Windsor Solutions



Advanced Form Design Guide

Updated For Version 4.9

Revision Date: July 6, 2020

**Contents**

[Version Control 3](#_Toc44664074)

[Overview 5](#_Toc44664075)

[Form Level Design 6](#_Toc44664076)

[Document Generation 6](#_Toc44664077)

[Payment Voucher 6](#_Toc44664078)

[Hardcopy Signature Form 6](#_Toc44664079)

[Electronic Signature Authorization Form 6](#_Toc44664080)

[Email Output 7](#_Toc44664081)

[Alternative Identifier 7](#_Toc44664082)

[Fee Calculations 8](#_Toc44664083)

[Internal Data Controls 11](#_Toc44664084)

[Providing a Direct Link to a Form from Another Website 15](#_Toc44664085)

[Data Integration 15](#_Toc44664086)

[Section and Control Design 16](#_Toc44664087)

[Display a Dynamically Assigned Name to a Section Repeater 16](#_Toc44664088)

[Advanced Controls 19](#_Toc44664089)

[Linked Controls 34](#_Toc44664090)

[Custom Validation 38](#_Toc44664091)

[Data Inheritance 41](#_Toc44664092)

[Conditional Display 45](#_Toc44664093)

[Binding Lookup Lists to Form Controls 52](#_Toc44664094)

[Conceal and Purge Data 58](#_Toc44664095)

[Appendix A - Formula Builder Examples 63](#_Toc44664096)

[Appendix B: Lookup Population 66](#_Toc44664097)

[Using a Script to Manually Create nFORM Lookups 66](#_Toc44664098)

# Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date | Notes |
| 4.2 | Windsor Solutions, Alecia Elia | 9/7/2017 | Initial Development. Conditional Display. |
| 4.2 | Windsor Solutions, Christine Galica | 11/17/2017 | Fee Calculations, Custom Validation |
| 4.2 | Windsor Solutions, Alecia Elia | 11/30/2017 | Advanced Tables |
| 4.2 | Windsor Solutions, Christine Galica | 12/22/2017 | Linked Controls, Calculated Controls |
| 4.3 | Windsor Solutions, Alecia Elia | 1/9/2018 | Updated to reflect changes in version 4.3 |
| 4.4 | Windsor Solutions, Alecia Elia | 1/12/2018 | Updated to reflect changes in version 4.4 |
| 4.4 | Windsor Solutions, Alecia Elia | 2/12/2018 | Data Inheritance added |
| 4.4 | Windsor Solutions, Kevin Lyons | 2/27/2018 | Minor updates and additions to content. |
| 4.5 | Windsor Solutions | 7/17/2018 | Updates to reflect v4.5. |
| 4.6 | Windsor Solutions | 2/22/2019 | Updates to reflect v4.6 and slightly refined the name of the document to Advanced Form Design Guide (versus Advanced Design Guide). Added references to the document generation design guide, email output design guide and payment voucher design guide documents. |
| 4.6 | Windsor Solutions | 2/27/2019 | Added content for configuration of Alternative Identifiers and references to other advanced documents. Converted the Dynamic Data Source section to Select Control Data Sources and added content for all data sources. |
| 4.6 | Windsor Solutions | 3/4/2019 | Moved configuration of Internal Data Controls from Design Guide to Advanced Design Guide. Added topic on dynamically assigning names to section repeaters and concealing, purging a data point and exporting/importing form definitions. Also, reorganized the content a bit to separate the form level configuration from the control level configuration. |
| 4.6 | Windsor Solutions | 3/10/2019 | Added topics on configuration of anonymous/guest access and providing direct links to forms. Also added appendix with instructions for populating LOOKUP and LOOKUP\_VALUE tables. Added a tip for large lookup list design. |
| 4.6 | Windsor Solutions | 4/15/2019 | Adjusted formula formatting. |
| 4.6 | Windsor Solutions | 5/10/2019 | Minor formatting adjustments and moved Anonymous/Guest Support and Form Import/Export topics from this document to the standard Form Design Guide. |
| 4.7 | Windsor Solutions | 5/28/2019 | Versioned for v4.7. |
| 4.8 | Windsor Solutions | 2/7/2020 | Versioned for v4.8. |
| 4.8 | Windsor Solutions | 5/20/2020 | Added additional formula builder examples. |
| 4.9 | Windsor Solutions | 7/6/2020 | Updated to reflect v4.9. Changed iif functions references to iff. Added reference to new Hardcopy Signature and Electronic Signature Authorization document generation capabilities. |

# Overview

The nFORM system supports the ability to design custom forms for submission by the regulated community. These forms are highly configurable and the system provides much flexibility for form designers.

Effectively representing an agency’s forms electronically will often result in a much different presentation than on paper. There are many different approaches and techniques for implementing form elements in nFORM. Designing these forms can sometimes be an artistic expression, requiring implementation, trial, feedback and refinement based on real-world use.

As with any system, the more flexible the capabilities of a system, the more sophisticated the system will be to support the dynamic nature of the system. With this in mind, Windsor established this Advanced Form Design Guide to provide information and guidance on the more advanced design topics and advanced form elements.

# Form Level Design

This section describes the advanced design topics at the form level.

## Document Generation

nFORM supports the ability to define a Document Template (template) for staff to generate documents in support of form processing. A template can be designed for the system as a whole as well for specific forms. The content in the template can be static or dynamically driven based on data provided in the submission. This capability allows agencies and form designers to provide a highly formatted document generation capability to its staff.

This functionality is sometimes referred to as “Instant Permit”, because a main use for this new capability will be to generate a permit, based on information provided in the application, after an application has been approved. However, this feature can be used for other official documents such as certificates, licenses, ID cards, etc.

For additional details on this document generation feature as well as the details of how to configure and implement the feature, please reference the Document Generation Design Guide document.

## Payment Voucher

The nFORM system supports the ability to define custom Payment Vouchers for submitters to download in support of the payment remittance process.

A Payment Voucher can be designed for the system as a whole as well for specific forms. The content in the Payment Voucher can be static or dynamically driven based on data provided in the submission. This capability allows agencies and form designers to provide a highly formatted payment voucher to online submitters.

For additional details on this payment voucher customization feature as well as the details of how to configure and implement the feature, please reference the Payment Voucher Design Guide document.

## Hardcopy Signature Form

The nFORM system supports the ability to define custom Hard Copy Signature Forms for submitters to download in support of the form submission signature process.

A Hard Copy Signature Form can be designed for the system as a whole as well for specific forms. The content in the Hard Copy Signature Form can be static or dynamically driven based on data provided in the submission. This capability allows agencies and form designers to provide a highly formatted Hard Copy Signature Form to online submitters.

## Electronic Signature Authorization Form

The nFORM system supports the ability to define custom Electronic Signature Authorization Form for submitters to download in support of the electronic signature authorization process.

An Electronic Signature Authorization Form can be designed for the system as a whole. The content in the Electronic Signature Authorization Form can be static or dynamically driven based on data in the user’s account profile. This capability allows agencies to provide a highly formatted Electronic Signature Authorization Form to online users in order to authorize online signing.

## Email Output

The nFORM system supports the ability to define custom content to be delivered via automated emails sent from the nFORM system.

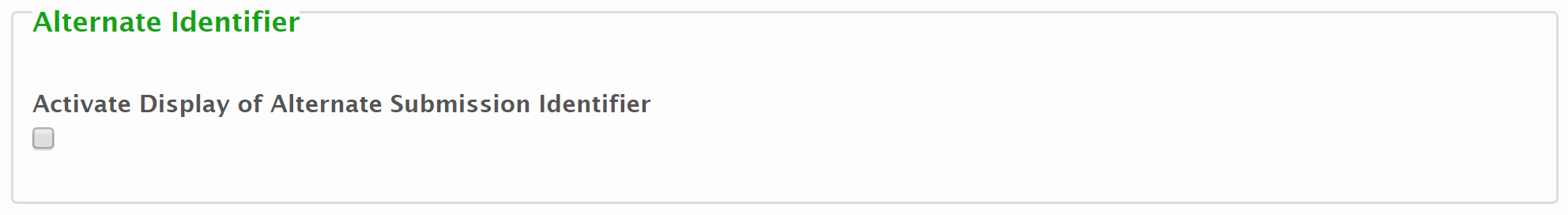
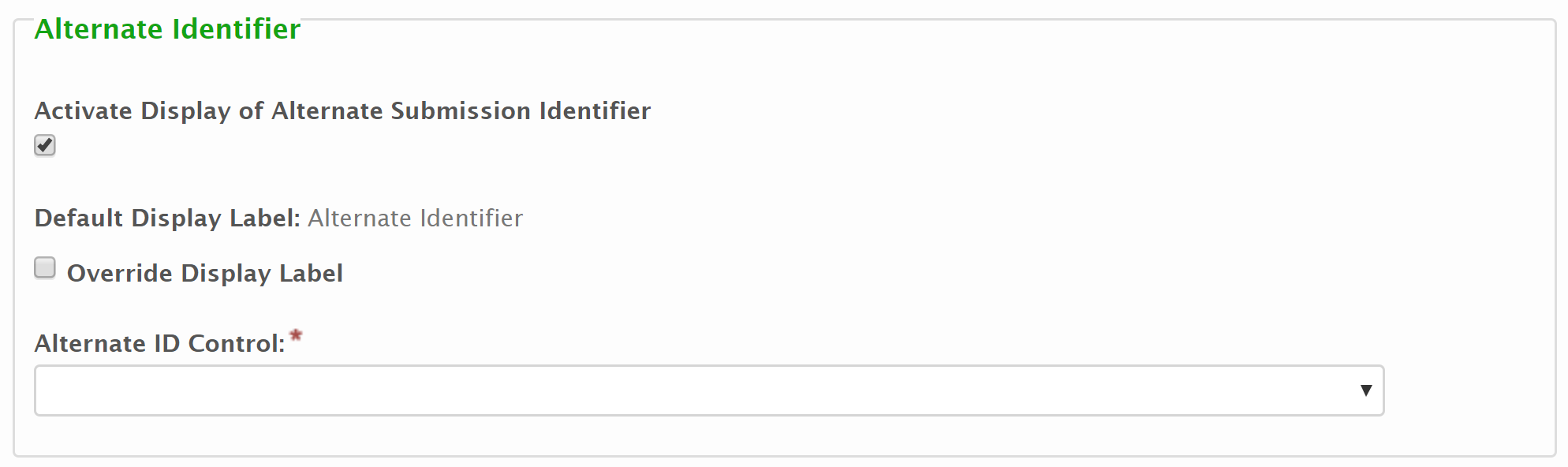
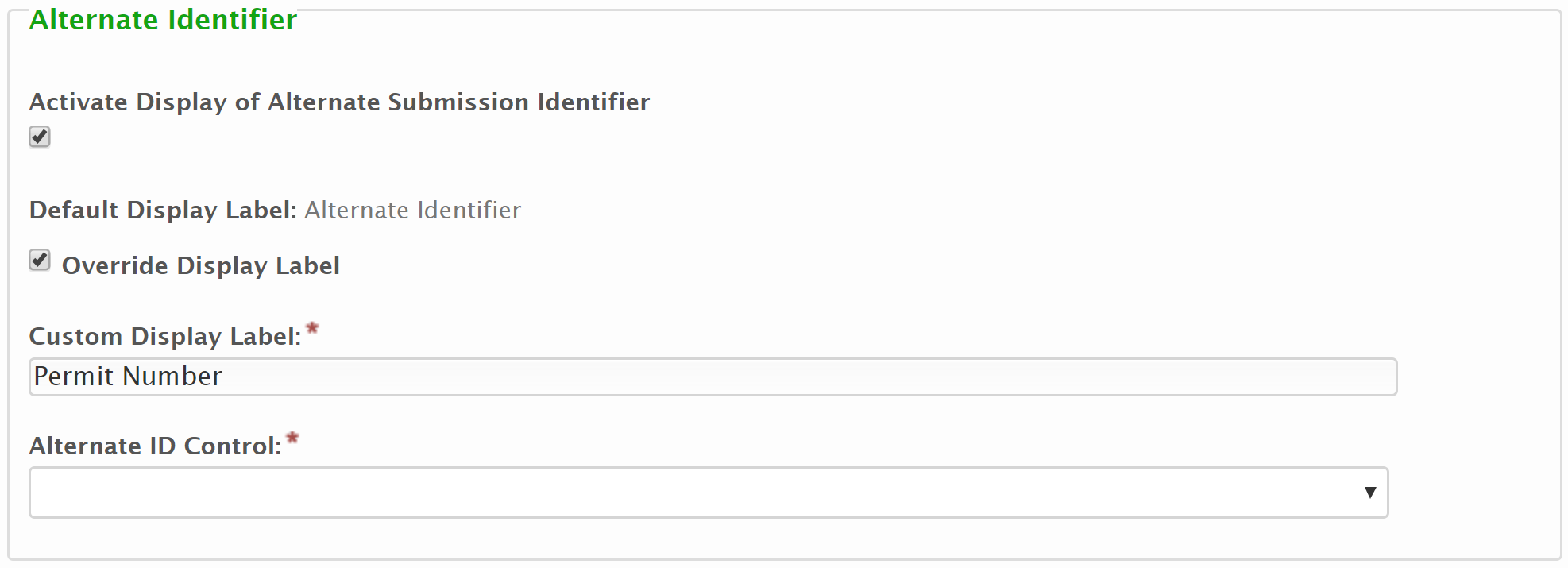
Individual emails can be customized to include custom content. The content in the emails can be static or dynamically driven based on data provided in the submission. This capability allows agencies to provide a highly formatted and targeted emails to online submitters.

For additional details on this email output customization feature as well as the details of how to configure the emails, please reference the Email Template Design Guide document.

## Alternative Identifier

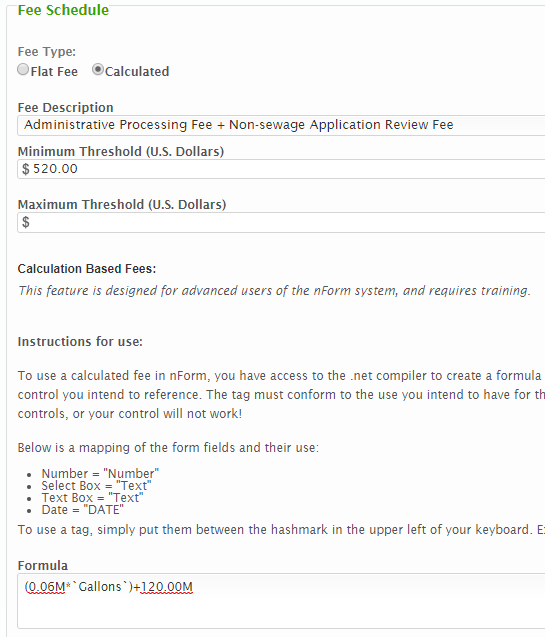
The nFORM system supports the ability to define an alternative identifier for a form submission (e.g., Permit ID, Application ID, etc.). Alternate Identifier allows the form designer to collect data from the submitter, which will then be displayed on the Processing Dashboard, and submission headers as a way to associate a meaningful name that the application can remember and search on.

Use the following steps to configure an alternative identifier:

* Click on the **Forms** link at the top of the page to open the **Forms Manager**.
* Click on the **Edit** icon  for the form in question to open the **Forms Designer**.
* Ensure the current version of the form is in a Draft status to ensure the form configuration can be edited.
* Click on the **Form Details** tab.
* Scroll to the **Alternative Identifier** section of the form.  
  
* Check the **Activate Display of Alternative Submission Identifier** checkbox to enable this feature and display additional relevant settings.  
  
* **Display Label**: By default, “Alternative Identifier” is displayed as the label for references to this value. When utilized, the default label (i.e. "Alternative Identifier") will be changed to another value (e.g., Permit Number, Document Number, etc.).
  + To override this display label to a custom value, check the **Override Display Label** checkbox. This will be enable the **Custom Display Label** field for entry.
  + Enter the desired display label value in the **Custom Display Label** field.   
    
* **Alternative ID Control:** The Alternative ID Control field is used to specify the control that will be used to house and manage the alternative identifier. From this field, select Tag value for the control that be considered the Alternative ID Control on the form.  
  **Note:** Only Short Text, Read-Only and Calculated controls that are assigned a Tag value can be referenced as an Alternative Identifier.

## Fee Calculations

Fees can be assessed to a submitter at the time of form submission. These fees can be a standard flat fee, or a calculated value based on information provided within the submission. These fee calculations can be built directly in the Fee tab of the Form Builder in the Formula field.



Calculated fees allow form designers to define Minimum and Maximum threshold values that will be used if the calculation formula yields a value above or below the defined threshold(s). These threshold values are not required.

**Notes:**

* When a fee is calculated based on submitter input, all form control(s) that contain this input for the calculation must be assigned a Tag so that the fee calculation can refer to the control(s).
* This section describes performing the calculations directly within the Formula area of the Fee configuration. While there are multiple ways to implement fee calculations, Windsor recommends performing computations in a Calculated control (hidden or visible) on the form and then simply referencing that Calculated control form the Formula field of the Fee tab of the Form builder. This technique provides a simplified user interface and formula builder for constructing formulas. In addition, this option allow a form designer to present the fee that will be due, in real time, as the data is entered by the submitter.

The following sections provide a description of the different types of calculation formulas that can be utilized from within the Formula field of the fee configuration settings. The fee formulas utilize the C# arithmetic functions. More complex calculations may be handled on a case by case basis and may require assistance.

### Simple Arithmetic Formula

Numeric controls can be used in the most simple of form fee calculations, and simple arithmetic is sufficient in most cases.

An example of a basic arithmetic calculation would be a simple per unit calculation. If there is a control on the form for the total number of gallons in a tank and we want to base the calculated fee off that total volume multiplied by $0.06, this can be written as:

|  |
| --- |
| `Gallons`\*0.06M |

where the Tag for the control that the user reports the total volume in gallons of their tank is Gallons.

**Considerations:**

* When referencing a control in the form, the control Tag must be enclosed in backticks (`). A backtick (`) a different character from a single quote (‘).
* When referencing a choice within a single or multiple selection control list, a **.description** is required after the closing backtick of the tag. See scenarios below for examples.
* The available mathematical operators include, but are not limited to + (add), - (subtract), / (divide), and \* (multiply). Parentheses can also be used where appropriate.
* An **M** character needs to be appended to the end of constant numbers used in arithmetic formulas to ensure the system casts the number as the correct data type.

### Aggregate Functions

Aggregate functions can be used to calculate aggregations for values in repeatable (tabbed) sections. These functions are appended to the end of the tag name on which the calculation is to be done.

For example, if we are trying to add together numeric values provided from a control in a repeatable form section, such as all the capacities of all the tanks in a repeating tank section we can write the calculation as:

|  |
| --- |
| `Tank\_Capacity`.Sum() |

where the Tag for the tank capacity volume control is Tank\_Capacity.

These aggregate functions can also utilize conditional logic for the aggregate function. This is written by establishing a variable and then comparing the variable to a value.

For example, if we are trying to count the number of all the tanks with capacities greater than 1000 we can write the calculation as:

|  |
| --- |
| `Tank\_Capacity`.Count(c => c > 1000) |

where the Tag for the tank capacity volume control is Tank\_Capacity and c is the variable name chosen to represent capacity. This will count the number of times the `Tank\_Capacity` field has a value greater than 1000.

**Considerations:**

* The variable name is irrelevant to the function as long as it is consistent across the function, however, data type names (e.g., date, string, integer, etc.) should be avoided to ensure the system knows how to handle the variable.
* The application utilize Enumerable methods for these aggregate functions and a more comprehensive list of applicable operators can be found on the Microsoft website. Methods that have been used in nFORM calculated fee formulas in the past include .Count(), .Max(), .Min(), .Sum(), and .Where()

### Conditional If/Then Logic

Conditional operators can be used to support If/Then type logic for conditional logic in complex fee calculations.

The conditional operator **?** returns one of two values depending on the value of a Boolean conditional statement. The syntax for using this conditional logic is **[condition] ? [first\_expression] : [second\_expression]** with the question mark following the condition and the colon separating the two different expressions. The first expression is evaluated if the condition is True, the second is evaluated if the condition is False.

For example, if there is a calculated fee requirement that the fee be $0 if the submitter has a fee status of Exempt, but $125 if the fee status is anything else, this can be written as:

|  |
| --- |
| `Fee\_Status` == “Exempt” ? 0.0 : 125.00 |

where the Tag for the fee status control is Fee\_Status.

**Considerations:**

* Constant string values should be enclosed in quotation marks to ensure the system casts the value as the string data type.
* Other Boolean operators can also be used in place of the == (equal to) operator, such as != (not equal), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), etc

While this list of complex calculation types is not extensive, it covers those types of calculations which have been built in nFORM in the past and allows for a wide variety of complex fee calculations to be created that suit most form requirements.

## Internal Data Controls

**Note:** This section does not apply to users using nFORM through an nVIRO integration.

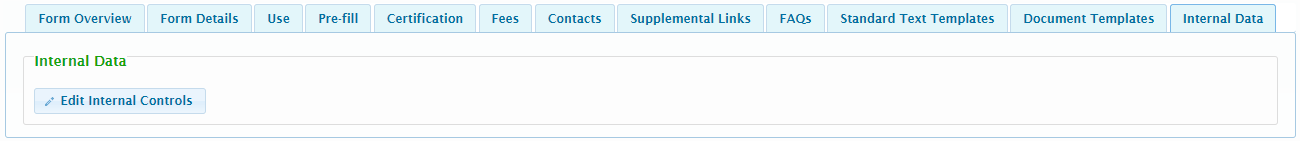
Forms can be configured with internal-only data controls to allow additional internal only data points to be added to a form to support the submission process. These attributes are accessible, viewable, and editable by internal users of the system only. The attributes are initially configured for a form during form design and are available for data entry during the submission processing phase. Additional attributes can also be added during submission processing, if needed.

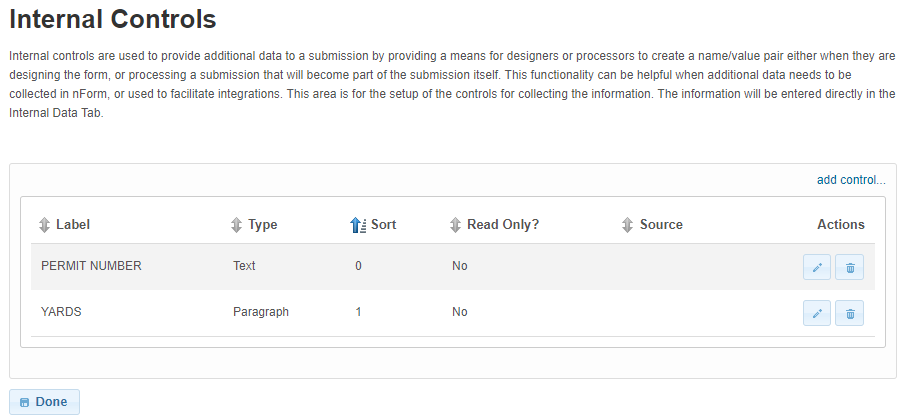
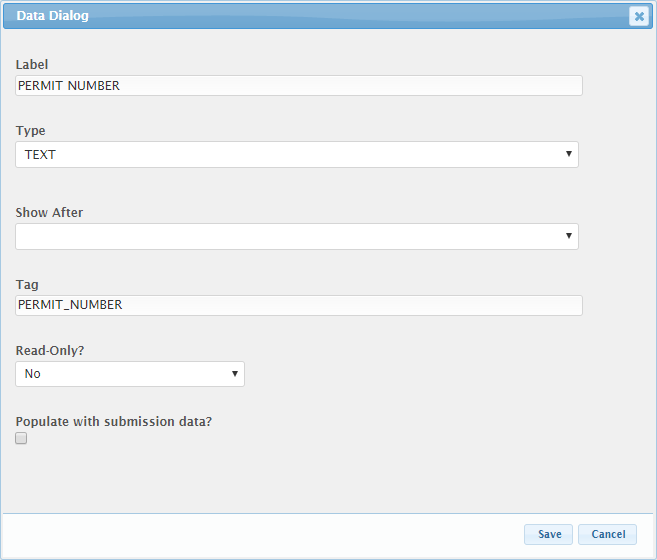
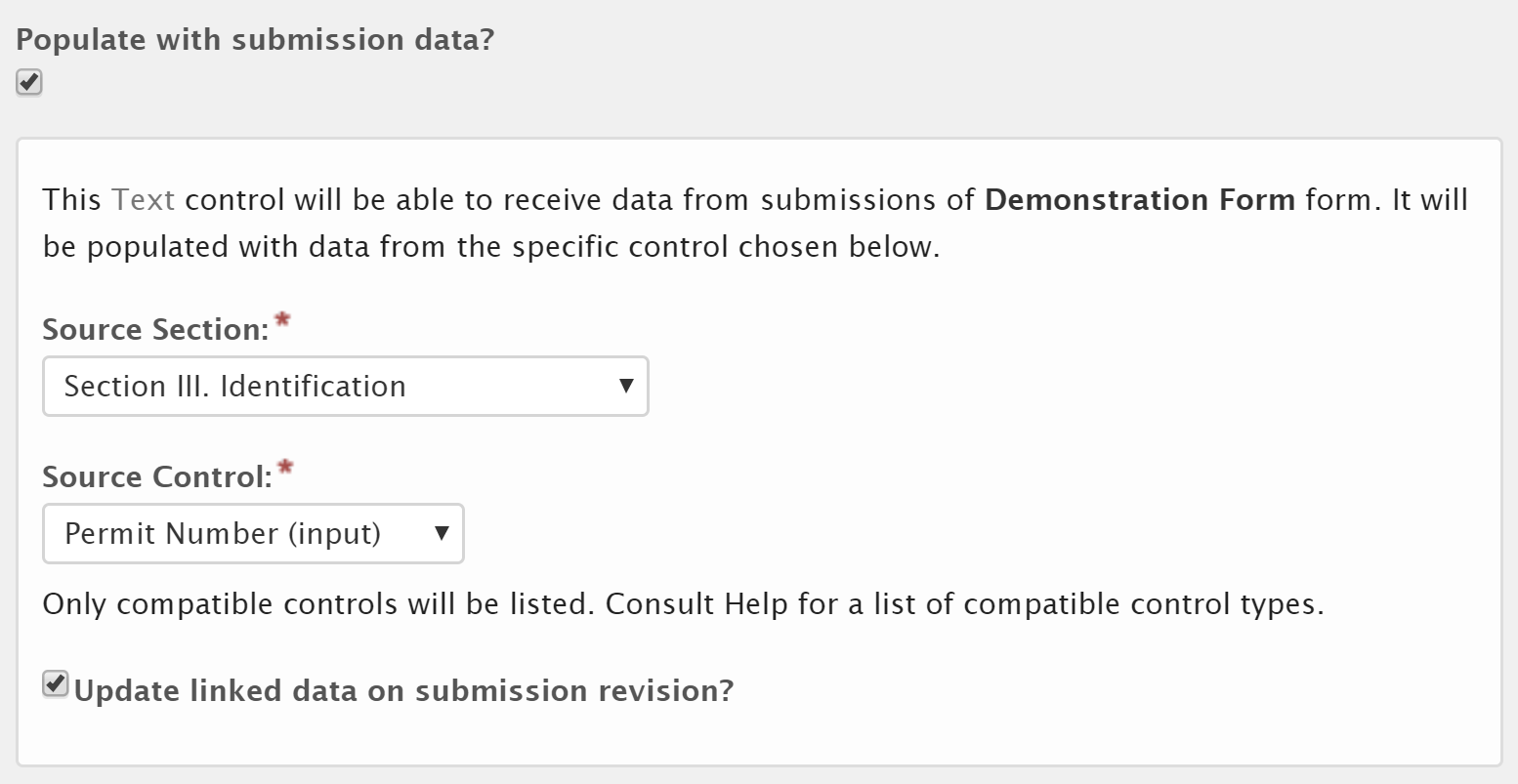
This section of the document will describe how to configure the default Internal Data Controls for a form, how to add/adjust Internal Data Controls for a submission and how to use Internal Data Controls on a submission.

### Configure Default Internal Data Controls for a Form

This section of the document describes the setup/configuration of the default set of internal data controls (for collecting additional information, by internal staff) for the form.

To configure the default Internal Data Controls for a form, follow these steps:

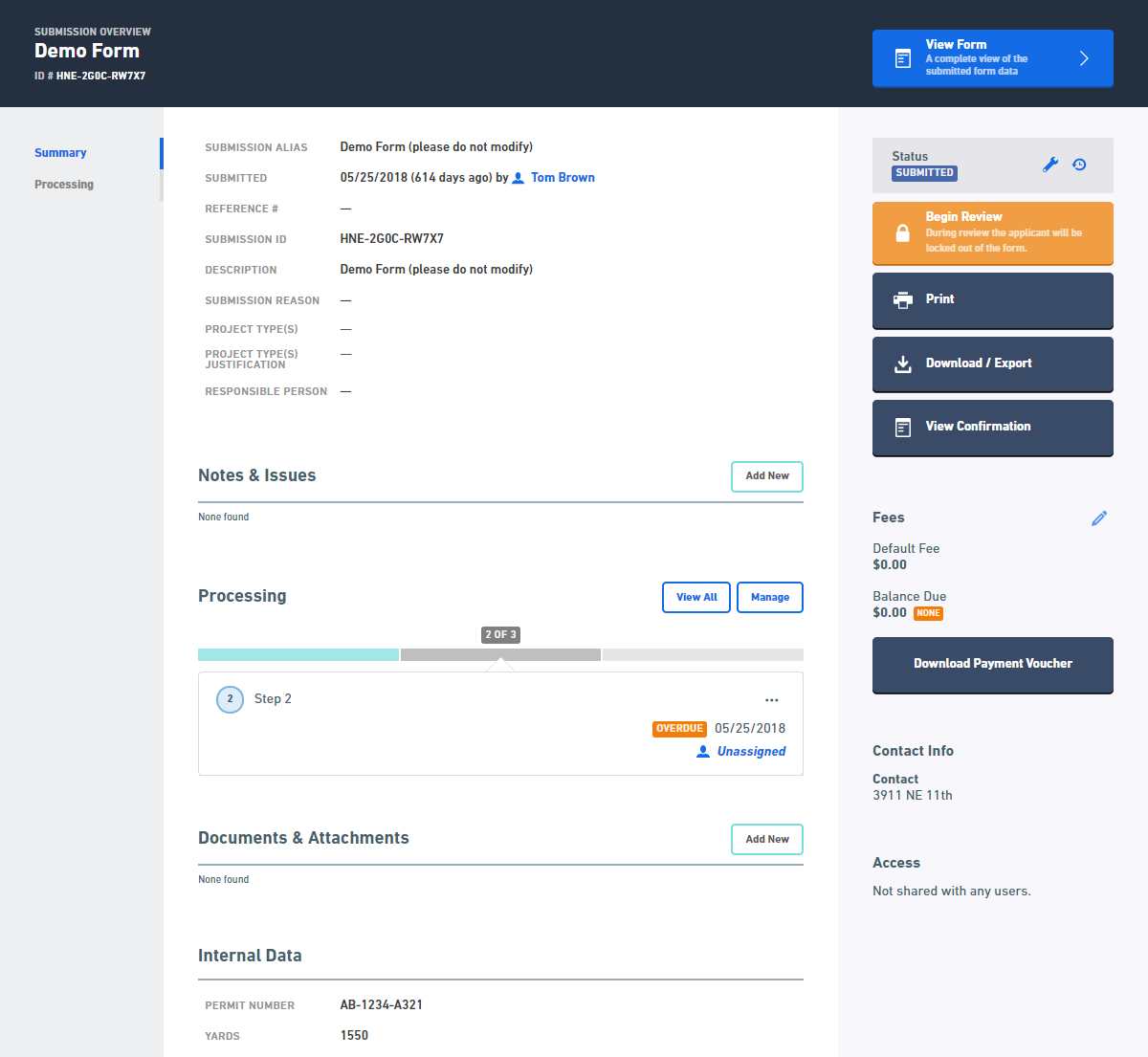


* Click on the **Forms** link at the top of the page to open the **Forms Manager**.
* Click on the **Edit** icon  for the form in question to open the **Forms Designer**.
* Ensure the current version of the form is in a Draft status to ensure the form configuration can be edited.
* Navigate to the **Internal Data** tab in the Form Details step of the Form Designer.
* Click the **Edit Internal Controls** button to open the list of internal data controls that have been configured.  
  
* Click the **Add Control** link to add a new internal data control. To edit an existing internal data control, click the **Edit** button to the right of the control in question.
* **Add/Edit Internal Data Controls** – When adding/editing an Internal Control, the following attributes are available:  
  
  + A **Label** will describe the internal control. This will be the value that is presented to a user when working with internal controls.
  + The **Type** specifies the type of internal control. Internal Controls can be of the following Types:
    - Checkbox
    - Date
    - Dropdown
    - Paragraph
    - Text
    - Time
  + The **Show After** value is used to specify a custom sort order for the list of values.
  + The **Tag** value is a unique name assigned to an internal control to allow the control to be identified. This attribute is essential for facilitating the data importing and exporting processes.
  + The **Read-Only** attribute is used to make a internal data control read-only and locked. This is useful when populating the internal data control value based on a value entered on the submission.
  + The **Populate with Submission Data** is used to configure an internal data control to be populated based on a value from the submission. If checked, the **Source Section**, **Source Control**, and **Update linked data on submission revision** fields will be displayed. These values will be used to link the internal data control to a field within the submission.  
    
  + The **Source Section** field is used to select and identify the section which houses the field that will be copied to the internal data control field.
  + The **Source Control** field is used to select identify the control (within the selected **Source Section**) which represents the field that will be copied to the internal data control field.
  + Check the **Update linked data on submission revision** field to configure the internal data control to be updated to the latest value following each submission. If this value should remain untouched following the initial submission (and any changes by internal staff), leave this attribute unchecked.
  + Click **Save** button to save internal data control.
* Repeat the above steps for each desired Internal Data Control.

### Populate Internal Data Controls for a Submission

This section of the document describes how Internal Data Controls can be populated for a submission.

To populate the Internal Data Controls for a specific submission, follow these steps:

* Open the **Submission View** page for the submission in question from the **Processing Dashboard**.  
   
* Hover over the Internal Data attribute and click on the field. The attribute will now be editable.  
   
* Enter the appropriate values in this field.
* Click out of the field to save the entered values.

## Providing a Direct Link to a Form from Another Website

A link to the form can be added to a public website to allow users to navigate directly to the nFORM submission wizard. For example, if an agency has a web page specific to the Title V permitting process, a direct link to the Title V permit application can be presented. The URL syntax for a form link is as follows:

|  |
| --- |
| **https://**[baseNformUrl]**/?FormTag=**[formTagName]**&skipLandingPage** |

For example:

|  |
| --- |
| <https://nsuite.windsorsolutions.biz/demo/nsuite/nform/?FormTag=FOIA_FORM&skipLandingPage> |

Note: forms must be published, externally available (i.e., Form for Internal Use Only attribute is unchecked) and a Form Tag value must be assigned to enable this feature to work.

<https://nformtest.windsorsolutions.biz/TDEC/?FormTag=CGP_NOI&prepopulateContext=TNR154897&skipLanding>Page

### Configuring Submission Exit Behavior

After a form has been submitted, the user is presented with a "Return to..." button. This button can be configured to take the user back to an unsecured page (outside of the system). Typically, the page that hosted the link that originally directed them to the anonymous form. Configure the return URL setting in the **varEnvironmentOverrides.config** files app.submission.confirmation.buttonUrl setting.

## Data Integration

The nFORM system supports the ability to import (e.g., Auto-Fill and Pre-Fill), export and/or change data in the system based on external sources/destinations. See the Data Integration section of the *Integration Guide* for additional details on these features.

# Section and Control Design

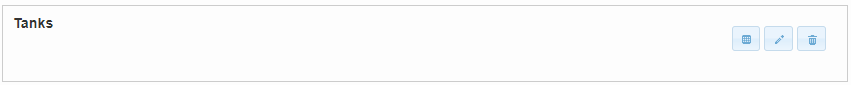
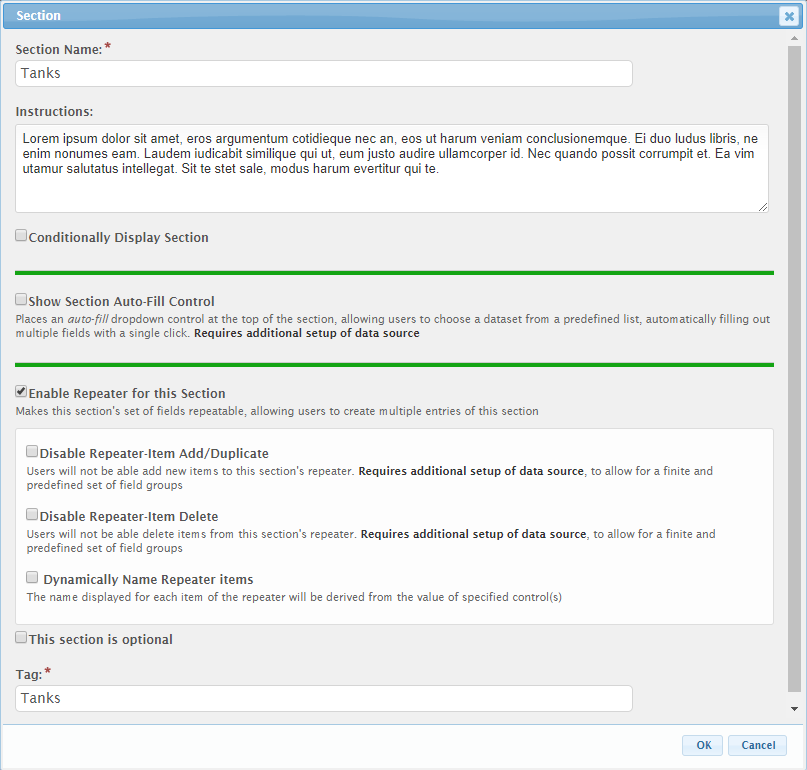
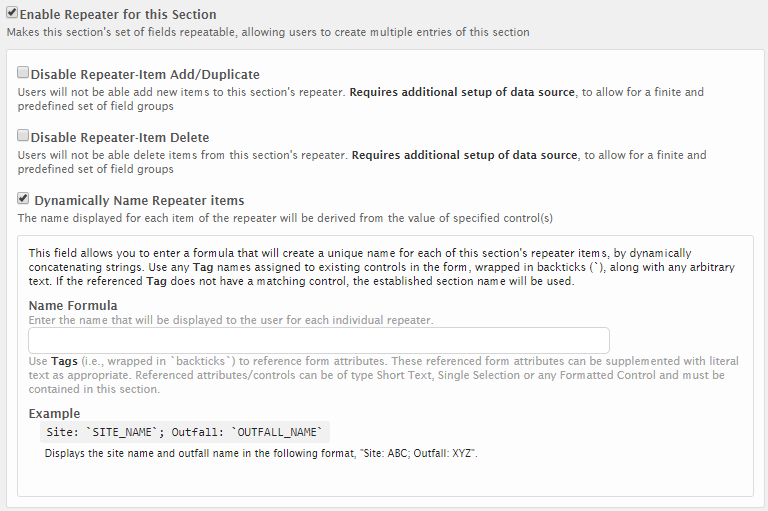
This section describes the advanced design topics at the section and control level.

## Display a Dynamically Assigned Name to a Section Repeater

The system supports the ability for a section to be repeated. For example, the form may have a section describing the numerous Tanks in a process. The system supports the ability for the same set of questions to each answered for each tank, and for the details of these tanks to be contained in the same section.

In addition, the system supports the ability to display a dynamically assigned name to each repeating items in a section based on the content held in the section repeater. For example, using the Tanks example previously used, each tank may have a Tank Name and the item in the repeater can display the Tank Name in the title.

To assign a dynamically assigned name to a repeating section, perform the following:

* Click on the **Forms** link at the top of the page to open the **Forms Manager**.
* Click on the **Edit** icon  for the form in question to open the **Forms Designer**.
* Ensure the current version of the form is in a Draft status to ensure the form configuration can be edited.
* Navigate to the **Sections and Controls** (Step 2) section of the Form Designer.
* Click on the **Edit** icon for the section in question.  
  
* From the **Section** dialog, click the **Enable Repeater for this Section** checkbox to enable the repeater configuration settings.  
  
* Click the **Dynamically Name Repeater Items** checkbox to enable the dynamic name configuration settings.  
  
* Enter the formula for deriving the repeating section name in the **Repeater-Item Name Formula**field. This formula field allows for specifying the text to display for each item in the list. For example:

|  |
| --- |
| **`TANK\_NAME`** |

* A great deal of flexibility has been provided with this formula which supports very simple and very complex derivations. See the Formula Builder Examples section of this document for details on format of formulas used in this area.

## Advanced Controls

Controls types are classified into one of three types: Simple, Formatted and Advanced. The advanced controls include:

* Name
* Location
* Address
* Attachment
* Table
* Advanced Table
* Contact
* Hidden Text
* Calculated

These controls are specifically designed to serve key tasks with specific capabilities, configurations, and looks.

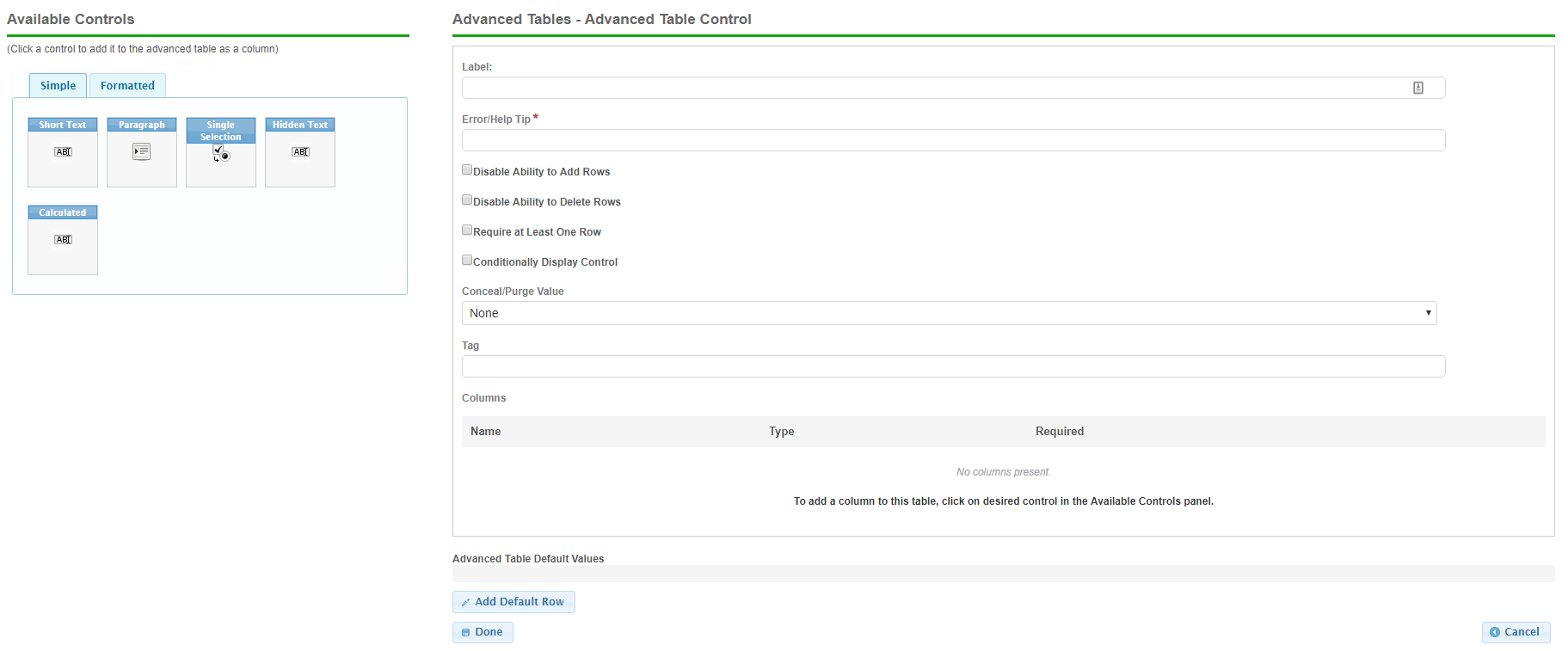
### Advanced Table Control

The Advanced Table control provides a means for presenting a data entry table to a user within a form.

The Advanced Table provides the ability to utilize many of the existing control types within a dynamic table. Form designers have the ability to configure the control type utilized in each column of a table. A user can add as many rows as needed to a table or a developer can configure a set number of rows. This flexibility allows for a wide variety of combinations and scenarios to support an agency’s needs.

#### Form Design

When an Advanced Table control is added to a section of a form, the Advanced Table control editor is presented.



Below is a description of each attribute available for the Advanced Table control.

**Label:** The label text will display as the header of the table.

**Error/Help Tip:** The text to display when the user hovers over the table or when an error is found in the table.

**Disable Ability to Add Rows:** Selecting this option will disable the “add rows” option for the table from the form. At least one default row must be added during configuration to enable the table for use.

**Disable Ability to Delete Rows:** Selecting this option removes the “delete row” option from the form. This prevents a user from removing a necessary row.

**Require at Least One Row:** Selecting this option requires the user to complete at least one row of the table. This is similar to marking other controls “required” but limits that requirement to a single row of the table.

**Conditionally Display Control:** Advanced Tables can be conditionally displayed just like any other control.

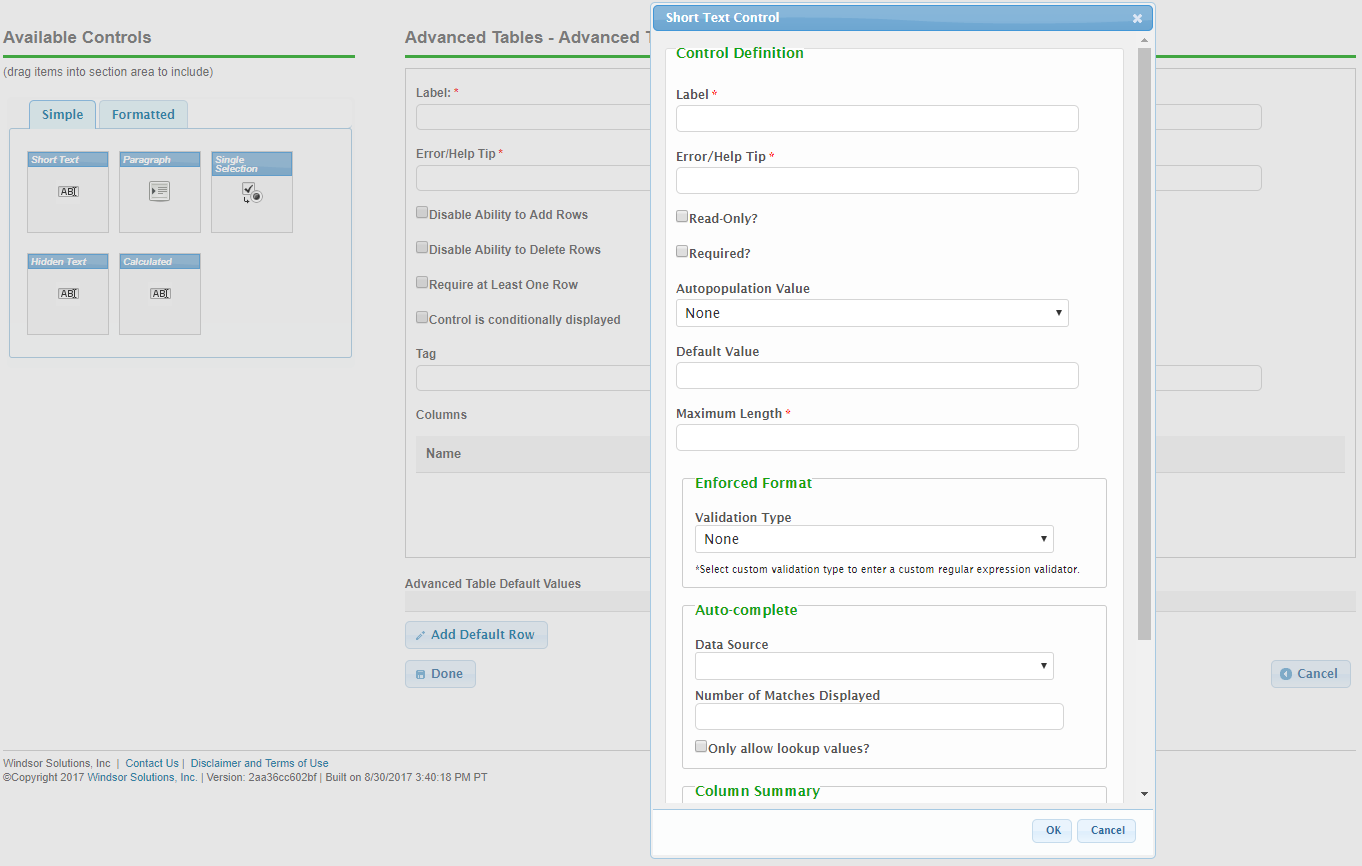
**Conceal/Purge Value:** For sensitive information, control value selection can set the conceal, conceal and purge, or none. If "None" is selected, data will display in standard ways. If "Conceal" is selected, data entered in this control will be concealed from most users. If "Conceal and Purge" is selected, data entered in this control will be concealed from most users and will be purged from the submission, once the submission is finalized. See the Conceal and Purge section of this document for additional details on this feature.

**Tag:** The unique name assigned to the control to allow the table control to be uniquely identified. This attribute is essential for facilitating the data importing and exporting processes. This “Tag” field assigns a Tag value to the table. Tags are assigned to the table as well as each column/control contained within the table allowing reference to the table and/or an individual column in the table.

**Columns:** This area will define the columns presented in the table.

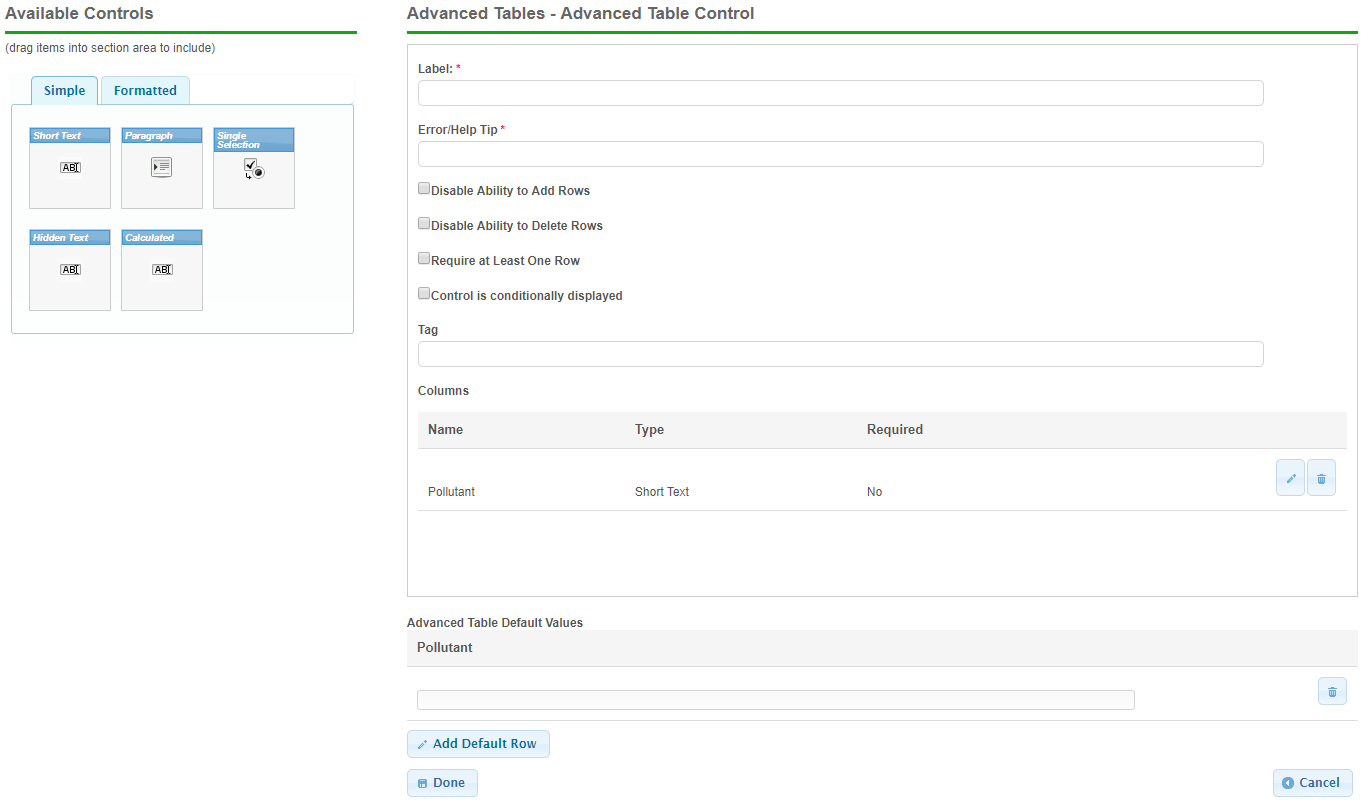
Controls are selected from the “Available Controls” box and added to the “Columns” area of the editor. Once a control is added to the columns section, the edit window for that control type appears. The control is then configured in the same way that it would be outside of an advanced table – with the exception of some control attributes being unavailable within a table. The display order of controls within the table is determined by the sequence in which they are listed in the column section. The first control listed in the columns section will be the far left or first column. The control at the bottom of the list will be the far right or last column in the table.

**Note:** Not all controls are available for advanced table configuration. For example, a contact control cannot be placed within a table due to space limitations. Likewise, an advanced table cannot be placed within another advanced table.



**Advanced Table Default Values**

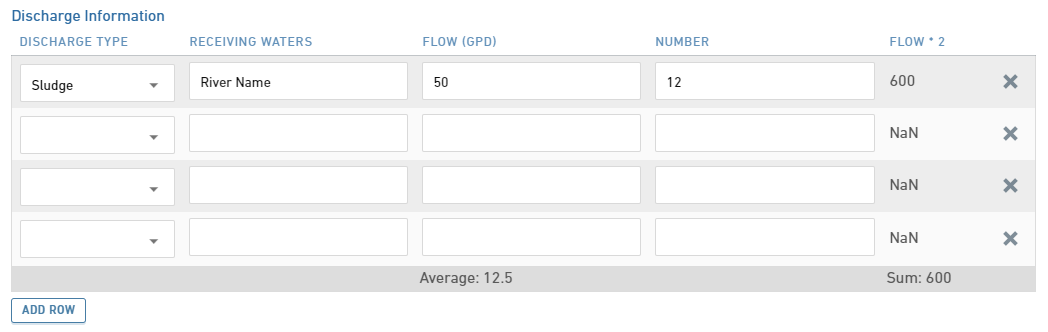
Default row(s) can be added once there is at least one control in the columns. Default rows display in the “Advanced Table Default Values” section of the editor. These rows will display in the form, defaulting the column values as selected/entered.



**NOTE:** If a control within a table has been marked “required” then a form cannot be submitted until all rows of that table have entered values for that column.

#### Submission Wizard Presentation

Below is an example of how an advanced table may appear on a form. Advanced tables are highly flexible and allow for a variety of control options, quantity of columns and rows, and control types. The final look of the table is dependent on this configuration selected.



### Contact Control

The Contact control allows for capture of detailed information for a contact person, including attributes such as a name, title, company, phone numbers, email, and address, etc.

#### Form Design

When a Contact control is added to a section of a form, the control editor is presented.

Below is a description of each attribute available for the Contact control.

**Label:** The label for the control will be displayed to the submitter.

**Error/Help Tip:** The help tip that will be presented to a submitter when hovering their mouse over the help tip icon (http://qaweb.windsor.com/nForm/Content/help/DefaultAgencyStaffHelp/img/HelpTipIcon.jpg) for the control.

**Data Entry:** The data entry area for the control. Note: if Read-Only attribute is enabled, the fields will be locked and data entry will not be allowed.

**Contact**

**Prefix:** The prefix for the contacts name (e.g., Dr., Ms., etc.).

**First Name:** The first name for the contact.

**Last Name:** The last name for the contact.

**Title:** The title of the contact.

**Company Name:** The company which the contact represents.

**Email:** The email address for the contact.

**Phone:** The phone number for the contact.

**Ext.:** The extension for the phone number for the contact.

**Fax:** The fax number for the contact.

**Address**

**Address Line 1:** The street address line for the address.

**Address Line 2:** The secondary address line for the address (e.g., BLDG #, APT #, STE #).

**Location Description:** A supplementary description for the address (e.g., 1/4 mile past mile post 214).

**City:** The city for the address.

**State/Area:** The state for the address.

**County:** The county for the address, if applicable.

**Postal Code:** The postal code for the address.

**Country**: The country for the address.

**Validate Address:** When clicked, the address entered will be verified against a domestic address service. If an improved address match is found, it will be presented for selection.

**Control Display:** Control may not display depending on Display Formula value configuration.

#### Control Properties in Form Design

The following configuration properties are available for the control:

**Header:** The label that will be presented to the submitter in the header of the control.

**Error/Help Tip:** The help tip that will be presented to a submitter when hovering their mouse over the help tip icon (http://qaweb.windsor.com/nForm/Content/help/DefaultAgencyStaffHelp/img/HelpTipIcon.jpg) for the control. This attribute will also be utilized to provide additional information to the submitter when an invalid value is available for the control.

**Read-Only?:** If set to "Yes", the control will be displayed in a read-only mode. If set to "No", the control will be editable. If set to "If Prepopulated", the control will be displayed in read-only mode, if the control is pre-filled with data.

**Autopopulate Contact and Address?:** If checked, the contact and address fields will automatically be populated in the control based the information found in the submitters users profile.

**Conditionally Display Control?:** If checked, the Display Formula will be enabled for configuration.

**Display Formula:** The formula used to determine if this control will be displayed in the form. See Formula Builder (Section 11.6) topic for additional details on use of this field.

**Conceal/Purge Value:** For sensitive information, control value selection can be set the conceal, conceal and purge, or none. If "None" is selected, data will display in standard ways. If "Conceal" is selected, data entered in this control will be concealed from most users. If "Conceal and Purge" is selected, data entered in this control will be concealed from most users and will be purged from the submission, once the submission is finalized.

**Tag:** The unique name assigned to an the control to allow the control to be uniquely identified. This attribute is essential for facilitating the data importing and exporting processes.

**Components:**

**Main Contact Info:**

**Contact Group Label:** The label displayed above the Contact information area in the control. If left blank, no label will displayed in this area.

**Individual:**

**Display Individual Name:** If checked, the individual First Name and Last Name fields will be displayed.

**Individual Name Required:** If checked, the individual First Name and Last Name fields will be required to be entered by the submitter.

**Prefix:** If checked, the Prefix field will be displayed.

**Prefix Required:** If checked, the Prefix field will be required to be entered by the submitter.

**Middle Name:** If checked, the Middle Name field will be displayed.

**Middle Name Required:** If checked, the Middle Name field will be required to be entered by the submitter.

**Title:**

**Display Title**: If checked, the Title field will be displayed.

**Title Required:** If checked, the Title field will be required to be entered by the submitter.

**Organization/Company Name:**

**Display Organization/Company Name:** If checked, the Organization/Company Name field will be displayed.

**Organization/Company Name Required:** If checked, the Organization/Company Name field will be required to be entered by the submitter.

**Organization/Company Name Custom Label: I**f a value is entered in this field, this will be the label presented for the Organization/Company Name field. If no value is provided, Organization Name will be displayed.

**Phone:**

**Display Phone:** If checked, the Phone fields will be displayed.

**At Least 1 Required:** If checked, at least one phone number will be required to be provided by the submitter.

**Phone Number Group Label:** The label displayed above the Phones area in the control. If left blank, no label will displayed in this area.

**Multiple phone numbers allowed:** If checked, multiple phone number can be provided by the submitter.

**Allowed phone number types:**

**Main:** If checked, the user will be allowed to enter a main phone number.

**Business:** If checked, the user will be allowed to enter a business phone number.

**Home:** If checked, the user will be allowed to enter a home phone number.

**Mobile:** If checked, the user will be allowed to enter a mobile phone number.

**Other:** If checked, the user will be allowed to enter an other type of phone number.

**Note:** Selecting more than one phone type changes the appearance of the control and position of the phone number data entry fields.

**Fax:**

**Display Fax:** If checked, the Fax field will be displayed.

**Fax Required:** If checked, the Fax field will be required to be entered by the submitter.

**Email Address:**

**Display Email Address:** If checked, the Email field will be displayed.

**Email Address Required:** If checked, the Email field will be required to be entered by the submitter.

**Address:**

**Display Address:** If checked, the address fields (e.g., Address Line 1, Address Line 2, City, State/Area, Postal Code) will be displayed.

**Address Required:** If checked, an address will be required to be entered by the submitter.

**Address Group Label:** The label displayed above the address information area in the control. If left blank, no label will displayed in this area.

**Location Description:** If checked, the Location Description field will be enabled, allowing for entry of a location description (e.g., 1/4 beyond mile post 214)

**Country:** If checked, the Country field will be enabled.

**Country Required:** If checked, the Country field will be required.

**County:** If checked, the County field will be enabled.

**County Required:** If checked, the County field will be required.

**Require County if Country is provided (US Only):** If checked, the County field will be required if the Country field is populated (and equal to United States).

**Validation:**

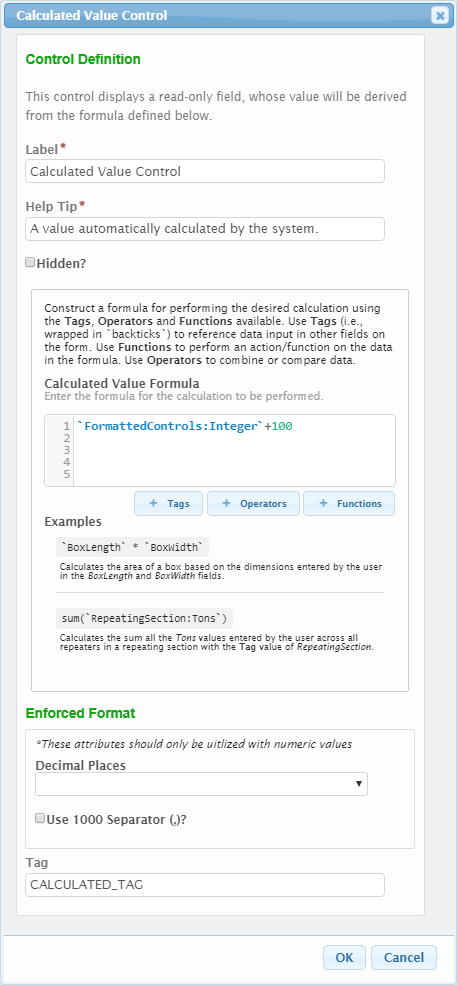
**Allow partial addresses:** If checked, partial addresses will be accepted (e.g., city, state and zip code with no street address are accepted, etc.).

### Calculated Control

The Calculated Control is an advanced read only control type that can be used to display the result of a calculation as a read only field on a form. Returned values can evaluate to a string, Boolean or numeric value. To add a calculated control to a form section, the calculated control can be found on the Advanced tab when selecting from available controls.

#### Form Design

When a Calculated control is added to a section of a form, the control editor is presented.



Below is a description of each attribute available for the Calculated control.

**Label:** The label that will be presented to the submitter for the control.

**Help Tip:** The help tip that will be presented to an submitter when hovering their mouse over the help tip icon (http://qaweb.windsor.com/nForm/Content/help/DefaultAgencyStaffHelp/img/HelpTipIcon.jpg) for the control. This attribute will also be utilized to provide additional information to the submitter when an invalid value is available for the control.

**Hidden?:** If checked, the system will hide the display of this control from the submitter. The calculated control can be hidden if the intent is to use the control for exporting calculated data only or as the dynamic display name for a repeating section.

**Calculated Value Formula:**  The formula used to determine the value(s) that will be displayed in the form.

**Note:** The user can utilize the buttons for Tags, Operators, and Functions to develop their formula. Alternatively, the formula can be hand keyed into the Display Formula textbox.

**Enforced Format:**

* **Decimal Places**: Specifies the number of decimal places with will be enterable/displayed for the value.
* **Use 1000 Separator (,)**?: If checked, a comma will be displayed to separate 1000’s (e.g., 1,343,200).

**Tag:** The unique name assigned to an the control to allow the control to be uniquely identified. This attribute is essential for facilitating the data importing and exporting processes.

#### Tags

Tags in the Display Formula window are the tags for controls within the form that are available for reference. When a tag is referenced in the formula it must be surrounded by backticks (` `). Tags added to the questions from the Tags button will already have these backticks.

A control within the same section as the one configured to conditionally display will be referenced as simply the control tag name in the syntax of `CONTROL\_TAG`.

|  |
| --- |
| `SITE\_CNTY` |

A control in a different section than the one configured to conditionally displayed will be referenced as `SECTION\_TAG:CONTROL\_TAG`.

|  |
| --- |
| `SITE:SITE\_CNTY` |

Controls available for reference will display when the ‘TAGS’ button is selected.

Notes:

* When referencing a control in the form, the control Tag must be enclosed in backticks (`). A backtick (`) a different character from a single quote (‘).
* Advanced table controls cannot be referenced in a conditional display formula as of nFORM version 4.2.
* When referencing a choice within a single or multiple selection control list, a **.description** is required after the closing backtick of the tag. See scenarios below for examples.

#### Operators

Operators indicate how the formula will evaluate the value within a control and the criteria that control is evaluated against. Available operators include, but are not limited to:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | 'And' |
| || | 'Or' |
| >= | 'Greater Than Or Equal To' |
| > | 'Greater Than' |
| <= | 'Less Than Or Equal To' |
| < | 'Less Than' |
| == | 'Equal' |
| != | 'Not Equal' |
| + | Add numbers or join strings |
| - | Subtract numbers or negate number |
| \* | Multiple numbers |
| / | Divide numbers |
| ? | Ternary operator (condition ? itTrue: ifFalse) |
| ^ | Power operator (x ^ y) |
| ! | Boolean negation operator |

Note: Some values may be returned as a string concatenation. In these cases, the value may need to be converted to the desired format using num(), str() or val() functions.

#### Functions

The Functions button lists predefined procedures that have been configured into nFORM for use in evaluating tag values. These functions include, but are not limited to:

|  |  |
| --- | --- |
| **Function** | **Description** |
| sum(x, y, z, ..., n) | Adds all members of an array using "+". x, y, z and n represent individual values. |
| avg(x, y, z, ..., n) | Averages all members of an array using "+" and "/", returns 0 on empty array. x, y, z and n represent individual values. |
| max(x, y, z, ..., n) | Returns the number with the highest value. x, y, z and n represent individual values. |
| min(x, y, z, ..., n) | Returns the number with the lowest value. x, y, z and n represent individual values. |
| abs(x) | Returns the absolute value of x. x represents an individual value. |
| count(array) | Counts all members in an array (e.g., Advanced Table or Repeater). |
| first(array) | Returns first element of an array (e.g., Advanced Table or Repeater). |
| last(array) | Returns last element of an array (e.g., Advanced Table or Repeater). |
| date(dateValue) | Converts argument to date. Works on array children. |
| today() | Returns current date. |
| now() | Returns current date and time. |
| formatDate(date,format) | Converts date argument to formatted string. Works on array children. |
| dateDiff(startDate, endDate, dataPart) | Calculates the difference between two date values (startDate and endDate) in years, months or days (datePart). |
| dateAdd(startDate, interval, datePart) | Calculates the date after which a certain date/time interval has been added. |
| ceil(x) | Returns x, rounded upwards to the nearest integer. x represents an individual value. |
| floor(x) | Returns x, rounded downwards to the nearest integer. x represents an individual value. |
| pow(x,y) | Returns the value of x to the power of y. x represents individual value and y represents the power to value. |
| round(x) | Rounds x to the nearest integer. x represents an individual value. x represents an individual value. |
| num(n) | Converts argument to number. x represents an individual value. |
| str(any) | Converts argument to a string. x represents an individual value. |
| val(x) | Assigns most appropriate data type (e.g., number, boolean, or native type) based on actual value returned. x represents an individual value. |
| type(x) | Returns the JavaScript "typeof" result for the object. x represents an individual value. |
| iff(condition, ifTrue, ifFalse) | Use iff to return a value based on a condition being True or False. If the condition is True, iIf returns ifTrue value; if it is False, returns ifFalse value. |
| select(switchValue, {case: conditionX, value: x}, {case: conditionY, value: y},  ...) | Switch statement, performed from left to right, returning the value for the first case condition that equals the switchValue. conditionX and conditionY represent individual conditions. x and y represent individual values. |
| coalesce(x, y, z, ..., n) | Evaluates the arguments in order and returns the current value of the first expression that does not evaluate to NULL. x, y, z and n represent individual values. |
| trim(x) | Removes leading and trailing white space and line terminator characters from a string. x represents an individual value. |
| toLowerCase(x) | Converts string to lower case. x represents an individual value. |
| toUpperCase(x) | Converts string to upper case. x represents an individual value. |
| replace(text, findWhat, replaceWith) | Replaces text within a string. text represents string element being evaluated. findWhat represents the text to match. replaceWith represents the text to substitute for the matched text. |
| contains(array, x, y, ..., n) | Returns true if array contains all listed values. array represents an array of values. x, y and z represent individual values. |
| lookup(recordSelectionLogicalTest, resultControlTag) | Looks up a value from a column cell or control based on a logical test applied to select the row or record, in an Advanced Table or Repeater, respectively. |

#### Scenarios

The follow section provides a number of different formula examples and samples.

##### Scenario 1: Display the Sum of Numeric Values from a Repeating Section

In this scenario, a number control is used on a repeating section to collect the values to be summed and a calculated control is used to display the total sum on a different form section. The basic formula would be:

|  |
| --- |
| SUM(NUM(`REPEATING\_SECTION\_TAG:NUM\_CONTROL\_TAG`)) |

The components of this formula break down as:

* **`REPEATING\_SECTION\_TAG`** is the section tag for the repeating section on which the referenced control lives. This is required if the referenced control is not in the same section as the calculated control
* **`NUM\_CONTROL\_TAG`** is the numeric control on the repeating section to be used in the calculation.
* **NUM()** is a function that casts the entered value as a numeric. Ensures the addition operator is used for addition rather than calculation
* **SUM()** is the function for summing up all values in the provided list. In this case, it sums up the numeric results from the discharge volume control on the discharges section.

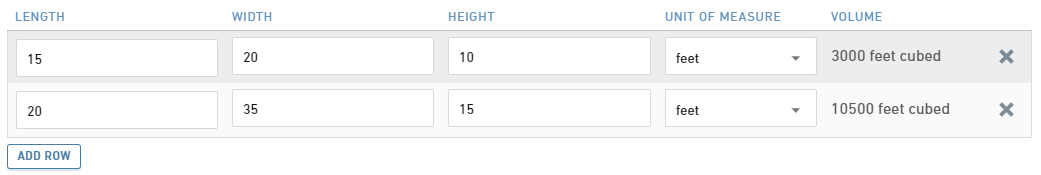
##### Scenario 2: Display the Volume of a Cube in an Advanced Table Control

In this scenario, an advanced table control is being used to collect information on a rectangular room’s dimensions to calculate the volume of the room. The basic formula would be:

|  |
| --- |
| (`LENGTH`\*`WIDTH`\*`HEIGHT`) + " " + `UNITS`.description + “ cubed” |

The components of this formula break down as:

* **`LENGTH `, `WIDTH`,** and **`HEIGHT`** are the tags for the numeric controls within the advanced table control.
* **\*** is the multiplication operator for multiplying the numeric control values
* **+** is being used here to concatenate a series of strings for display.
* **`UNITS`** is the tag for the single select control for units on the advanced table control.
* **.description** is used at the end of the tag due to a selection list being referenced.



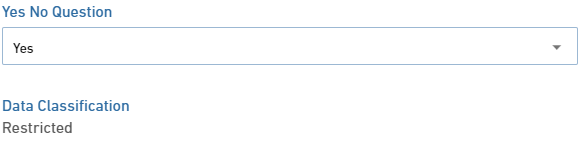
##### Scenario 3: Display Text Conditionally Based on the Answer to a Previous Selection

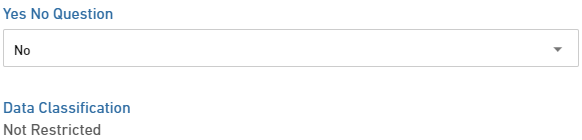
In this scenario, the calculated control displays a certain text string if a certain answer is given to a single select and displays a different string if a different answer is given. The basic formula would be:

|  |
| --- |
| iff(`YES\_NO`.description=="Yes", "Restricted", "Not Restricted") |

The components of this formula break down as:

* **`YES\_NO`** is the tag of the single select control to be used in the calculation
* **.description** is used at the end of the tag due to a selection list being referenced.
* **==”Yes”** indicates that the formula results in true only when the selection choice is “Yes”.
* **Iff()** is the IF THEN function syntax that allows to display one thing if the conditional evaluates to true and another if the conditional evaluates to false. In this case, if the answer to the yes/no question is Yes, “Restricted” will display in the calculated control. If the answer is anything other than Yes, “No Restricted” will display in the calculated control.





An alternative way in which to write this same formula would be to use the ternary operator rather than the Iff function. This formula written with the ternary operator would appear as follows:

|  |
| --- |
| `YES\_NO`.description=="Yes"?"Restricted":"Not Restricted" |

**Note**: The ternary operator is explained in more detail in the Calculation Formulas section of this guide.

##### Scenario 4: Add a $100 Fee If Start Date Within 30 Days of End Date

In this scenario, two date controls are used to collect the start date and end dates. If the start date is within 30 days of the end date, a $100 fee will be assessed. The basic formula would be:

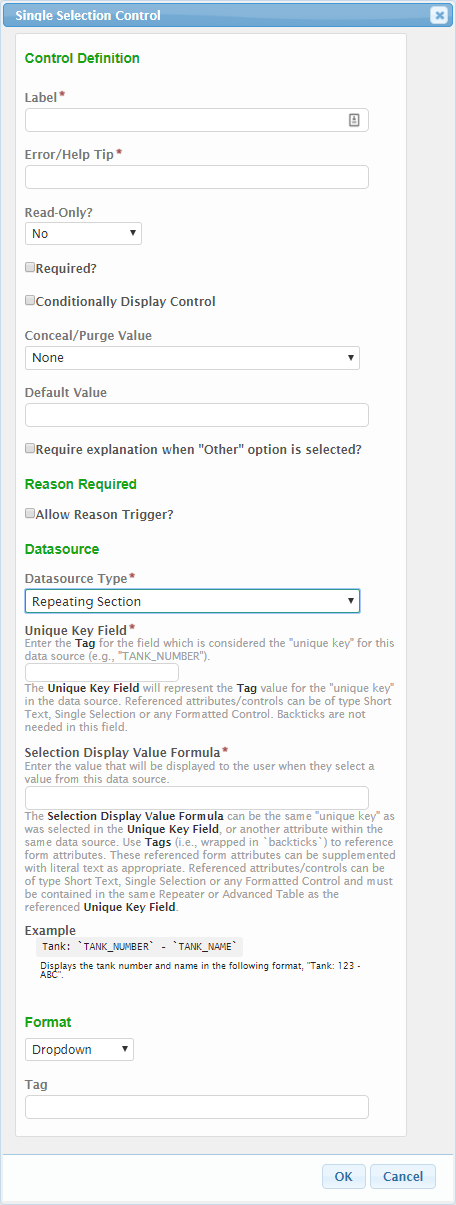
|  |
| --- |
| iff(dateDiff(`StartDate\_Tag`, `EndDate\_Tag`, “days”) >= 30,100,0) |

The components of this formula break down as:

* **`StartDate\_Tag`** is the tag value for the start date field.
* **`EndDate\_Tag `** is the tag value for the end date field.
* **iff(condition, ifTrue, ifFalse)** is the function that returns one of two values, depending on whether the Boolean condition evaluates to true of false. The Boolean expression compares the number of dates between start and end dates and if less than or equal to 30, returns True. Otherwise, return False. If True, 5 is returned. If False, 0 is returned.
* **datediff(startDate, endDate, datePart)** is a function that returns the count (as an integer) of the specified datePart bonudaries crossed between the specified startDate and endDate. “days” is the datePart value.

## Linked Controls

Linked controls allow single and multi-select controls to dynamically display choices from data collected within a submission, specifically data collected in an advanced table control or a repeating section.



To set up a single or multi-select control as a linked control, the following elements must be set:

**Default Value:** If entered, the controls value will be defaulted to the value entered in this attribute.

**Require explanation when “Other” option is selected?:** If checked, an "Other" value will be added to the end of the list. When "Other" is selected by an submitter, an "Other" text field will be activated and required to be entered.

**Reason Required - Allow Reason Trigger?**: If checked, the **Trigger Label** and **Trigger Match Text** attributes will be enabled. The Reason Trigger feature is one where if a particular text value is matched, a custom reason field will appear. This is often used in a Yes/No situation, where if the user answer "Yes", the will need to provide a reason as to why they answered Yes.

* **Trigger Label**: The label that will displayed as the "reason" field.
* **Trigger Match Text**: The text value that will be matched in order to enable/disable the reason field.

**Datasource:** Single and Multi-select controls allow the use of a few different sources for the choices to be displayed to the user. Specifies the Data Source to be used for the list in the control.

* **Typed In:**  Allows user to enter the list of values to present to the submitter. If selected, an **Options** text field will be displayed and allowing the user to 'Type In' the list of values to be available for the control. Each item in the list will be separated by a carriage return. See the Select Control Data Source section of the *Advanced Form Design Guide* for details on this feature
* **Dynamic:** Allows user to select a pre-configured data source to populate the controls list of values. These data sources are stored in the LOOKUP and LOOKUP\_VALUE tables in the database. If selected, **Dynamic Data Source** dropdown field displays allowing for selection of desired list of values. See system administrator for additional details. See the Select Control Data Source section of the *Advanced Form Design Guide* for details on this feature
* **Repeating Section** or **Advanced Table:** Allows user to select a Repeating Section or Advanced Table as the data source. The user will be able to specify the unique control to link to for linking purposes (not necessarily displayed to the user) and then a formula can be entered to specify the text to display to the user in the selectable list. If selected, the **Control tag for Selected Value (saved value)** and **Display Formula for Selection** attributes will display for configuration. See the Select Control Data Source section of the *Advanced Form Design Guide* for details on this feature.

**Format:** The format in which the control selections will display (e.g., Dropdown or Radio Buttons).

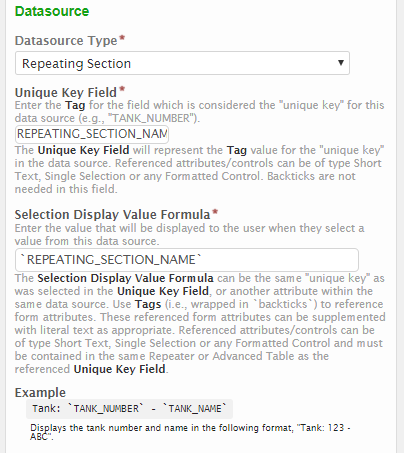
**Control tag for Selected Value (saved value):** The tag for the control to be saved in the database as the submtiter’s response for this single or multi-select. When using an Advanced Table control as the linked control data source, this must be set to the tag of the control within the advanced table control, not the tag of the advanced table control itself. When using a control from a Repeating Section as the linked control data source, this must be set to the tag of the control on the repeating section, not the tag of the repeating section itself. The data from the control indicated in this field and *only* the data from the control indicated in this field will be saved to the database. Concatenation, calculation, and conditionality are not available here.

**Display Formula for Selection:** This field allows the user to create a formula to display to the submitter in the single or multi-select list. Concatenation, calculation, and conditionality are available here, but form designers must keep in mind that this formatting is not what will be saved to the database, but rather what will display to the user for selection.

### Scenarios

#### Scenario 1: Link a Single Select List to a List in a Repeating Section

In this scenario, a repeating section collects information about a series of contacts, including the contact’s name. On a different section, a linked control is set up to allow the user to select a primary contact from the list of contacts inputted in the repeating section.



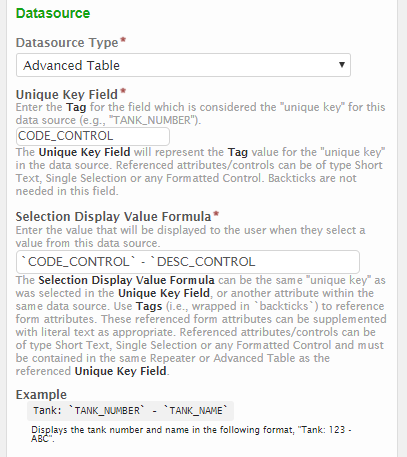
**Datasource:** **Repeating Section**

Control tag for Selected Value (saved data): **REPEATING\_SECTION\_NAME** -This is the tag for the contact name control on the repeating section.

**Display Formula for Selection:** **`REPEATING\_SECTION\_NAME`** - Because the desire is to have the submitter select exactly what will be saved to the database, only the section tag for the contact name control on the repeating section is used. Here, the tag is enclosed in backticks because it is evaluated as a formula.

#### Scenario 2: Link a Multi-Select List to a List from an Advanced Table Control

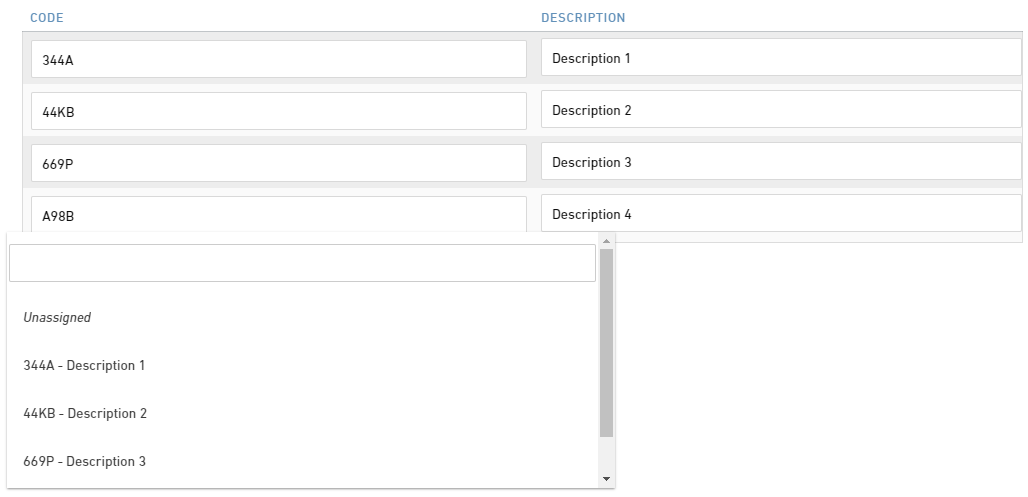
In this scenario, an advanced table is set up that defaults a series of codes and the corresponding descriptions. In the same section, a linked multi-select control is used to allow the collection of a number of those codes that may have special meaning to the submitter. The submitter may know the description of the codes they want, but not the actual code, but the code is what should be selected and stored. The code is set up as the saved value and a formula is created to display the description and the code to the submitter for selection.



**Datasource:** **Advanced Table**

Control tag for Selected Value (saved data): **CODE\_CONTROL** –This is the tag for the code control on the advanced table. Only the code will be saved to the database.

Display Formula for Selection: **`CODE\_CONTROL` – `DESC\_CONTROL`** - In this scenario it is desirable to display both the code and the description to the user to ensure they select the correct code. This formula contains both the tag for the code control and the tag for the description control, formatted to display with a couple of spaces and a dash in between. This allows submitters who may know the description of the item they want, but not the code, to still be able to select the correct code. Note that the explicit text does not need to be enclosed in quotation marks to display correctly.



## Custom Validation

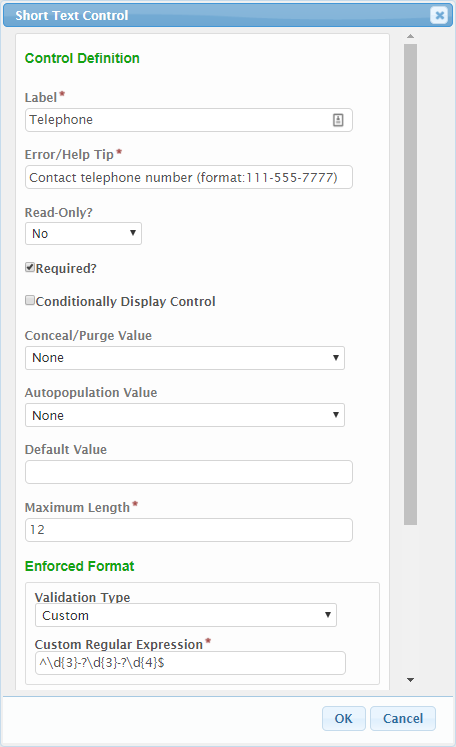
nFORM has several options for validating the data that is submitted by users in nFORM to ensure that data is collected in the proper format. Validation in nFORM allows form designers to ensure that the value entered by the user matches the required pattern or format and prevents users from submitting with values that do not match the defined format.

There are three types of validation available to form designers:

* Formatted Controls
* Pre-built Validation Types for the Short Text Control
* Custom Validation for the Short Text Control

Formatted Controls allow form designers to quickly add controls of a certain format to a form in as few steps as possible, but does not give the form designer additional control over the format that the control accepts. The Short Text control also includes an option to choose an enforced format. Pre-built validations for the Short Text control give the form designer more control over the acceptable values than the formatted controls, but are still limited to the validation formats that the system has supplied. Both formatted controls and pre-built validation types are limited to the formats that are already built into nFORM: number, phone, email, URL, date, and time. While these formats may be sufficient for most forms, some form designers may require more control over the validation.

If the pre-built validation formats are not sufficient for a control, form designers can create custom validation by selecting a Validation Type of Custom and entering a regular expression in the Custom Regular Expression field that appears. A regular expression (commonly referred to as RegEx) is a sequence of characters with specific syntax rules that can be used to form a search pattern. For custom validation in nFORM, this forces the user’s input to match the search pattern specified in the custom validation.



In nFORM, when a user’s input matches the pattern defined by the regular expression, that input is acceptable. Any input that does not match will not be accepted by the system and if the field is required, the user will be unable to submit until a match is entered. For this reason, it is good practice to provide a detailed description of the format the control is expecting into the Error/Help Tip for controls that employ custom validation to ensure users know why their incorrectly formatted input is not being accepted.

Note: there are a number of RegEx libraries available online which can be helpful, but do test/confirm any RegEx’s you select in the nFORM system as not all RegExs presented will be compatible in this context.

### Metacharacters

Within regular expressions, characters that have special meaning within the pattern are called metacharacters. These metacharacters combine with regular characters to specify the pattern to be matched. The metacharacters are {}[]()^$.|\*+? and \.

Metacharacters become regular characters when escaped with the backslash (\), and conversely, certain regular characters can become metacharacters when escaped. For example, if the pattern to match contains a dollar sign (ex, if the user is expected to enter a money value in the format of $1.00) then the $ sign would need to be escaped by preceding it with a backslash in the expression so it is not interpreted as the metacharacter indicating the end of the text. Conversely escaping an s by preceding it with a backslash turns it into a metacharacter representing white space rather than matching an s.

The following is a brief list of metacharacters that are commonly used within nFORM for custom validation and some examples on how they may be used:

*\* = the escape character

ex: *\$\d\.\d{2}* matches on both ‘$5.00’ or ‘$1.50’

*^* = start of text

ex: *^(ad)* matches ‘added’, but not ‘mad’ or ‘sad’.

*$* = end of text

ex: *(ad)$* matches ‘mad’ or ‘sad’, but not ‘added’

*\s* = any white space (spaces, tabs, new lines)

ex: *a\sb* matches ‘a b’

*\d* = any numeric digit (0-9)

ex: *\d* matches ‘0’, ‘1’, ‘2’, ‘9’ etc.

*.* = anything (letter, number, symbol, white space)

ex: *a.c* matches all of the following ‘abc’, ‘a0c’, ‘a c’, ‘a!c’

*\** = the preceding, zero or more times

ex: *ad\** matches ‘a’, ‘ad’, ‘add’, ‘adddddd’, etc.

*+* = the preceding, one or more times

ex: *ad+* matches ‘ad’, ‘add’, ‘addddddd’, etc, but does not match ‘a'

*?* = the preceding, zero or one times

ex: *ad?* Matches ‘a’ or ‘ad’

*|* = the preceding or the succeeding

ex: *a|b* matches either a or b

*()* = defines a marked subexpression

ex: *(ab)+c* matches both ‘abc’ and ‘abababc’ where the subexpression can be repeated one or more times

*{m,n}* = the preceding at least *m*, but no more than *n* times

ex: *(ad){2,3}$* matches ‘abab’ or ‘ababab’ but does not match‘ab’ or ‘abababab’

*[]* = matches a single character that is contained within the brackets

ex: *[a-z]* matches any lowercase character in the alphabet and *[A-Za-z0-9]* matches any alphanumeric character.

### Scenarios

#### Scenario 1: Validate that a Zip Code is a Valid Vermont Zip Code

This regular expression forces the input to start with the characters ‘05’ (*^05*), followed by any three numbers *(\d{3}*), which can then conditionally be followed by a dash and any 4 numbers, zero or one times (*(-\d{4})?*) and nothing more (*$*). What the regular expression does is ensures that the zip code entered begins with ‘05’ to indicate that it is a valid Vermont zip code, and then allows an additional four number extension followed by a dash if necessary. The formats that are acceptable are 05### and 05###-####.

|  |
| --- |
| *^05\d{3}(-\d{4})?$* |

#### Scenario 2: Validate that a Formatted Permit Number

This regular expression forces the input to start with the number 1 (*^1*), followed by a dash and any 4 numbers (*-\d{4}*), and then nothing else (*$*). The regular expression ensures that the value the user enters is a valid UIC permit number in the pattern of 1-####.

|  |
| --- |
| *^1-\d{4}$* |

#### Scenario 3: Validate a Specific Phone Number Format

This regular expression forces the input to start with any 3 numbers (*^\d{3}*), followed by zero or one dashes (*-?*), followed by any three numbers (*\d{3}*), followed by any four numbers (*\d{4}*), then nothing else. The regular expression ensures that the value the user enters is a phone number in a format of either ###-###-#### or ##########. This is particularly useful if a program database is expecting the phone number in a particular format or has restrictions on the length since the standard phone number control allows a number of different formats and doesn’t have restrictions on the number of characters contained within a phone extension.

|  |
| --- |
| *^\d{3}-?\d{3}-?\d{4}$* |

## Data Inheritance

New form submissions can be configured to pre-populate (or inherit) data from a previously submitted form. This is referred to as Data Inheritance in nFORM. This can be a powerful tool for supporting data pre-population, when appropriate. One benefit to this data pre-population technique is that no back-end coding is required – all configuration is implemented within the Form Designer.

The form providing data to the pre-population process is referred to as the “source” form and the form receiving the data is referred to as the “recipient” form. If all controls in the source form are Tagged and the form is published, all remaining configuration is performed on the “recipient” form.

When a submitter selects and begins a form to submit, the user will be prompted to enter the Submission Number for the prior Submission that will provide/donate data to their new recipient Submission.

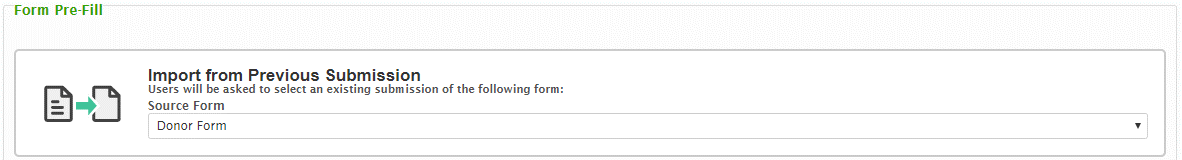
### Configuring for Data Inheritance in the Source Form

The source form, or the form from which data will be pulled to populate the recipient form, must be configured and published before the recipient form can be set up to pre-population from it.

The main configuration settings needed for a source form to be considered ready for Data Inheritance include:

1. Source Form, Sections and controls that will provide data must be fully configured and Tagged. Since pre-population is established by linking recipient form controls to a specific source form, section and control, the configuration must be established to enable this linking.
2. The source form must be been published. Only the published version of a form is available as a source form.

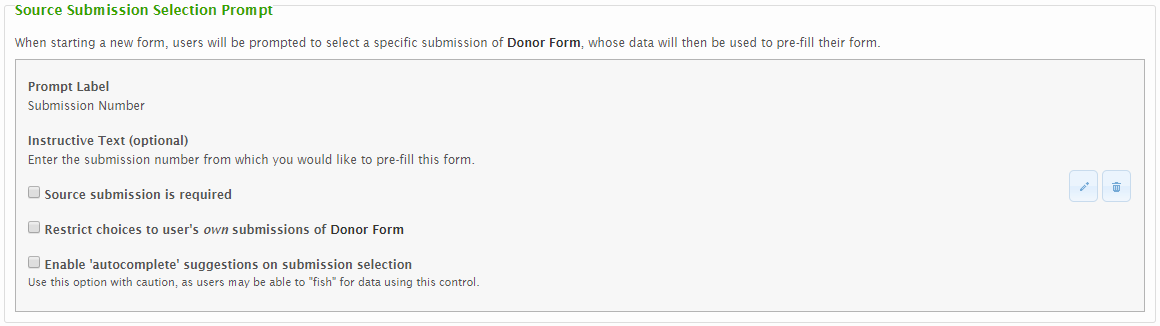
### Configuring Data Inheritance in the Recipient Form



Once the source form has been configured sufficiently, the recipient form will need to be enabled and configured for pre-population. To enable data import from a prior form submission, a source form must be selected. This capability is enabled under the “Pre-fill” tab of the recipient form in the Form Designer.

Under the “Import from Previous Submission” section, available source forms are listed within the drop down menu under “Source Form”. Any previously published form is available for reference. Select the desired source form from this dropdown.

**Note:** Only one form can be configured as the source for a recipient form. However, the same source form can be referenced by numerous recipient forms.Once a form is selected, the Source Submission Selection Prompt box will display.



After the source form has been selected, the “Source Submission Selection Prompt” box will display. Click on the Edit icon to enable the Manage Identifier dialog and to enter a Prompt Label and Instructive Text. These fields will define the Prompt (for Submission Number) and instructive text that will be presented to the external users when they are prompted to provide the source form Submission Number to support the data inheritance process. If the Prompt Label is not configured, the user will not be prompted to enter the Submission Number and pre-population will not work.

If the “Source submission is required” box is checked, a user will be unable to complete and submit the form unless they first provide a Submission Number from which to pre-populate data. This ensures that the form is being pre-populated with existing data but also prevents use of the form if the source form was not previously submitted using this system.

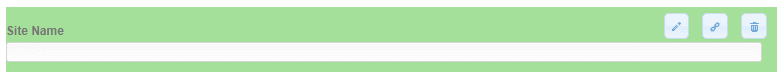
If “Restrict choices to user's *own* submissions of *[Form Name]*” is checked, only prior submissions provided from the same user account will be accepted.

If “Enable 'autocomplete' suggestions on submission selection” is checked, Submission Numbers will be suggested to the user in the Submission Number field based on matches found in the database for the portion of the Submission Number entered. Use this option with caution as users may be able to "fish" for data using this fill option.

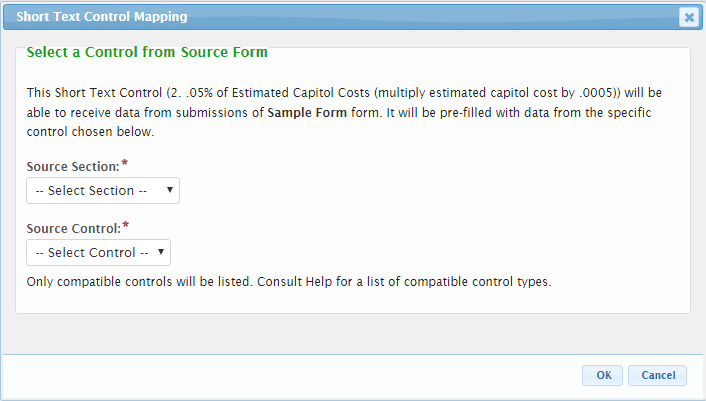
#### Linking Controls from within Recipient Form

Once the form level settings are configured to enable data inheritance, the individual controls within the recipient form will need to be linked to a source control from the source form.

To link a recipient control to a source form control, navigate into a section containing a control to be linked. Hover over the control and a Link Control icon will appear.



Clicking on the Link Control icon will open a mapping dialog box.



In the Mapping window, compatible controls (and sections containing compatible controls) are listed in the dropdowns for “Source Section” and “Source Control”. First, select the Source Section where the desired control resides in the source form. Once selected, the Source Control box will populate with the compatible controls available within the selected section. Select the desired Source Control.

##### Compatible Control Types

All controls can inherit data from controls of the same type. Some controls can inherit from other types as well. See below for a listing of compatible control types for data inheritance.

|  |  |
| --- | --- |
| Recipient Control Type | Compatible Source Control Type(s) |
| Short Text | All Formatted controls (Date, Email, Number, Time, URL, Phone)  Calculated  Hidden  Paragraph  Single Select  Short Text |
| Paragraph | All Formatted controls (Date, Email, Number, Time, URL, Phone)  Calculated  Short Text  Hidden  Paragraph  Single Select |
| Hidden | All Formatted controls (Date, Email, Number, Time, URL, Phone)  Calculated  Short Text  Hidden  Paragraph  Single Select  Paragraph |
| Single Select | Short Text  Hidden  Single Select |
| Formatted Controls | Formatted Control of like-type  Short Text (dependent on format of data)  Hidden (dependent on format of data) |

## Conditional Display

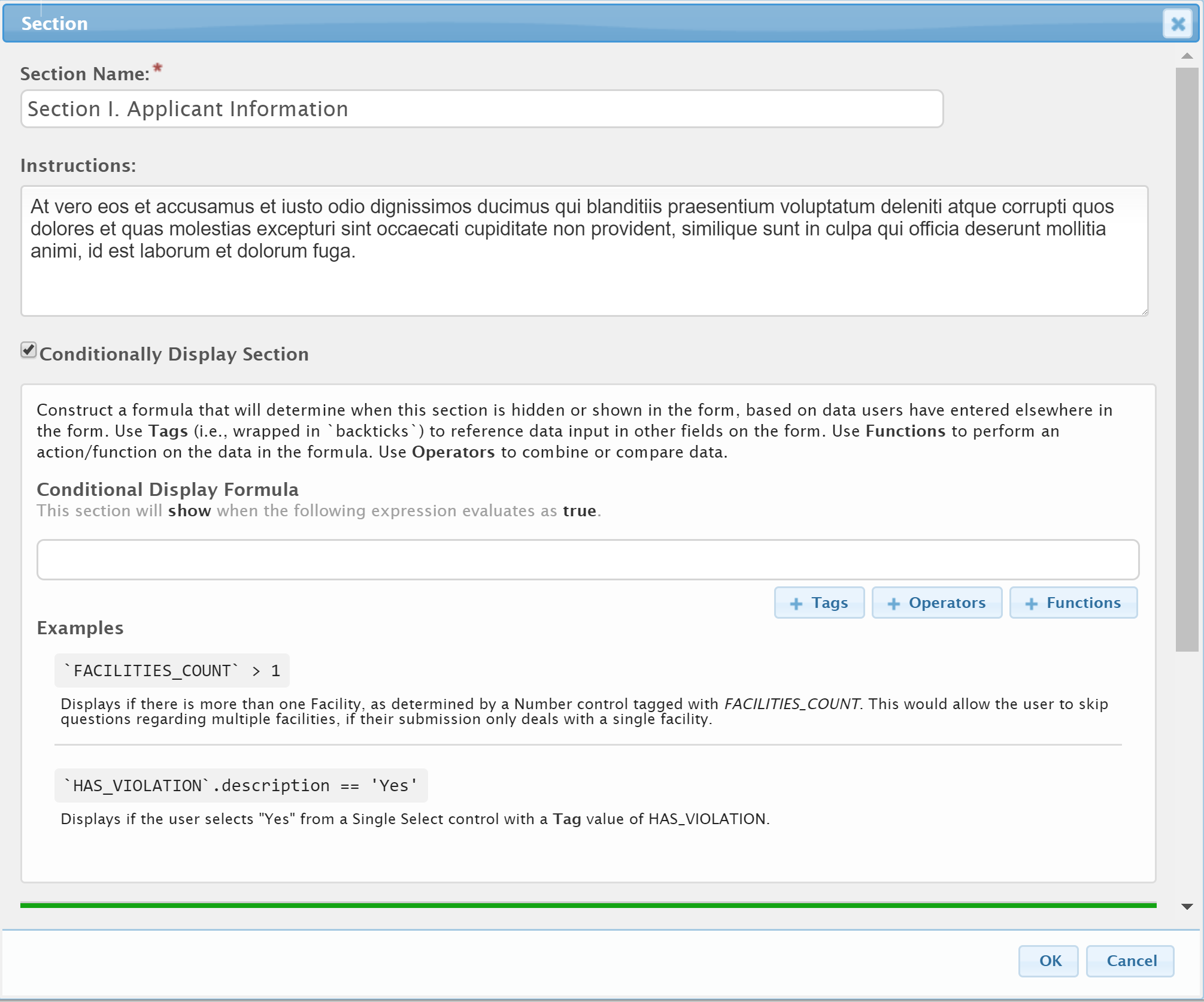
Controls and sections can be configured to display based on a user’s response in one or more controls on a form. This allows for support of a wide variety of business scenarios. For example, if a feature is of type tank, the system can present additional tank specific information. Another example is, if a submitter has received prior violations they can be asked to provide dates/descriptions of the violation(s).

### Form Design

Conditional display can be configured for both a section of a form or an individual control. For both, the process to control display is the same. The section or control will be displayed or hidden based on the results of a formula. A display formula is configured, defining the criteria that would result in the control being displayed.

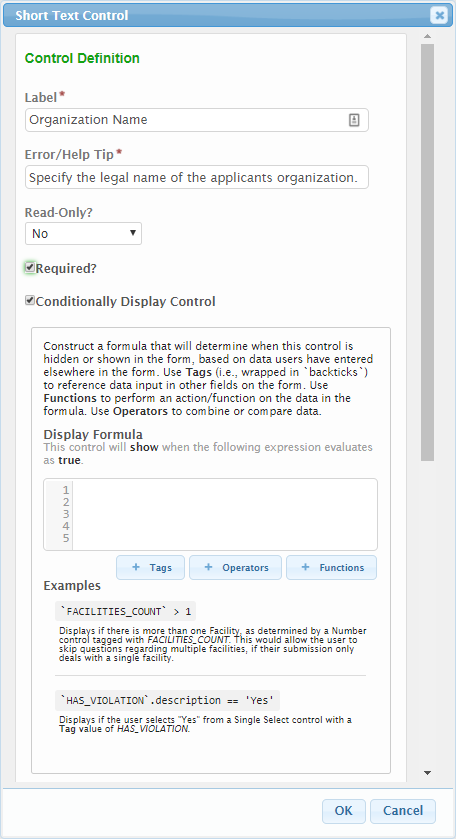
#### Section Conditionally Displayed

To enable the conditional display feature for a section, check the “Section is conditionally displayed” checkbox on the Section design dialog. The Display Formula textbox will appear. See Display Formula section below for additional details on Display Formula.



#### Control Conditionally Displayed

To enable the conditional display feature for a control, check the “Conditionally Display Control” checkbox on the Control design dialog. The Display Formula textbox will appear. See Display Formula section below for additional details on Display Formula.



### Display Formula

Conditional display can be configured for both a section of a form or an individual control. For both, the process to control display is the same. The section or control will be displayed or hidden based on True/False logic. A display formula is configured, defining the criteria that would result in the control being displayed.

* If the formula results to True, then the control is displayed.
* If the formula results to False, then the control remains hidden.

The user can utilize the buttons for Tags, Operators, and Functions to develop their formula with guidance from the system. Alternatively, the formula can be hand key into the Display Formula textbox.

See Formula Builder Examples section of this document for additional examples of these formulas.

#### Tags

Tags in the Display Formula window represent the Tags assigned to controls within the form that are available for reference. When a Tag is referenced in the Display Formula it must be surrounded by backticks (` `). Tags added to the questions from the Tags button will already have these backticks.

A control within the same section as the one configured to conditionally display will be referenced as simply the control tag name in the syntax of `CONTROL\_TAG`. For example:

|  |
| --- |
| `SITE\_CNTY` |

A control in a different section than the one configured to conditionally displayed will be referenced as `SECTION\_TAG:CONTROL\_TAG`. For example:

|  |
| --- |
| `SITE:SITE\_CNTY` |

Controls available for reference will display when the ‘TAGS’ button is selected.

**Notes:**

* When referencing a control in the form, the control Tag must be enclosed in backticks (`). A backtick (`) is a different character from a single quote (‘).
* When referencing a choice within a single or multiple selection control list, a **.description** is required after the closing backtick of the tag. See scenarios below for examples.
* Referencing advanced table controls is possible in nFORM version 4.5 and newer.

#### Operators

Operators indicate how the formula will evaluate the value within a control and the criteria that control is evaluated against. Available operators include, but are not limited to:

* **>=** Greater than or equal to
* **>** Greater than
* **<=** Less than or equal to
* **<** Less than
* **==** Equal to
* **!=** Not equal to
* **+** Addition
* **-** Subtraction

#### Functions

The Functions button lists predefined procedures that have been configured into nFORM for use in evaluating tag values. These functions include, but are not limited to:

* contains
* iff
* num
* str
* max
* min
* avg
* round

### Scenarios

The follow section provides a number of different Display Formula scenarios and example formulas.

#### Scenario 1: Evaluate a choice from a single selection control

In this scenario, an instruction control is configured to conditionally display based on user selection from a single selection control. The basic formula for this would be:

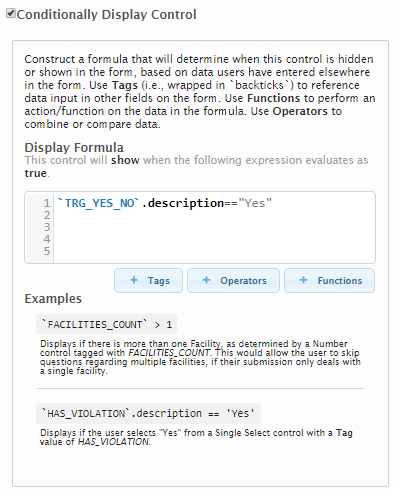
|  |
| --- |
| `SELECTION\_CONTROL\_TAG\_NAME`.description==”Response” |

In this example, the choices within the drop-down box are “Yes” or “No”. The instruction control is configured to display only when the user selects a “Yes” answer. The formula used here is:

|  |
| --- |
| `TRG\_YES\_NO`.description==”Yes” |

The components of this formula break down as:

* **`TRG\_YES\_NO`** is the tag of the triggering control.
* **.description** is used at the end of the tag due to a selection list being referenced.
* **==”Yes”** indicates that the formula results in true only when the selection choice is “Yes”.



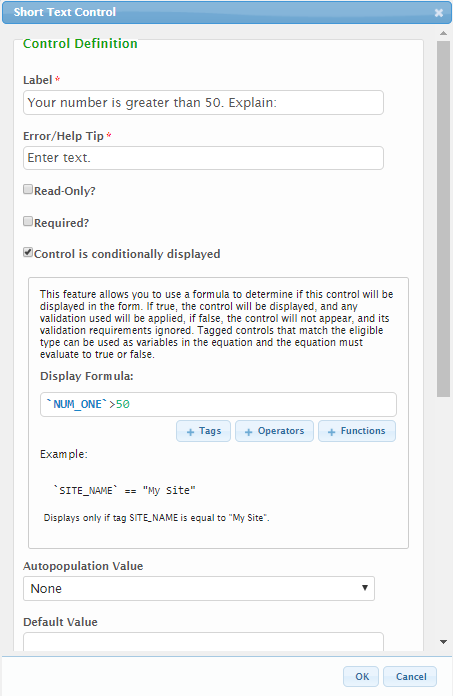
#### Scenario 2: Evaluate against a numerical value input

In this scenario, a user enters a numerical value into a number control. Another control is conditionally displayed if the input value is **greater than 50**. The formula used in the example below is:

|  |
| --- |
| `NUM\_ONE`>50 |

The components of this formula break down as:

* **`NUM\_ONE`** is the tag of the triggering control.
* **>** this operator indicates the relationship (greater than) between the value of the triggering control and a set value that will result in a true response.
* The number **50** indicates the value that the triggering control input is evaluated against. Note that numerical values within formulas are not enclosed in quotes (unlike the “Yes” selection choice in the example above).



#### Scenario 3: Evaluate choice from a multiple-selection control

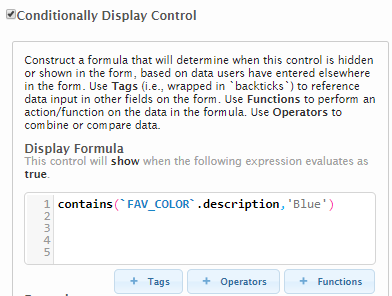
In this scenario, a user selects one or more values from a multiple-selection control. Another control is conditionally displayed based on a particular value selected by the user.

When referencing a multiple-selection control, the **contains** function is used to declare the value or values that will result in a TRUE validation. The formula used in the example below is:

|  |
| --- |
| contains(`FAV\_COLOR`.description,’Blue’) |

The components of this formula break down as:

* **contains** evaluates the select values to determine if any match the criteria defined in the equation
* **`FAV\_COLOR`.description** is the tag of the control being referenced. The .description indicates that the description of the selected value(s) will be evaluated.
* **‘Blue’** is the value of the option within the multiple-selection control which, if selected, will result in a TRUE response. Note the use of single quotes around the value in this scenario.



If more than one selection from the multiple-selection control could result in a TRUE response, the matching options are specified within an iff statement. An example would be:

|  |
| --- |
| iff(contains(`FAV\_COLOR`.description,’Blue’),1,iff(contains(`FAV\_COLOR`.description,’Green'),1, 0)) |

In this example, the selection of Blue and Green would evaluate to TRUE.

#### Scenario 4: Evaluating multiple potential choices from a single-selection control

In this scenario, a control is configured to conditionally display based on user selection from a single-selection control. Unlike in Scenario 1, this formula will search for multiple potential selections using the contains function. The basic formula for this would be:

|  |
| --- |
| contains([‘Value1’,’Value2’,’Value3’],`SELECTION\_CONTROL\_TAG\_NAME`.description) |

For example, consider two controls: one is a single-select control containing a list of counties and the other is a textbox. The formula below is added to the textbox to configure it so that it only displays when the user selects one of the specified counties in the single-select control containing the counties. The formula used here is:

|  |
| --- |
| contains([‘Beaufort’,’Charleston’,’Georgetown’,’Horry’,’Berkeley’,’Colleton’,’Dorchester’,’Jasper’],`SITE\_CNTY`.description) |

The components of this formula break down as follows:

* **contains** evaluates the select values to determine if any match the criteria defined in the list
* **`SITE\_CNTY`.description** is the tag of the control being referenced (the single-select control containing a list of counties). The .description indicates that the description of the selected value will be evaluated.
* **[‘Beaufort’,’Charleston’,’Georgetown’,’Horry’,’Berkeley’,’Colleton’,’Dorchester’,’Jasper’]** is the array of values within the single-selection control which, if any one of these is selected, will result in a TRUE response.

#### Scenario 5: Using iff function to declare TRUE/FALSE

The previous scenario showed how a control can be configured to display if certain conditions evaluate as TRUE—for example, if a specified control **does** contain certain selected values. Using the **iff** function, it is also possible to do the reverse: to configure a control to display if certain conditions evaluate to FALSE. For example, you might want to display a control if a specified control **does not** contain certain selected values. In some cases, such as referencing especially long lists in a single- or multiple-selection control, it is easier to evaluate against only a few values. (In other words, it can be easier to specify “if a control does not contain X” rather than saying “if a control does contain A, B, C, D, E, F, or G”) By combining the **iff** and **contains** functions, the logic shown in the previous scenario can be reversed. The basic formula for this would be:

|  |
| --- |
| iff(condition, ifTrue, ifFalse) |

This can be read as “If **condition** is true, then return the **ifTrue** value, otherwise, return the **isFalse** value”.

Consider an example in which a textbox is configured with the iff formula shown below. This formula references a single-select control to determine if one of the three provided values has been selected by the user. Without the iff function, when one of these three values is selected, a TRUE result is generated and the conditionally displayed control (the textbox) would be shown. Adding the iff function allows the developer to declare when the TRUE and the FALSE responses occur.

|  |
| --- |
| iff(contains([‘Value1’,’Value2’,’Value3’],`SELECTION\_CONTROL\_TAG\_NAME`.description),0,1) |

The components of this formula break down as follows:

* **iff** is the main functionconfigured for the textbox control
* **contains([‘Value1’,’Value2’,’Value3’],`SELECTION\_CONTROL\_TAG\_NAME`.description)** is the condition statement of the iff function, referencing the values of a single-select control on the form.
* **0** is specified as the ifTrue response. In this instance, 0 means FALSE so “if one of the specified values is selected in the single-select control, return FALSE” (i.e., do not display the textbox).
* **1** is specified as the ifFalse response. In this instance, 1 means TRUE so “if one of the specified values is not selected in the single-select control, return TRUE” (i.e., display the textbox).

The end result is that the “contains” function is modified to work as a “does not contain” function.

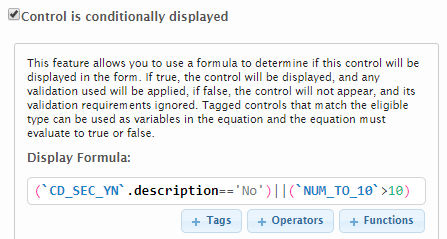
#### Scenario 6: Evaluating against multiple controls

In this scenario, user responses on more than one control are evaluated to determine conditional display.

Using the same methods described in the other scenarios, each control referenced can be individually evaluated. The display formula for each control is surrounded by parentheses and then each formula is connected using either the **or operator** (**||**) or the **and operator** (**&&**).

In the example below, two controls are evaluated. In the first formula, a TRUE result is determined if the user selects ‘No’ from the single-selection control. In the second formula, a TRUE result is determined if the user enters a value greater than 10 in the designated number control. Connecting the two formulas is the **or operator**, represented by two pipe symbols (**||**). When these pieces are put together, the full formula evaluates to a TRUE statement when **either** of the individual formulas evaluates to TRUE.

Conversely, if the **and operator** (**&&**) were used, **both** of the given formulas would need to evaluate to TRUE for the entire expression to evaluate to TRUE.



## Binding Lookup Lists to Form Controls

Lookup values available for selection in a Single-Select control, Multi-Select control and Short Text with Type-Ahead enabled can be configured in a number of different manners. These include:

* **Typed In:**  Allows user to enter a static list of values to present to the submitter.
* **Dynamic:** Allows user to select a data driven list, which is stored in the database.
* **Repeating Section** or **Advanced Table:** Allows user to select values entered on the form in a Repeating Section or Advanced Table.

**Note: Short Text Controls only support the Dynamic method.**

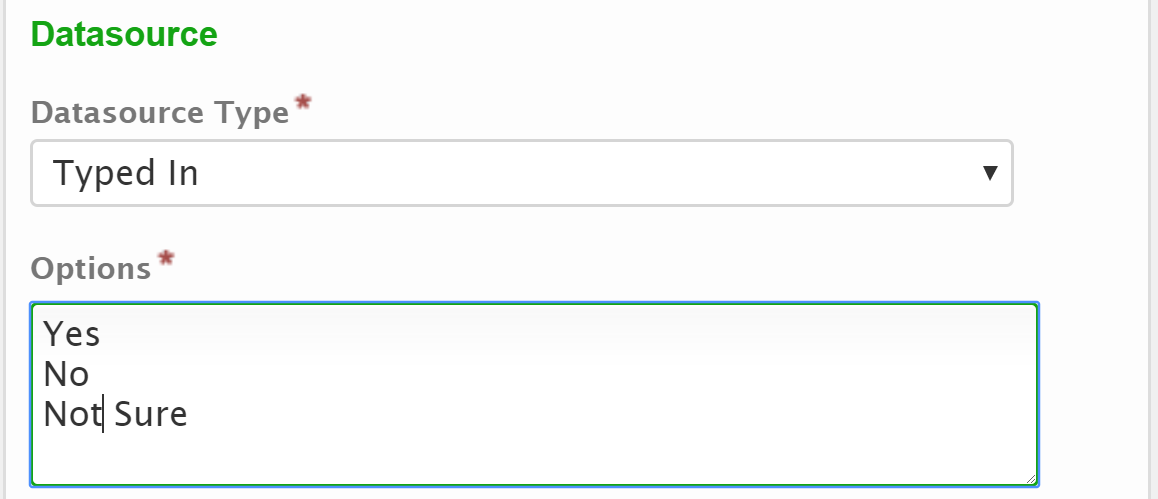
**Tip**: If a lookup list is larger than approximately 50 items, it is recommended that a Short Text control with a Dynamic Auto-Complete data source be utilized, to maintain the performance of the form design (especially in IE).

Use the steps below to configure these data sources.

### Typed In

The Typed In data source allows user to enter a static list of values to present to the submitter in a control.

Use the following steps to configure a Typed In data source:

* Open a Single or Multi-Select control edit dialog (in edit mode).
* In the **Datasource Type** field select “Typed In”. Once selected, an **Options** field will be displayed.  
  
* Type in the list of values to be presented in the control in the **Options** field. Each item in the list will be separated by a carriage return.  
  

### Dynamic

The Dynamic data source allows user to select a data driven list, which is stored in the database. This could be in support of allowing the user to select from a list of items such as Counties, Company Names, Site ID’s/Names, or NAICS codes on a Short Text, Single Select or Multi-Select control. This dynamic data source is stored in the LOOKUP and LOOKUP\_VALUE tables in the database. This list can be updated on a regular basis (e.g., nightly or weekly), if appropriate. See the Dynamic Data Source Configuration section below for more details on how to populate the supporting data tables.

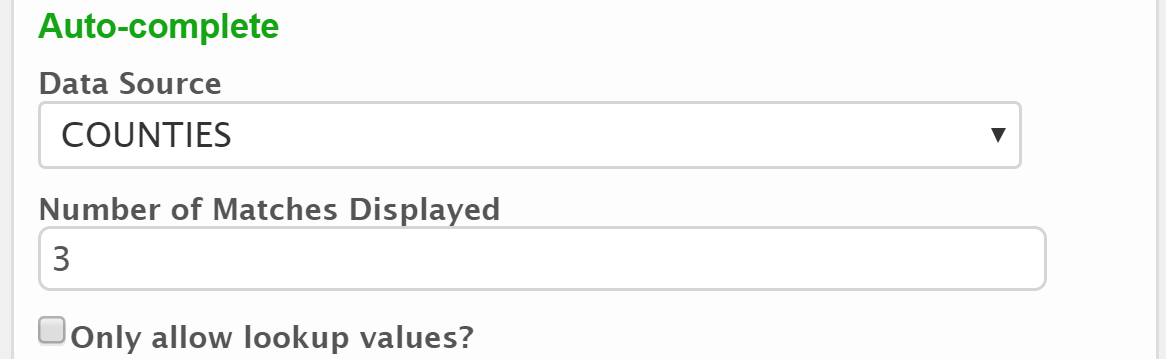
#### Performance Tips for Large Lookups

If you have a large lookup table with more than 50 entries, it is recommended that the Short Text control is used. Slow performance can occur when large lookups are bound to a Single Selection or Multiple Selection Control since the entire lookup table is embedded in the form data which must be downloaded to the user's browser.

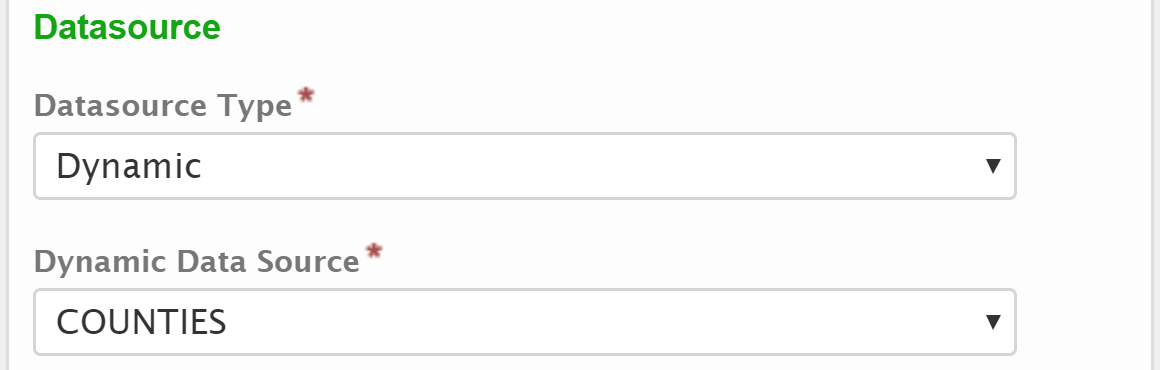
In contrast, when a Short Text control is used, lookup values are dynamically loaded from the server as the user types in the field, reducing the amount of data sent to the user's browser.

#### Control Configuration

Use the following steps to configure a Dynamic data source for a Short Text Control:

* Open a Single-Select, Multi-Select, or Short Text control edit dialog (in edit mode).
* In the Auto-Complete **Data Source** field select the desired dynamic list.   
  
* In the **Number of Matched Displayed** field, specify (as a whole number) the maximum number of suggestions/matches to be displayed in the dropdown list when valid text is entered.
* If the **Only Allow Lookup Values?** Field is checked, the submitter can only select/enter values found in the available list. If unchecked, values not in the list can be entered.

Use the following steps to configure a Dynamic data source for a Single or Multi-Select Control:

* Open the Single or Multi-Select control edit dialog (in edit mode).
* In the Datasource **Datasource Type** field select “Dynamic”. Once selected, a **Dynamic Data Source** field will be displayed.  
  
* Select list to be presented in the control from the **Dynamic Data Source** field. This list will be driven by the lookups lists defined in the LOOKUP table.  
  

#### Dynamic Data Source Configuration

The dynamic data sources are stored in the LOOKUP and LOOKUP\_VALUE tables in the database. These tables must be populated appropriately to support the “Dynamic” lookup technique. The LOOKUP table defines each dynamic lookup list available and the LOOKUP\_VALUE table defines each individual value for selection in the associated dynamic lookup list.

This list is updated in the system as a part of a periodic automated process and can be configured to occur on a regular basis (e.g., nightly or weekly), if appropriate. These tables are described below.

#### **LOOKUP**

Below is a definition of each attributes available in the LOOKUP table which defines each dynamic data source available within the nFORM system:

* **LOOKUP\_ID:** The unique identifier that represents the record.
* **NAME:** The name of the lookup list which will be presented to users in the form designer for selection and use.
* **CREATED\_DATE**:The date and time the record was created.
* **CREATED\_USER**:The unique identifier that represents the user that created the record.
* **UPDATED\_DATE**: The date and time the record was last updated.
* **UPDATED\_USER**: The unique identifier that represents the user that last updated the record.
* **IS\_AVAILABLE\_FOR\_DROPDOWN:** This indicates whether this list can be used as a dropdown list in addition to the type ahead. This should not be enabled on lists exceeding 200 individual entries.

#### **LOOKUP\_VALUE**

Below is a definition of each attributes in the LOOKUP\_VALUE table which defined each individual item in a dynamic data source:

* **LOOKUP\_VALUE\_ID:** The unique identifier that represents the record.
* **LOOKUP\_ID:** The identifier that represents the parent LOOKUP record.
* **NAME**: The short name of the lookup value.
* **DESCRIPTION**: The longer description of the lookup value.
* **CREATED\_DATE**: The date and time the record was created.
* **CREATED\_USER**:The unique identifier that represents the user that created the record
* **UPDATED\_DATE**: The date and time the record was last updated.
* **UPDATED\_USER**: The unique identifier that represents the user that last updated the record.
* **IS\_ACTIVE:** Indicates if the name value pair is still active, and selectable, or if its expired, and needs to be reselected.

The lookup lists are cached when the system is started. If the list is being updated on a regular basis, it will be important that the lookup lists are populated prior to the system being restarted and re-cached for the updates to take effect.

See the Appendix B: Lookup Population topic for techniques on populating these tables.

#### Inactive Values

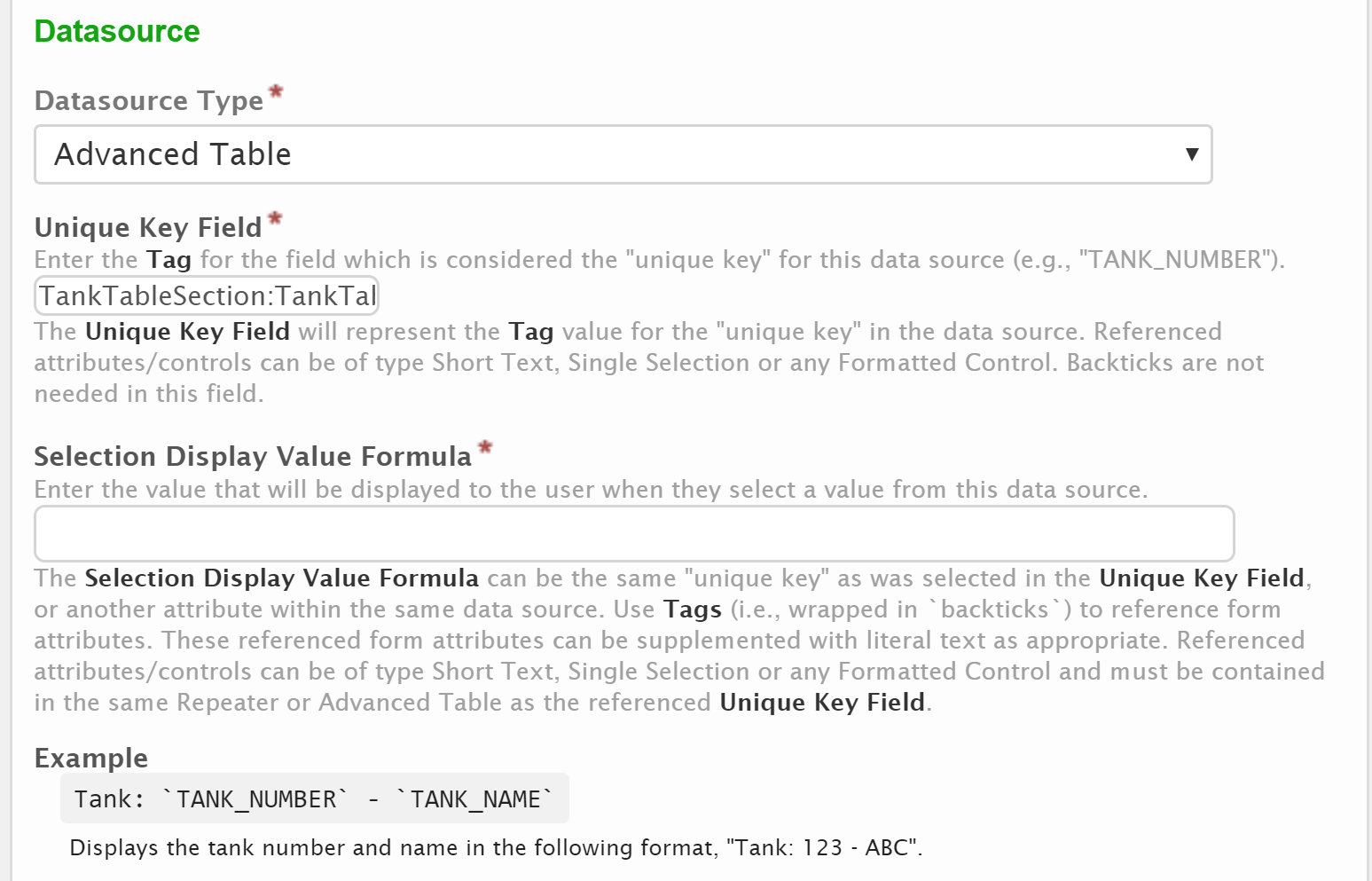
* If a value is outdated and no longer applicable, the *IS\_ACTIVE* value can be set to “0” in the table to inactivate the value. This will disallow the value from being selected in draft and future submissions while preserving historical data.
* Inactive values on the LOOKUP\_VALUE table can be passed to a new submission through data inheritance if that value was used on the submission record being inherited from. However, the field will display warning indicating that the selection is invalid and the user will be unable to submit.

### Repeating Section and Advanced Table

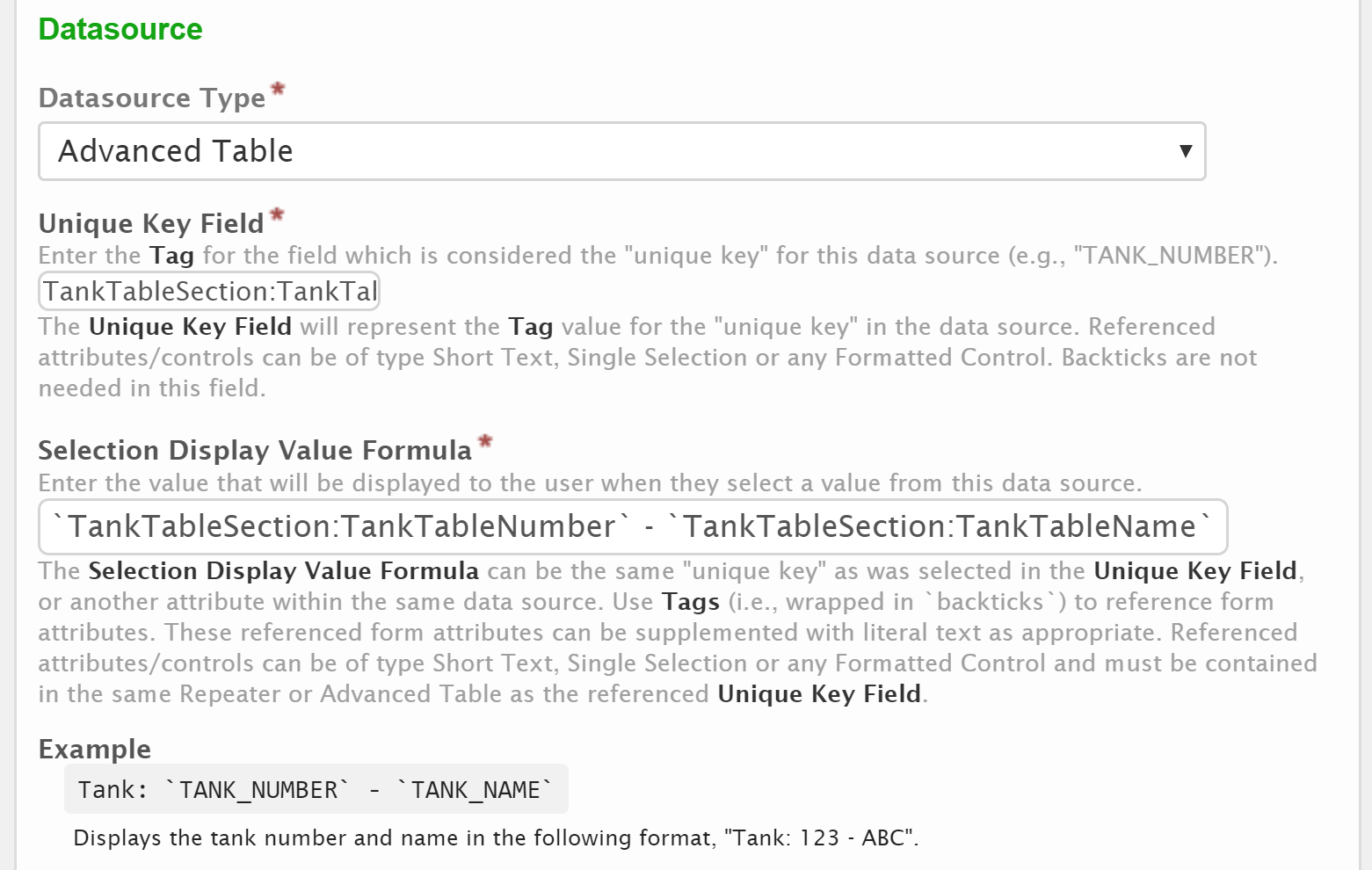
The Repeating Section and Advanced Table data sources allows user to select values entered on the form in a Repeating Section or Advanced Table. This will allow for sections to be associated to one another. For example, if a user defines all tanks used in a system in an Advanced Table, another form section could ask the user which tank is considered the primary grey water tank and the user can select this tank from a list of tanks they defined in the Advanced Table.

The form designer user will be able to specify the unique control used to link the values (this control value not necessarily displayed to the user) and then a formula can be entered to specify the text to display to the user in the selectable list. For example, a “Tank ID” can be utilized as the field that links the lists, while the submitter may only see the Tank Name value in the select list.

Use the following steps to configure a Repeating Section or Advanced Table data source:

* Open a Single or Multi-Select control edit dialog (in edit mode).
* In the Datasource **Datasource Type** field select “Repeating Section” or “Advanced Table”. Once selected, the **Unique Key Field** and **Selection** **Display Value Formula** attributes will display for configuration.  
  
* Enter the Tag for the control on the Repeating Section or Advanced Table that is considered unique, and can be used for linking the value (e.g., Key, ID, Number, etc.) in the **Unique Key Field**. Any value used as the **Unique Key Field** must be required on form entry. This field will represent the primary unique identifier that will be saved with any linked selection.  
  Note: Referenced attributes/controls can be of type Short Text, Single Selection or any Formatted Controls. Backticks are not needed in this field.  
  
* In the **Selection Display Value Formula** field, enter the formula that will be used to determine the text that will be displayed to user for each value in this Single or Multi-Selection control. A great deal of flexibility has been provided with this formula which supports very simple and very complex derivations. See the Calculation Formulas section of this document for more details on acceptable formula formats. Use **Tags** (i.e., wrapped in `backticks`) to reference form attributes. Referenced form attributes/controls must reside in the same Repeating Section or Advanced Table that is referenced in the Unique Key Field. These referenced form attributes can be supplemented with literal text, as appropriate.

**Tip**: It is recommended that values used in this formula be required for form entry.  
**Tip**: Any references a Tag value must be contained within ticks (i.e., `). Please note that the tick (i.e., `) differs from the single quote (i.e., ').

**Note**: Referenced attributes/controls can be of type Short Text, Single Selection or any Formatted control.  
**Note**: The **Selection Display Formula** can include the same “unique key” as was selected in the **Unique Key Field**, or other attributes within the same data source can be displayed, for example `KeyControl`.   


## Conceal and Purge Data

The system provides the ability to conceal and purge sensitive information in form submissions. When a data point is concealed, the information will be hidden from view for most users (users who don’t hold confidential viewer role). When a data point is set to be concealed and purged, the information will be hidden from view from most users and will be purged from the submission, upon submission finalization.

**Note**: Concealed and Purged data points will not be available for export from the system due to their sensitivity.

In summary, the conceal and purge logic supports the following situations:

***Submission in In Process Status***

* External User (Submitter): Can see all data.
* Internal User (Not the Processor): data is hidden in controls marked 'Conceal and Purge' and 'Conceal'.
* Internal User (Processor): Can see all the data.

***Submission in Completed Status***

* External User (Submitter): Can't see data with control of 'Conceal and Purge'. Can see data with control of 'Conceal'.
* Internal User (Not the Processor): Data is hidden.
* Internal User (Processor): Can't see data with control of 'Conceal and Purge'. Can see data with control of 'Conceal'.

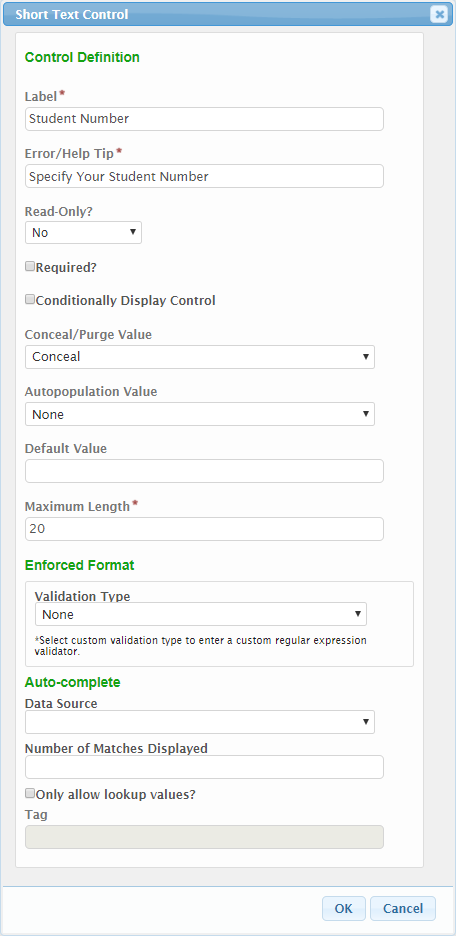
***COR Availability for a Submission in Completed Status***

* External User (Submitter): COR displays the data for both the concealed control and conceal and purge control.
* Internal User (Not the Processor): COR is not available.
* Internal User (Processor): COR displays the data for both the concealed control and conceal and purge control.

### Conceal Data Point

A data point can be concealed. This will obscure the entered value for the data point from view for most users (users who don’t hold the confidential viewer role).

To conceal a data point perform the following:

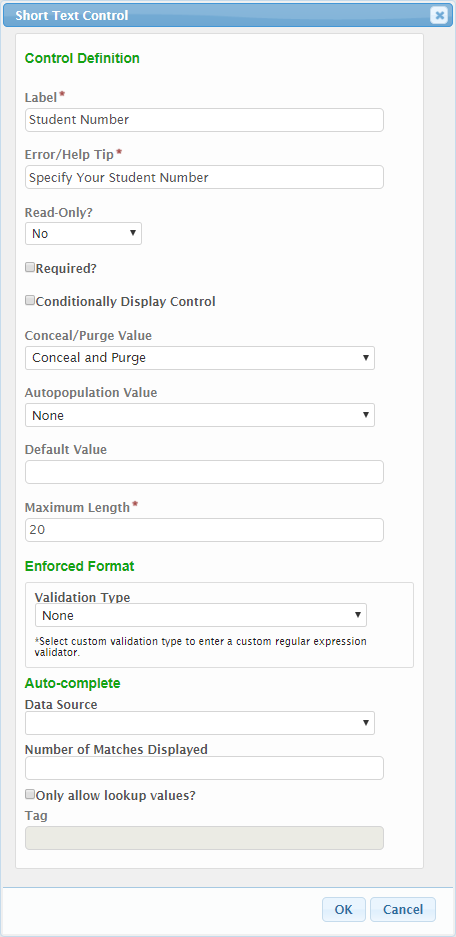
* Click on the **Forms** link at the top of the page to open the **Forms Manager**.
* Click on the **Edit** icon  for the form in question to open the **Forms Designer**.
* Ensure the current version of the form is in a Draft status to ensure the form configuration can be edited.
* Navigate to the **Sections and Controls** (Step 2) section of the Form Designer.
* Click on the **Controls** button  for the section in question to open the **Controls Designer**.
* Click on the **Edit** button  for the section in question to open the **Control** edit dialog.
* Select “Conceal” from the **Conceal/Purge Value** field to set the control to be concealed.  
    
  **Note**: the **Tag** field will be disabled when the Conceal and Purge attributes are used. This is to ensure that this sensitive value is not visible/exportable from the system.
* Click the **OK** button to save the control configuration.

### Conceal and Purge Data Point

A data point can be concealed and purged. When a data point is concealed and purged, the value entered is obscured from view from most users (users who don’t hold confidential viewer role) and will be purged from the submission, upon submission finalization.

Important Note: if the form is configured to established a Copy of Record (COR) for each submission, the COR of record (which may contain the concealed value), will not be purged after finalization, although it will only be accessible for users who hold confidential viewer role. For situations where the value must be completely expunged from the system, a COR should not be established.

To conceal and purge a data point perform the following:

* Click on the **Forms** link at the top of the page to open the **Forms Manager**.
* Click on the **Edit** icon  for the form in question to open the **Forms Designer**.
* Ensure the current version of the form is in a Draft status to ensure the form configuration can be edited.
* Navigate to the **Sections and Controls** (Step 2) section of the Form Designer.
* Click on the **Controls** button  for the section in question to open the **Controls Designer**.
* Click on the **Edit** button  for the section in question to open the **Control** edit dialog.
* Select “Conceal and Purge” from the **Conceal/Purge Value** field to set the control to be concealed.  
    
  **Note**: the **Tag** field will be disabled when the Conceal and Purge attributes are used. This is to ensure that this sensitive value is not visible/exportable from the system.
* Click the **OK** button to save the control configuration.

# Appendix A - Formula Builder Examples

Below are some examples of formulas that can be used in the Formula Builder.

**Exact Text Comparison Example**

The following formula evaluates to True if the SiteName field is equal to "My Site".

|  |
| --- |
| `SiteName` == "My Site" |

**Empty Field Example**

The following formula evaluates to True if the SiteName field is empty.

|  |
| --- |
| `SiteName` == null |

Note: “null” is lowercase.

**Not Empty Field Example**

The following formula evaluates to True if the SiteName field is not empty.

|  |
| --- |
| `SiteName` != null |

Note: “null” is lowercase.

**Select Control Selection (Displayed Description) Exact Comparison Example**

The following formula evaluates to True if the selected (and displayed) value in the SelectMenu field (Single or Multiple Select Control) is equal to "Yes". Note the ".description" selection to signify that the comparison will be performed on the displayed description, versus the hidden ID. If the hidden ID value needs to be referenced, ".id" can be specified.

|  |
| --- |
| `SelectMenu`.description == "Yes" |

**Select Control Selection (Displayed Description) If/Then Comparison Example**

The following formula evaluates to True if the selected (and displayed) Pending Violations Select Control value is equal to "Yes", otherwise returns False.

|  |
| --- |
| iff(`PendingViolations`.description == "Yes", true, false) |

**Select Control Selection (Hidden ID) Exact Comparison Example**

The following formula evaluates to True if the ID for the selected value in the SelectMenu field (Single or Multiple Select Control) is equal to "2".

|  |
| --- |
| `LinkedValue`.id == "2" |

**Select Control Selection (Displayed Description) Contains Comparison Example**

The following formula evaluates to True if the selected (and displayed) value in the County (Single or Multiple Select Control) contains the text "Multnomah", "'Clackamas" or "Washington".

|  |
| --- |
| contains(`County`.description, 'Multnomah','Clackamas','Washington') |

**Simple Math (Multiplication) Example**

The following formula multiplies BoxLength by BoxWidth to calculate area.

|  |
| --- |
| `BoxLength`\*`BoxWidth` |

**Aggregation Function (Summarize) Across Repeating Sections Example**

The following formula sums all the Tons values across all Repeaters in a Repeating Section.

|  |
| --- |
| sum(`RepeatingSection:Tons`) |

**Aggregation Function (Average) Across Repeating Sections Example**

The following formula averages the Tons values across all Repeaters in a Repeating Section.

|  |
| --- |
| avg(`RepeatingSection:Tons`) |

**Return Cell Value from Advanced Table Based on Value in Another Cell**

The following formula returns an [Emissions Value] value from a cell in an Advanced Table where another cell, on the same row, has a matching value [VOC name of Formaldehyde].

|  |
| --- |
| first(lookup(`Emissions\_Table\_Section:Emissions\_Table`.AT\_VOC\_NAME == "Formaldehyde",`Emissions\_Table\_Section:Emissions\_Table`.AT\_Emissions\_Value)) |

**Return Cell Value from Repeater Based on Value in Another Field within Repeater**

The following formula returns an [Emissions Value] value from a field in a Repeater where another field, on the same repeater, has a matching value [VOC name of Formaldehyde].

|  |
| --- |
| first(lookup(`Emissions\_Repeater:VOC\_NAME' == "Formaldehyde",`Emissions\_Repeater:EMISSION\_VALUE')) |

**Count Values Exceeding Limit in an Advanced Table**

The following formula counts the number of rows in an Advanced Table where a [Emissions Value] value exceeds a limit (e.g., 10).

|  |
| --- |
| count(lookup(val(`Emissions\_Table\_Section:Emissions\_Table`.AT\_Emissions\_Value) >  10,`Emissions\_Table\_Section:Emissions\_Table`.AT\_Emissions\_Value)) |

**Count Values Exceeding Limit in a Repeater**

The following formula counts the number of records in a Repeater where a [Emissions Value] value exceeds a limit (e.g., 10).

|  |
| --- |
| count(lookup(val(`Emissions\_Repeater\_Section:RS\_EMISSION\_VALUE`) > 10, `Emissions\_Repeater\_Section:RS\_EMISSION\_VALUE`)) |

**Note:** Copying and pasting these formulas directly from this page to the formula builder field may result in incorrect formula definitions due to conversion issues between the toolsets. Formulas should be hand entered in the formula builder for desired results.

# Appendix B: Lookup Population

Below are some example scripts/procedures which can be run to populate the LOOKUP and LOOKUP\_VALUE tables.

## Using a Script to Manually Create nFORM Lookups

The script below can be used for a one-time loading of a new lookup, populating nFORM LOOKUP and LOOKUP\_VALUE tables from a source table.

|  |
| --- |
| BEGIN    DECLARE @LOOKUP\_ID UNIQUEIDENTIFIER;  DECLARE @USER\_ID UNIQUEIDENTIFIER;    SELECT @LOOKUP\_ID=NEWID() -- This is the nFORM login used for automated operations  SELECT @USER\_ID = PRINCIPAL\_ID from NFORM.PRINCIPAL where LOGIN = 'DATAINIT';    INSERT INTO NFORM.LOOKUP          ( LOOKUP\_ID ,            NAME ,            CREATED\_DATE ,            CREATED\_USER ,            UPDATED\_DATE ,            UPDATED\_USER ,            IS\_AVAILABLE\_FOR\_DROPDOWN          )  SELECT @LOOKUP\_ID , -- LOOKUP\_ID - uniqueidentifier            N'MyName' , -- NAME - nvarchar(255)            GETDATE() , -- CREATED\_DATE - datetime2            @USER\_ID , -- CREATED\_USER - uniqueidentifier            GETDATE() , -- UPDATED\_DATE - datetime2            @USER\_ID , -- UPDATED\_USER - uniqueidentifier            1  -- IS\_AVAILABLE\_FOR\_DROPDOWN - bit            ;    INSERT INTO NFORM.LOOKUP\_VALUE          ( LOOKUP\_VALUE\_ID ,            LOOKUP\_ID ,            NAME ,            DESCRIPTION ,            CREATED\_DATE ,            CREATED\_USER ,            UPDATED\_DATE ,            UPDATED\_USER           -- , IS\_ACTIVE          )  SELECT NEWID() , -- LOOKUP\_VALUE\_ID - uniqueidentifier            @LOOKUP\_ID , -- LOOKUP\_ID - uniqueidentifier            MySourceTable.CODE , -- NAME - nvarchar(600)            MySourceTable.DESCR , -- DESCRIPTION - nvarchar(600)            GETDATE() , -- CREATED\_DATE - datetime2            @USER\_ID , -- CREATED\_USER - uniqueidentifier            GETDATE() , -- UPDATED\_DATE - datetime2            @USER\_ID ,  -- UPDATED\_USER - uniqueidentifier           1  -- IS\_ACTIVE - bit  FROM MySourceTable;    END |