not be set with field tokens. Some properties, particularly color-and some font-related properties, perform internal conversions when they are set. This conversion happens before the controls are bound to the SelectCommand data. In most of these cases, you can still achieve the desired result by using the CssClass property and setting the visual aspects of the control there - or by using the Style property. For example:

```xml
<SelectCommand CommandText="SELECT 'White' AS FColor, 'Blue' AS BColor"/>
<TextBox id="MyText" ForeColor='[[FColor]]' BackColor='[[BColor]]'/>
```

The above example will throw an error. You can change this to use the Style property and Join() function, though, and achieve the same results:

```xml
<TextBox id="MyText" Style='[[Join("color:{0};background-color:{1}", [[FColor]], [[BColor]])]]'/>
```

All we're doing in the above example is creating using the Join() function and form field tokens to do a basic substitution. At run-time, the `{0}` will be replaced with the value of `[[FColor]]` and `{1}` will be replaced with `[[BColor]]`. The result is this:

```
Style="color:White;background-color:Red;"
```

- **Usage In Templates:** For templates, since field tokens retrieve data from each record, they cannot be used in areas of your template that aren't associated with a record. For instance, you cannot use field tokens in the `<HeaderTemplate>`, `<FooterTemplate>`, or `<NoItemsTemplate>` sections. You can use them in `<ItemTemplate>` and `<AlternatingItemTemplate>` sections though.

- **Standard Token Usage Rules:**
  - Tokens must begin with double-brackets `[[ ]]` and end with double- brackets `]]`
  - Tokens can be used as the value of an HTML attribute or in standard text. For example:
    ```xml
    <img src="[[PictureUrl]]" align="left" /> <strong>[[UserFullName]]</strong>
    ```
  - In many cases, tokens can also be used as the attribute value for an XMod Pro tag. However, when using them in this manner, you MUST delimit the attribute value with single quotes, not double quotes. For example:
    ```xml
    CORRECT: <xmod:detailbutton text="[[UserFullName]]" />
    INCORRECT: <xmod:detailbutton text="[[UserFullName]]" />
    ```
Example

Using Field Tokens in a Template

```xml
<modx:template>
...<HeaderTemplate>
  <h1>Users</h1>
</HeaderTemplate>
<ItemTemplate>
  <strong>Full Name: [[DisplayName]]<br />
  <strong>Address: [[Address1]]<br />
  [Address2]<br />
  [City], [State] [Zip]<br />
</ItemTemplate>
...<modx:template>
```

Using Field Tokens in a Form Control

This example gets the amount of inventory (StockOnHand) left for the given product and uses that to limit the maximum quantity that can be ordered using the range validator.

```xml
<modx:template>
<SelectCommand CommandText="SELECT StockOnHand FROM Inventory WHERE ProductId = @ProductId">
  <param name="ProductId" value='[[Url:ProductId]]'/> 
</SelectCommand>
...<label for="txtQuantity" text="Quantity to Order:" />
<textbox id="txtQuantity" datafield="Quantity" datatype="int32" />
<validate type="range" target="txtQuantity" minimumvalue="1" maximumvalue='[[StockOnHand]]' message='[[Join("You must enter a value between 1 and {0}",[[StockOnHand]])]/
...</modx:template>
```

Function Tokens

**Syntax Remarks Example**

Function Tokens provide you the ability to execute certain XMod Pro-supplied functions. Currently there is only one function: Join. As other functions are added, they will be added to this topic.

**Syntax**

```
[[FunctionName(parameter list)]]
```

**Remarks**

- These tokens can be used in templates and forms. Standard token rules apply. See discussion of Field Tokens.

- `[[Join()]]`: Then Join function allows you to merge text with field values and other token values. It can be used in plain text, but it's primary purpose is for use within tag attributes.

  1. The Join function's syntax looks like

    ```
    [[Join(Input, Value1, Value2, ...)]]
    ```

    As with all tokens, it starts with two open brackets `[[`. These are followed by the function name (Join) and an opening parenthesis `. Next comes a list of parameters to the function, separated by commas, and a closing parenthesis `]`. Finally the function and ends with two closing brackets `]]`. The parameters are listed below:
• **Input:** This is a text value. This must be wrapped in double-quotes. It can be standard text or it can include placeholders. **Placeholders** are defined by using a pair of curly braces with a number between them like: `{0}`, `{1}`, `{2}` and so on. Placeholders must start at 0 and increment by 1. At run-time, the placeholders will be replaced with the values that follow (see below).

• **Value1, Value2, ...:** This is a series of 1 or more text values, each separated by commas. At run-time, these values will be used to replace the placeholders found in the **Input** parameter. The first value will replace `{0}`, the second value - if any - will replace `{1}`, the third value will replace `{2}`, and so on. The number of values MUST match the number of placeholders.

2. **Some examples:**

• [[Join("Hello {0}" , "World!")]]
  **Result:** Hello World

• [[Join("Hello, {0}" , [[FirstName]])]]
  **Result:** Hello, John (presuming the value in the FirstName field is "John"

• [[Join("{0} plus {1} equals {2}" , "{1}" , "{2}" , "{3}" )]]
  **Result:** 1 plus 2 equals 3

• [[Join("{1} plus {0} equals {2}" , "{1}" , "{2}" , "{3}" )]]
  **Result:** 2 plus 1 equals 3

• [[Join("alert(""Hello {0}"";return false;" , [[FirstName]])]]
  **Result:** A Javascript alert dialog saying Hello John, presuming the value of in the FirstName field is "John". 

  NOTE: When you need to use double quotes within your Javascript call, escape them by using two double quotes in a row as in this example.

• [[Join("alert(""The Module ID is: {0}";return false;" , [[ModuleName]])]]
  **Result:** A Javascript alert with the text: The Module ID is 257 (the actual module ID will be different based on the actual module ID at run time).

• [[Localize:keyName]]: The Localize token is used to localize static text within your forms and templates. It is used in conjunction with localization resource files - the same types of files used by DNN for localization. The "keyName" refers to an item within the localization file. At run time, the token will be replaced with the value of the item in the localization file. For more information, refer to the [Localizing Text and Content](#) topic.

---

**Example**

```xml
<xm:template>
  ...
  <ItemTemplate>
    [[Localize:Company]] [[CompanyName]] <br />
    [[Localize:Address]] [[Address]]
    <xm:DeleteButton Text="[[Join("Delete {0}" , [[CompanyName]])]]"
      OnClientClick='[[Join("return confirm(""Really Delete {0}?"""";""">"
        
      </xm:DeleteButton>
  ...
</xm:template>
```

---

**Module Tokens**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Remarks</th>
<th>Example</th>
</tr>
</thead>
</table>

Module Tokens provide you with the ability to use information about the current module instance at run-time such as the module’s ID. Additional Module Tokens will be added to this topic as they become available.
Syntax

```
[[Module:ModuleName]]
```

Remarks

- These tokens can be used in templates and forms. Standard token rules apply. See discussion of Field Tokens.
- `[[Module:ID]]`: Returns the numeric ID that uniquely identifies the current module's instance. This ID is assigned by DotNetNuke when the module instance is first added to a page. It is useful when appended to a hard-coded value to help produce a value that is unique to the page.
- `[[Module:TabId]]`: Returns the numeric ID that uniquely identifies the tab (or page) on which the module instance resides. (New in version 1.4)

Example

```xml
<modx:template>

...<HeaderTemplate>
    <h1>This module's ID is: [[Module:ID]]</h1>
</HeaderTemplate>

...</modx:template>

<AddForm ClientName='[[Join("MyForm", [[Module:ID]])]]'>

</AddForm>
```

Portal Tokens

Syntax

```
[[Portal:portalSettingName]]
```

Remarks

- These tokens can be used in templates and forms. Standard token rules apply. See discussion of Field Tokens
- `[[Portal:Alias]]`: Returns the domain for the current portal as defined in the DNN Portal Aliases table. (New in version 1.4)
- `[[Portal:Description]]`: The description assigned to the current portal, as defined in the Portal Settings page. (New in version 1.4)
- `[[Portal:Email]]`: The contact email for this portal.
- `[[Portal:Expiry]]`: The expiration date of the current portal. Returns 12:00:00 AM if no expiration date has been set. (New in version 1.4)
- `[[Portal:HomeTabId]]`: The tab ID for the home page of the portal.
- `[[Portal:ID]]`: Returns the numeric ID that uniquely identifies the current portal's instance. This ID is assigned by DotNetNuke when the portal is created.
- `[[Portal:LoginTabId]]`: The tab in the portal that contains the login form. (New in version 1.4)
- `[[Portal:LogoFile]]`: The name of the image file used for the portal's logo
- **[[Portal:Name]]**: The name of the portal
- **[[Portal:TimeZoneOffset]]**: The offset, in minutes, from GMT for the current portal. This may be a positive or negative number. (New in version 1.4)

### Example

```xml
<modx:template>
...
</modx:template>
```

### Request Tokens

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Remarks</th>
<th>ExampleRequest</th>
</tr>
</thead>
</table>

Tokens provide you with the ability to use information about the current HTTP Request at runtime such as the current URL, the referring URL, etc. Note that some or none of these settings may be available. If they are, the information may not be useful. Internet security/privacy software can block many of these settings and other settings can be forged for benign or malicious reasons. This information is passed through the DotNetNuke filters to remove markup, scripting, and SQL.

**Syntax**

- `[[Request:requestSettingName]]`
- `[[Form:parameterName]]`
- `[[Url:parameterName]]`

**Remarks**

- **Usage**: These tokens can be used in templates and forms. Standard token rules apply. See discussion of Field Tokens.
- `[[Request:Referrer]]`: Returns the URL that sent the user to the current page.
- `[[Request:URL]]`: The URL of the current page
- `[[Request:PageName]]`: (New to version 4.0) The name of the current page, without the file extension.
- `[[Request:HostAddress]]`: The domain name of the user.
- `[[Request:HostName]]`: The domain name of the user.
- `[[Request:Agent]]`: The value of the "user-agent" HTTP header.
- `[[Request:Browser]]`: The text transmitted in the HTTP header to identify the user's browser.
- `[[Request:Locale]]`: The locale ID (i.e. en-US, en-GB, es-MX, es-ES, fr-FR, etc.) for the currently active culture. (added in version 2.1)
- `[[Form:parameterName]]`: If you need to use a Form parameter (also called a POST string parameter), you can use this token. Simply replace parameterName with the name of the parameter passed via the HTTP POST method to the page. Please note that "Form" in this context refers to the name typically given these parameters within HTML and does not refer to XMod Pro forms.
- `[[Url:parameterName]]`: If you need to use a URL parameter (also called a query string parameter), you can use this token. Simply replace parameterName with the name of the parameter in the URL.
- `[[Cookie:cookieName]]`: (new to version 3.0) This token will retrieve the value of a cookie with the name specified after the colon (cookieName in the example). If the cookie does not exist, a empty string is returned.
Example

```xml
<modx:template>
...
<?,template>
<HeaderTemplate>
<h1>HTTP Request Information</h1>
<p>
Referrer: <!--Request:Referrer--><br />
Current URL: <!--Request:URL--><br />
User's IP Address: <!--Request:HostAddress--><br />
Value of URL Parameter 'param1' is <!--Url:param1--><br />
Value of POST Parameter 'FormParam1' is <!--Form:FormParam1--><br />
</p>
</HeaderTemplate>
...</modx:template>
```

User Tokens

User Tokens provide you with the ability to use information about the current user at run-time such as the user's ID, name, email, etc.

Syntax

`[[User:userSettingName]]`

Remarks

- **Usage**: These tokens can be used in templates and forms. Standard token rules apply. See discussion of Field Tokens.
- `[[User:ID]]`: Returns the numeric ID that uniquely identifies the current user. This ID is assigned by DotNetNuke when the user is created.
- `[[User:FirstName]]`: The user's first name
- `[[User:LastName]]`: The user's last name.
- `[[User:DisplayName]]`: The user's display name
- `[[User:Username]]`: The username associated with the user's account.
- `[[User:Email]]`: The user's email address.
- `[[User:profileItemName]]`: An item from the user's profile. Replace profileItemName with the name of the item. For example: `[[User:Telephone]], [[User:Street]], [[User:Cell]], [[User:Fax]], [[User:Region]], etc.`

Example

```xml
<modx:template>
...
<?,template>
<HeaderTemplate>
<h1>[[User:DisplayName]] [[User:ID]]</h1>
<a href="mailto:[[User:Email]]">Send An Email</a>
</modx:template>
```