**Windsor Solutions**



Document Generation Design Guide

Updated for Version 4.16

Revision Date: May 11, 2023



This page intentionally left blank

**Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Notes |
| 4.6 | 2/22/2019 | Windsor Solutions | Initial version |
| 4.6 | 5/10/2019 | Windsor Solutions | Minor refinements to InstaPermit references. |
| 4.7 | 5/30/2019 | Windsor Solutions | Updated for v4.7. |
| 4.8 | 2/7/2020 | Windsor Solutions | Updated for v4.8. |
| 4.9 | 7/27/2020 | Windsor Solutions | Updated for v4.9. |
| 4.10 | 9/4/2020 | Windsor Solutions | Updated for v4.10. |
| 4.11 | 1/29/2021 | Windsor Solutions | Updated for v4.11. |
| 4.12 | 7/12/2021 | Windsor Solutions | Adjusted the Submitted By property reference. |
| 4.12 | 7/20/2021 | Windsor Solutions | Updated for v4.12. |
| 4.13 | 4/12/2022 | Windsor Solutions | Updated for v4.13. |
| 4.15 | 12/19/2022 | Windsor Solutions | Updated for v 4.15. |
| 4.16 | 5/11/2023 | Windsor Solutions | Updated for v 4.16. |

**Contents**

[Introduction 1](#_heading=h.gjdgxs)

[Overview 2](#_heading=h.30j0zll)

[Template Design 3](#_heading=h.1fob9te)

[Document Formatting/Layout 3](#_heading=h.3znysh7)

[Referencing Submission Data 4](#_heading=h.2et92p0)

[Template Assignment to Workflow 10](#_heading=h.tyjcwt)

[Appendix 13](#_heading=h.3dy6vkm)

[Standard Submission Attribute References 13](#_heading=h.1t3h5sf)

[Dynamic Submission Attribute References 15](#_heading=h.4d34og8)

# Introduction

nFORM supports the ability to define a Document Template (template) for staff to generate in support of form processing. A template can be designed for the system as a whole as well as 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 InstaPermit, because a primary usage of this capability is to generate a permit after an application has been approved. However, this feature can be used for other official documents such as certificates, licenses, ID cards, etc.

This document provides technical guidance on how to design and implement a document generation template.

# Overview

In order to provide the ability to generate highly formatted documents, Windsor has implemented a feature that allows Form Designers to:

1. Design custom template(s) in Microsoft (MS) Word
2. Assign a template to a workflow tasks in the system
3. Access submission data for inclusion within the template

Templates are formatted/designed in MS Word, using Aspose.Words for .NET and the LINQ coding syntax to access Submission data.

These templates are utilized and a document is produced when called as an action associated with a workflow step, initiated by a submission processor.

If desired, Windsor can configure an example “InstaPermit” form in your environment that can be used as a configuration reference for this document generation feature.

Document generation supports a highly configurable, dynamic, feature set and as such, requires advanced skills and detailed testing/verification. Customers who elect to define advanced configurations in document templates should consult with Windsor to ensure appropriate conventions and techniques are utilized to ensure a successful implementation.

# Template Design

## Document Formatting/Layout

Documents are formatted and designed using MS Word (version 2007 or higher). The MS Word document can be highly formatted and include advanced features such as headers, footers, paging, tables, etc.

Be sure to test out the layout and functionality of the template against a draft or copy of your intended production form to ensure document creation is occurring as expected.

You are in charge of the naming convention, but include no blank spaces or special characters and make sure the file is a **docx**.

### Anatomy of a Template

The anatomy of a template is very straightforward and simply contains standard or dynamic data attributes, along with static text.

**Static Text** – This is information within the document template that does not change in terms of content. This text may change position on the page based on population of static or dynamic data attributes. Static text, could include an image, and can be place in headers or footers.

**Standard Data** **Elements**– This is information that helps provide context for the submission and is based on either submission details or the submitter, but not the submission content. These are available for use in ALL templates as they are standard.

**Dynamic Data** – This is information that is pulled from the submission record itself, so it is data that is within sections and controls on the form that is submitted.

You are in charge of where and how to utilize these template elements. The remainder of this document helps educate on how best to establish a template.

### Building your template

#### Add Static Text First

The best approach to building your template is to design the template utilizing Static Text. You may use all elements of regular document creation including tables, section breaks, headers and footers etc.

#### Identify Submission Data

Identify the positions in your template where you wish to populate with data from a submission. Label them clearly, so that you make sure you address them all during attribute definition. If it is easier for you to complete this at the same time as you add the static text, then please do so. Recommended syntax for identifying submission data is as follows \*the data you want\*. The reason for this is that \* and \* do not conflict with the true beginning and ending tags of and help you to make sure all elements have been addressed. Getting your layout established before you start adding attribute definitions will help you get an outcome that looks as expected.

#### Add Attribute Definition

For each of your identified data elements to be populated from the submission. You must define them to match the syntax expected by the nFORM document generator.

* See details on Referencing Submission Data for including Standard Data Attributes and Dynamic Data Attributes, below.

#### Assign Template to Form

Follow the guidelines provided below to associate your template to your form. You will complete this in the form designer. You will have to associate the template to the form and then establish a workflow task that performs an action of generating a document using the established template.

#### Test, test, test

As mentioned earlier, please be sure to test out the layout and functionality of the template against a draft or copy of your intended production form to ensure document creation is occurring as expected. You can test the document creation via the preview option within the form designer, without having to publish your form – this reduces the number of old versions of your form and is cleaner, so test before you publish.

## Referencing Submission Data

The template has the ability to access data from the submission for reference on the form. In order to access this data, the system uses Aspose.Words and a Aspose specific variation of the LINQ coding syntax. Here are some helpful resources on the template syntax:

* <https://docs.aspose.com/display/wordsnet/Template+Syntax>
* <https://docs.aspose.com/display/wordsnet/Typical+Templates>

### Standard Data Attributes

When accessing standard submission data (e.g., Submission Date, Parent Organization, Submission Number, etc.), the Submission attributes can be accessed directly.

See the available static references available in the Appendix under the topic, *Standard Submission Attribute References*.

#### Basic insertion of a standard data attribute

The following example returns a formatted Submission # and is an example of the syntax for a basic standard data attribute.

|  |
| --- |
| <<[submission.formattedNumber]>> |

#### Formatting a date

The following example returns the date of the submission, formatted as a date as specified.

|  |
| --- |
| <<[submissionVersionSubmitted]:”M/d/yyyy h:mm tt”>> |

#### You can manipulate the format using the following format codes.

Format Code Meaning

M Displays one or two digits for the month, as necessary.

MM Always displays two digits for the month.

MMM The three-letter abbreviation for the month.

MMMM The full name of the month.

d Displays one or two digits for the day of the month, as necessary.

dd Always displays two digits for the day of the month.

ddd The three-letter abbreviation for the day of the week.

dddd The full name of the day of the week.

yy Always displays two digits for the year.

yyyy Always displays four digits for the year.

h Displays one or two digits for the hour, as necessary on a 12 hour clock.

hh Always displays two digits for the hour on a 12 hour clock.

H Displays one or two digits for the hour, as necessary on a 24 hour clock.

HH Always displays two digits for the hour on a 24 hour clock.

m Displays one or two digits for the minutes, as necessary.

mm Always displays two digits for the minutes.

tt Includes the am/pm designation.

**Some examples**

|  |  |
| --- | --- |
| Definition | Sample Result |
| ”MMMM d, yyyy h:mm tt” | February 20, 2019 4:03 pm |
| ”M/d/yyyy HH:mm” | 2/2/2019 16:03 |
| ”MMM d, yyyy h:mm tt” | Feb 20, 2019 4:03 pm |
| “dddd, MMM d, yyyy h:mm tt” | Wednesday, Feb 20, 2019 4:12 PM |

#### Amount Due Example

The following example returns the amount currently due, formatted as a currency.

|  |
| --- |
| <<[submission.processingFee + submission.adjustments - submission.onlinePaymentsTotal]:”C”>> |

### Dynamic Data Attributes

When accessing dynamic data in the submission (e.g., custom data points defined in form sections by a form designer), there are a few steps that should to be followed, such as:

1. Define variables at the top of the template for the data to be accessed. These are an important aspect for dynamic data as a pre-check for data existence helps prevent a template failure during document generation.
2. Perform any needed concatenation and appropriate error handling at the top of the template.
3. Reference the data point using the defined variable within the core of the template.

Following these steps will allow the template to minimize code within the core of the template and help preserve the integrity of the template.

Speaking to bullets #1 and #2 above, the designer should define the variables to be referenced and perform any needed concatenation and error handling at the top of the document. Here are some examples of how to perform this task.

#### Creating a Variable

Variables allow us to avoid having null values that could crash the template. Here is some guidance on how to create a variable in different circumstances.

1. Declare that we are defining a variable - var
2. Give the variable a name - [NameOfVariable
3. Identify what control to map the variable to by pointing to the control tag - = SectionControls.FirstOrDefault(sc=>sc.tag ==”SITE\_NAME”)
4. Specify which attribute within the control you are populating the variable with - .values.controlValue]
5. Close the Variable
6. Your variable declaration should look something like the following - var [siteName = SectionControls.FirstOrDefault(sc=>sc.tag ==”SITE\_NAME”).values.controlValue]

Here are some examples:

##### Short Text/Paragraph Example

In this example, the template is attempting to access text from a Short Text or Paragraph Control, the “Site Name”, that the user entered in the submission. The Short Text or Paragraph (e.g., Site Name) Control in the submission has a **Tag** value of “**SITE\_NAME**”. The first step is to define the **siteName** variable. Assuming Site Name control will always be available on the Submissions, use the following syntax to define the siteName control:

|  |
| --- |
| <<var [siteName = SectionControls.FirstOrDefault(sc=>sc.tag ==”SITE\_NAME”).values.controlValue] >> |

Once defined, siteName can be referenced from anywhere in the template using the following syntax:

|  |
| --- |
| Site Name: <<[siteName] :upper>> |

Note: the “:upper” reference is used to make all text uppercase.

##### Address Example

In this example, the template is attempting to access text from an Address Control, the “Site Address”, that the user entered in the submission. The “Address” (e.g., “Site Address”) Control in the submission has a **Tag** value of “ADDRS”. The code below performs the following tasks:

* Defines the **siteAddressCtrl** variable so simplify the many references to this control.
* Performs error handling to handle the situation where the site address is null.
* The **siteAddress** variable is established by concatenating all the constituent parts of the address control (e.g., street address, secondary address, city, state, zip, etc.) and to improve formatting, HTML break tags are added where needed.

See example below:

|  |
| --- |
| <<var [siteAddressCtrl=SectionControls.FirstOrDefault(sc=>sc.tag ==”ADDRS”)]>>  <<var [siteAddress = siteAddressCtrl == null ? “”:”\r”+ siteAddressCtrl.values.street + “\r” + siteAddressCtrl.values.street2 + (siteAddressCtrl.values.street2 != null?“\r”:””) + siteAddressCtrl.values.locality +”,” + siteAddressCtrl.values.areaCode + “ “ + siteAddressCtrl.values.postalCode] >> |

Note: each individual address component could be referenced separately (versus one address block, if desired.

Syntax "\r" provides a return; so does "\r\n".  The use of "\r\t", results in a return carriage and a tab. Practice to determine a result that meets your needs.

One defined, the siteAddress can be referenced from anywhere in the template using the following syntax:

|  |
| --- |
| Site Address: <<[siteAddress]:upper:”format” –**html** >> |

Note: the “:upper” reference is used to make all text uppercase. **The “format” –html** reference used to ensure the displayed text honors the HTML formatting tags, in case you would like to use html markup such as “<br />” instead of “\r”, or <strong>, etc.

##### Contact Control Example

In this example, the template is attempting to access a Contact Control, the “Submitter” information that the user entered in the submission. The Contact (e.g.,. Submitter) Control in the submission has a **Tag** value of “CNTCT”. The code below performs the following tasks:

* Defines the **submitterCtrl** variable so simplify the many references to this control.
* Performs error handling to handle the situation where the company name is null.
* The **submitterInfo** variable is established by concatenating some of the constituent parts of the submitter control (e.g., name, street address, secondary address, city, state, zip, etc.) and to improve formatting, HTML break tags are added where needed.

See example below:

|  |
| --- |
| <<var [submitterCtrl=SectionControls.FirstOrDefault(sc=>sc.tag ==”CNTCT”)]>>  <<var [submitterInfo = submitterCtrl == null ? "":"<br/>"+ submitterCtrl.values.companyNameContact+"<br/>"+ submitterCtrl.values.streetContact + "<br/>" + submitterCtrl.values.street2Contact + (submitterCtrl.values.street2Contact !=null?"<br />":"") + submitterCtrl.values.localityContact +"," + submitterCtrl.values.areaCodeContact + " " + submitterCtrl.values.postalCodeContact]:"format" –html>> |

Note: each individual address component could be referenced separately (versus one address block, if desired.

One defined, the **submitterInfo** can be referenced, in uppercase, from anywhere in the template using the following syntax:

|  |
| --- |
| <<[submitterInfo]:upper:”format” –html>> |

Note: the “:upper” reference is used to make all text uppercase. The “format” –html reference used to ensure the displayed text honors the HTML formatting tags.

#### Other Considerations

Below are some additional considerations when presenting dynamic data attributes:

* When defining the variables for the data to be accessed:
  + Perform this work at the top of the page (prior to the data attribute being referenced)
  + Do NOT add carriage returns in or between these definitions as the template will include these carriage returns in the presented document.
* If a referenced control in the template does not contain data within the submission, we must trap that to avoid a template failure. To trap for potentially absent data in a control, the following format of code can be used.

|  |
| --- |
| <<var [companyNameCtrl = SectionControls.FirstOrDefault(sc=>sc.tag==”COMPANY\_NAME”)]>>  <<var [companyName= (companyNameCtrl == null ?””: companyNameCtrl.values.controlValue)]>> |

* If a Control is empty (e.g., not entered), the value returned to the template will be a blank. In some cases, blank values are saved in controls as a “?” to help with URL encoding. In these cases, the “?” will need to be replaced with a blank value during display.
* Case does matter when referencing variables so be sure that your casing is in sync.
* **Configuration values** (in the Vars.config, VarsOverrides.config, etc.) can be exposed to the document generator. These can be exposed by referencing the existing configuration properties in the following format:

|  |
| --- |
| <add key="docgen.config.settings" value="  app.title,  app.name,  message.email.appBaseUrl "/> |

Any attributes referenced here can be accessed from within the document by referencing the configuration value without periods and capitalizing secondary words. So the app.Title configuration setting will be referenced as:

|  |
| --- |
| <<[appTitle]>> |

The message.email.appBaseUrl configuration setting will be referenced as:

|  |
| --- |
| <<[messageEmailAppBaseUrl]>> |

* **Internal values** can be exposed to the document generator. They can be exposed by referencing the existing internal data element in the following format:

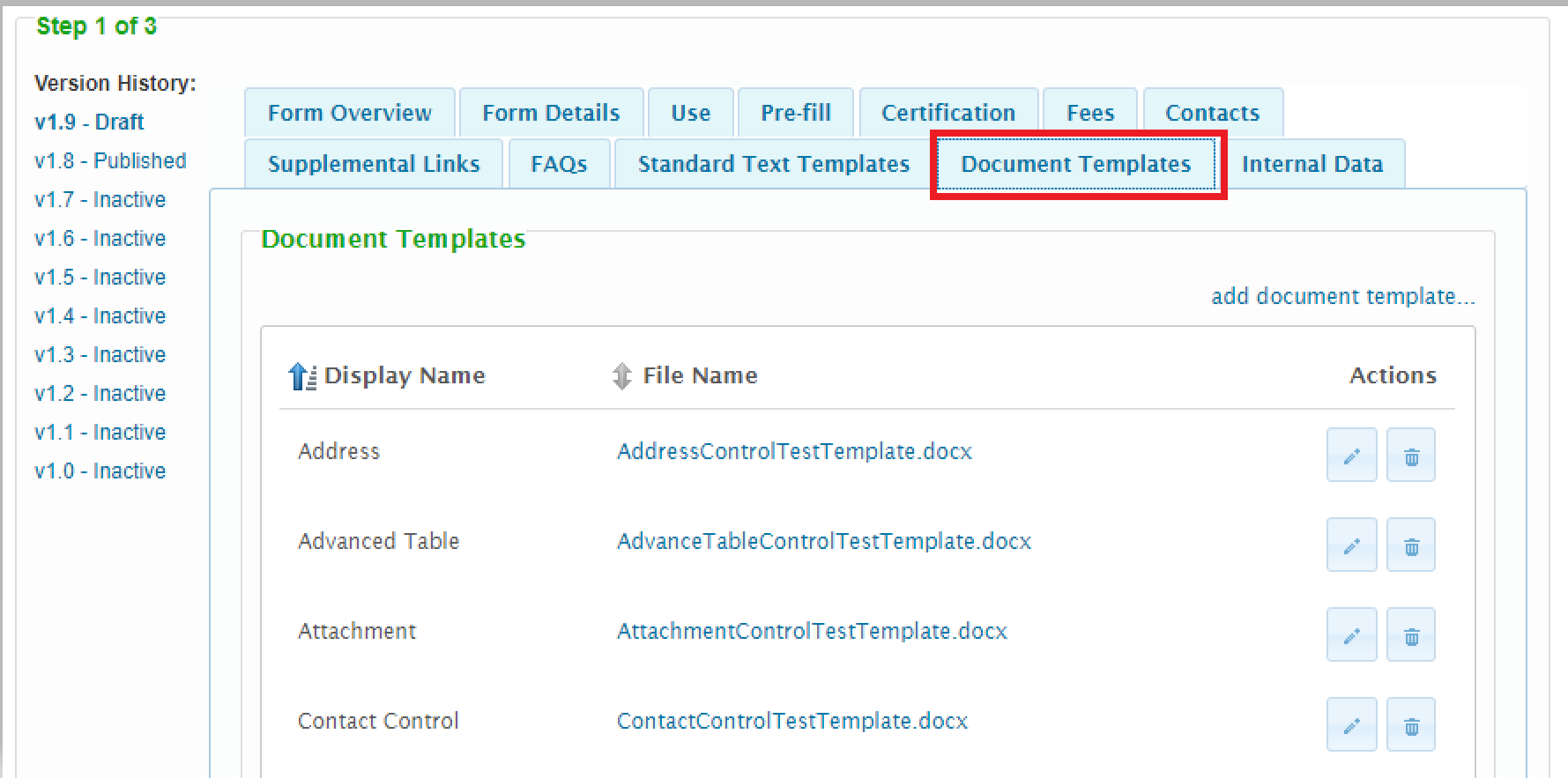
|  |
| --- |
| <<var[NewCert=InternalDataControls.FirstOrDefault(sc=>sc.definitionDictionary.tag =="CertNumInitial")]>> |

Once the variable is set you can reference it using the following structure:

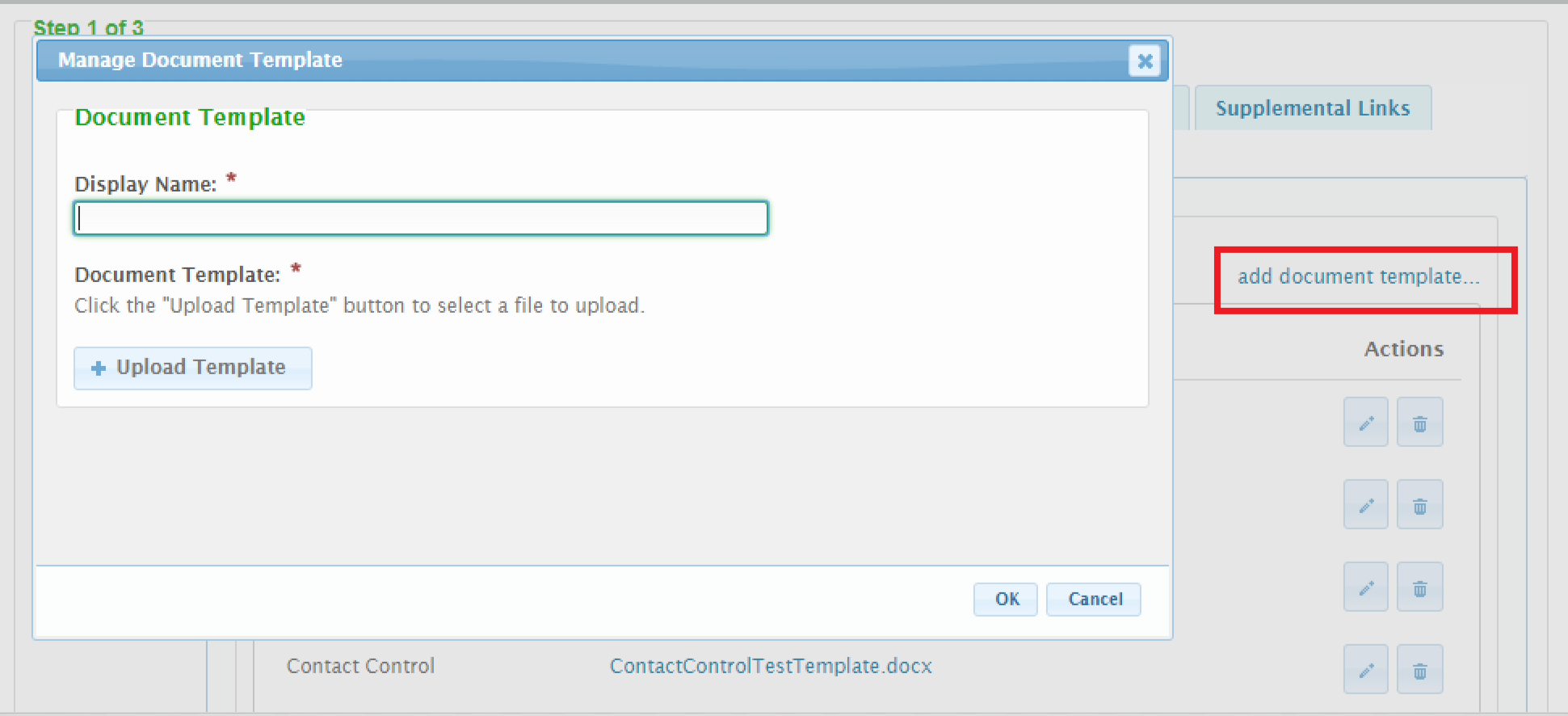
|  |
| --- |
| Certificate Number: <<[NewCert.values]:upper>> |

# Template Assignment to Workflow

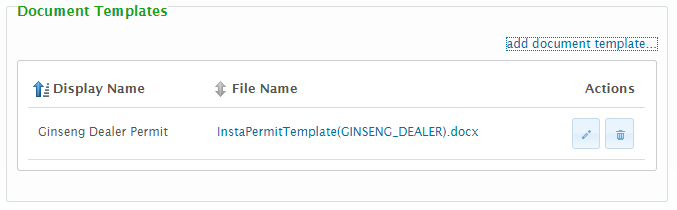
In order for a document to be generated using a designed template, it must be attached to a workflow step as an action. In order to do this the form designer must first associate the template with the form via the form designer Document Templates tab.



Add the document template and provide a display name.



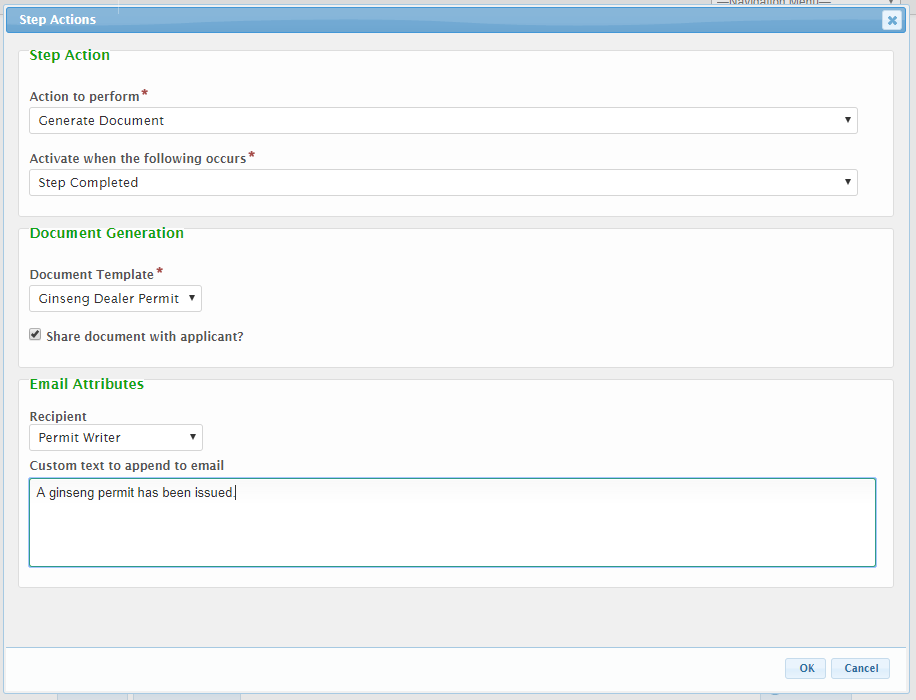
You will then see the document template attached to the form and it is now available for use in your workflow.



In the workflow step, define an action against the step that is responsible for generating the document as follows.

Specify the action of Generate Document and select the trigger, example ‘Step Completed’

Then pick the Document Template. There will likely just be one but the application supports many, so pick the one you wish to generate.



Specify whether or not this document will be shared with the submitter. Activating this setting will make the generated document visible to the submitter if they open the record in their account.   
Note: The documents are not currently emailed to the submitter, although this option is being considered for inclusion in an upcoming release.

Include any additional text to include in the email notification.



# Appendix

## Standard Submission Attribute References

Below is a list of key standard submission attributes that are available for reference in the template.

Note: more properties may be available. Please see the nFORM Control Attributes Reference Guide for guidance if an attribute needed is not referenced.

You may use any of these standard data elements in your template, just copy and paste to the location you want, even in the header and footer.

**FORM DATA**

* Form Header
  + Form Name:
    - Form Name: <<[formVersion.name]>>
    - Form Short Description: <<[formVersion.title]>>
    - Reference Number: <<[formVersion.referenceNumber]>> (this will only be populated after a processor has provided this number, post submission )
  + Alternative Identifier
    - Label: <<[formVersion.altIdentityLabel]>> (only the label is from the form version)
  + Reason for Submission: <<[formVersion.formType]>> (this will only be populated when the Reason for Submission attribute is enabled on the form design)
  + Online Payments Accepted: <<[formVersion.isOnlinePaymentAvailable]>>
  + Form Version:
    - Full Version Number: <<[formVersion.version]>>
    - Major Version Number: <<[formVersion.versionNumber]>>
    - Minor Version Number: <<[formVersion.minorVersion]>>
* Fees
  + Account Number: <<[formVersion.accountNumber]>> (must be populated on form design)
  + Flat Fees (static fee list): <<foreach [item in formVersion.fees]>><<[item.description]>>: $<<[item.amount] >><</foreach>>
  + Calculated Fee (will not have values if fee is not calculated)
    - Fee Description: <<[formVersion.fees[0].description]>>
    - Is Payment Calculated: <<[formVersion.paymentIsCalculated]>>
    - Minimum Threshold: <<[formVersion.fees[0].MinimumFee]>>
    - Maximum Threshold: <<[formVersion.fees[0].MaximumFee]>>
  + Fee Amount: <<[formVersion.processingFee]>> (not really relevant for FORM VERSION)
* Addresses: <<foreach [item in formVersion.addresses]>><<[ item.addressTypeName]>>:<<[ item.fullAddress] >><</foreach>>

### SUBMISSION DATA

* Submission Header:
  + Submission Number: <<[submission.formattedNumber]>>
  + Revision Number: <<[submission.LastSubmittedSubmissionVersion]>> (not yet available)
  + Submission Locked Indicator: <<[submission.isLocked]>>
  + Is Test/Preview Submission: <<[submission.isTest]>>
  + Submitted Date: <<[submissionVersionSubmitted]>>
  + Submitted By: <<[SubmissionVersionSubmittedBy]>>
  + Responsible person information: <<[responsiblePerson.LastName]>>

<<[responsiblePerson.FirstName]>>

<<[responsiblePerson.DisplayName]>>

<<[responsiblePerson.PrimaryEmail]>>

* Fees
  + Account Number: (entered by Designer)
  + Payment Remittance Address: (Not yet available)
  + Fee Description: <<[submission.fee.description]>>
  + Calculated Fee
    - Is Payment Calculated: <<[formVersion.paymentIsCalculated]>>
    - Minimum Threshold: <<[submission.fee.min]>>
    - Maximum Threshold: <<[submission.fee.max]>>
  + Fee
    - Amount: <<[submission.processingFee]>>
    - Formula: <<[submission.fee.formula]>>
  + Payments/Adjustments:
    - Manual Payments/Adjustments: <<[submission.adjustments]>>
    - Online Payments: <<[submission.onlinePaymentsTotal]>>
    - Total Payments/Adjustments: <<[submission.paymentsTotal]>>
  + Payment Status:
    - Payment Status: <<[submission.financialStatus.name]>>
    - More Payments Required?: <<[submission.hasMorePaymentsRequired]>>

### ORGANIZATION DATA

* Organization Header
  + Top-Level System Organization Name: <<[systemOrganizationName]>>
  + Organization Name: <<[organization.name]>>
  + Organization Code: <<[organization.organizationCode]>>

### USER DATA

* User Information:
  + Name:
    - Full Name: <<[loggedInUser.displayName]>>
    - User Name: <<[loggedInUser.login]>>
    - Email: <<[loggedInUser.primaryEmail]>>
  + Company: <<[loggedInUser.company]>>
    - Address: <<[loggedInUser.addresses.FirstOrDefault(a => a.addressType.Name =="home")]>> This is not yet available.

## Dynamic Submission Attribute References

Submission attributes that are custom to a form or dynamically generated, can also be referenced on a template. Control values can be referenced in a format such as below, here *Tag Value* is the Tag assigned to the control and *Control Type* is the control type of the attribute being accessed. Please see the Control Value Properties Appendix within the nFORM Integration Guide for available Control Types that can be referenced for a Control.

|  |
| --- |
| SectionControls.FirstOrDefault(sc=>sc.tag ==”[Tag Value]”).values.[Control Type]] >> |

**Note:** more properties may be available. Please see the nFORM Control Attributes Reference Guide for guidance if an attribute needed is not referenced.

The following describe how you would normally reference each control. Your Variable name is set to what you want to name the variable, the tag should be set to the tag within the control you wish to access.

### Simple

#### Short Text

**Establish Variable Example**

|  |
| --- |
| <<var [ShortText=SectionControls.FirstOrDefault(sc=>sc.tag ==”STC”).values.controlValue]>> |

**Reference Variable Example**

|  |
| --- |
| Short Text: <<[ShortText]>> |

#### Paragraph

**Establish Variable Example**

|  |
| --- |
| <<var [Paragraph = SectionControls.FirstOrDefault(sc=>sc.tag ==”PRGRPH”).values.controlValue] >> |

**Reference Variable Example**

|  |
| --- |
| Paragraph: <<[Paragraph]>> |

#### Instructions

No attributes are returned from this control.

#### Single Selection

**Establish Variable Example**

|  |
| --- |
| <<var [Single=SectionControls.FirstOrDefault(sc=>sc.tag ==”SSC”)]>> |

**Reference Variable Example**

|  |
| --- |
| Selection: <<[Single.values.select]>>  Reason: <<[Single.values.otherReason]>> |

#### Multi Selection

**Establish Variable Example**

|  |
| --- |
| <<var [Multi=SectionControls.FirstOrDefault(sc=>sc.tag ==”MSC”)]>> |

**Reference Variable Example**

|  |
| --- |
| Selection: <<[Multi.values.select.Replace(",",", ")]>>  Reason: <<[Multi.values.otherReason]>> |

### Formatted

#### Number

**Establish Variable Example**

|  |
| --- |
| <<var [Number=SectionControls.FirstOrDefault(sc=>sc.tag ==”NMBR”).values.controlValue]>> |

**Reference Variable Example**

|  |
| --- |
| Number: <<[Number]>> |

#### Date

**Establish Variable Example**

|  |
| --- |
| <<var [Date=SectionControls.FirstOrDefault(sc=>sc.tag ==”DTE”).values.controlValue]>> |

#### Email

**Establish Variable Example**

|  |
| --- |
| <<var [Email=SectionControls.FirstOrDefault(sc=>sc.tag ==”EML”).values.controlValue]>> |

**Reference Variable Example**

|  |
| --- |
| Email: <<[Email]>> |

#### URL

**Establish Variable Example**

|  |
| --- |
| <<var [URL=SectionControls.FirstOrDefault(sc=>sc.tag ==”URL”).values.controlValue]>> |

**Reference Variable Example**

|  |
| --- |
| URL: <<[URL]>> |

#### Phone

**Establish Variable Example**

|  |
| --- |
| <<var [Phone=SectionControls.FirstOrDefault(sc=>sc.tag ==”PHN”).values.controlValue]>> |

**Reformatting the input phone number:**

|  |
| --- |
| <<[Phone.Substring(0,3) + "-" + Phone.Substring(3,3) + "-" + Phone.Substring(6)]>> |

**Reference Variable Example**

|  |
| --- |
| Phone: <<[Phone]>> |

### Advanced

#### Name

**Establish Variable Example**

|  |
| --- |
| <<var [Name=SectionControls.FirstOrDefault(sc=>sc.tag ==”NME”)]>> |

**Reference Variable Example**

|  |
| --- |
| Full Name: <<[Name.values.fullNameValue]>>  Title: <<[Name.values.titleValue]>> |

#### Location

**Establish Variable Example**

|  |
| --- |
| <<var [Location=SectionControls.FirstOrDefault(sc=>sc.tag ==”LCTN”)]>>  If you want to separate out the Latitude and Longitude  <<var [lat=Location.values.mapCoord.Substring(0,ctrl.values.mapCoord.IndexOf(","))]>> <<var [lng=Location.values.mapCoord.Substring(ctrl.values.mapCoord.IndexOf(",")+1)]>> |

**Reference Variable Example**

|  |
| --- |
| Coordinates: <<[Location.values.mapCoord]>>  Latitude: <<[lat]>> Longitude: <<[lng]>> |

#### Address

**Reference Variable Example**

|  |
| --- |
| Line by line Address: <<[siteAddress]>>  1st Line of Address: <<[siteAddressCtrl.values.street]>>  2nd Line of Address: <<[siteAddressCtrl.values.street2]>>  Location Description: <<[siteAddressCtrl.values.description]>>  City: <<[siteAddressCtrl.values.locality]>>  State: <<[siteAddressCtrl.values.areaCode]>>  Zip: <<[siteAddressCtrl.values.postalCode]>>  County: <<[siteAddressCtrl.values.county]>>  Country: <<[siteAddressCtrl.values.countryCode]>> |

#### Attachment

**Establish Variable Example**

|  |
| --- |
| <<var [Attachment=SectionControls.FirstOrDefault(sc=>sc.tag ==”ATTCHMNT”)]>> |

**Reference Variable Example**

|  |
| --- |
| Comment: <<[Attachment.values.attachmentComment]>>  Confidential: <<[Attachment.values.confidential]>>  Confidential Reason: <<[Attachment.values.confidentialReason]>>  File ID: <<[Attachment.values.fileId]>>  File Name: <<[Attachment.values.fileName]>> |

#### Table

**Establish Variable Example**

|  |
| --- |
| var [Table=SectionControls.FirstOrDefault(sc=>sc.tag ==”TBLE”)] |

**Advanced Table Example**

|  |
| --- |
| The following example prints all values of the short text control. The tag for the control in this example is ”Short” while the tag for the advanced table itself is “ADV\_TABLE\_CONTROL”. The short text value is called on with “<<[shortValue]>>”.  <<var[advancedTable=SectionControls.FirstOrDefault(sc=>sc.tag=="ADV\_TABLE\_CONTROL")]>>  <<var[advancedTableRows =advancedTable.rows]>>  <<foreach [row in advancedTableRows]>><<var [shortColumn=row.columns.FirstOrDefault(c=>c.tag ==”SHORT”)]>><<var[shortValue =shortColumn== null || shortColumn.values ==null?””:shortColumn.values.controlValue]>>  <<[shortValue]>>  <</foreach>> |

#### Contact

**Establish Variable Example**

|  |
| --- |
| <<var [submitterCtrl=SectionControls.FirstOrDefault(sc=>sc.tag ==”CNTCT”)]>>  <<var [submitterInfo = submitterCtrl == null ? “”:”<br/>”+ submitterCtrl.values.companyNameContact+”<br/>”+ submitterCtrl.values.streetContact + “<br/>” + submitterCtrl.values.street2Contact + (submitterCtrl.values.street2Contact !=null?”<br/>”:””) + submitterCtrl.values.localityContact +”,” + submitterCtrl.values.areaCodeContact + “ “ + submitterCtrl.values.postalCodeContact] >> |

**Reference Variable Example**

|  |
| --- |
| Line by line Address: <<[submitterInfo]>>  Prefix: <<[submitterCtrl.values.prefixNames]>>  First Name: <<[submitterCtrl.values.firstNameContact]>>  Middle Name: <<[submitterCtrl.values.middleNameContact]>>  Last Name: <<[submitterCtrl.values.lastNameContact]>>  Title: <<[submitterCtrl.values.titleContact]>>  Company Name: <<[submitterCtrl.values.companyNameContact]>>  1st Phone: <<[submitterCtrl.values.phoneContact\_0]>> 2nd Phone: <<[submitterCtrl.values.phoneContact\_1]>> 3rd Phone: <<[submitterCtrl.values.phoneContact\_2]>> 4th Phone: <<[submitterCtrl.values.phoneContact\_3]>>Email: <<[submitterCtrl.values.emailContact]>>  1st Line of Address: <<[submitterCtrl.values.streetContact]>>  2nd Line of Address: <<[submitterCtrl.values.street2Contact]>>  Location Description: <<[submitterCtrl.values.descriptionContact]>>  City: <<[submitterCtrl.values.localityContact]>>  State: <<[submitterCtrl.values.areaCodeContact]>>  Zip: <<[submitterCtrl.values.postalCodeContact]>>  County: <<[submitterCtrl.values.countyContact]>>  Country: <<[submitterCtrl.values.countryCodeContact]>> |

#### Hidden Text

**Establish Variable Example**

|  |
| --- |
| <<var [Hidden=SectionControls.FirstOrDefault(sc=>sc.tag ==”HDDN”)]>> |

**Reference Variable Example**

|  |
| --- |
| Hidden: <<[Hidden.values.controlValue]>> |

#### Calculated

**Establish Variable Example**

|  |
| --- |
| <<var [Calculated=SectionControls.FirstOrDefault(sc=>sc.tag ==”CLCLTD”).values.controlValue]>> |

**Reference Variable Example**

|  |
| --- |
| Calculated: <<[Calculated]>> |

#### Repeaters

If references to data found within a form repeater are desired, consulting services will be required to perform advanced configuration of the document template. However, an example below could provide insight into the proper syntax depending on experience level:

|  |
| --- |
| This example prints values from the text field (highlighted in green) located within the repeater control (highlighted in blue).  <<var [repeatSection = submissionVersion.sections.FirstOrDefault(s=>s.tag  == "REPEATER" && s.repeatableParentSectionId == Null)]>>  <<var [Repeat = submissionVersion.sections.Where(s=>s.id == repeatSection.id || s.repeatableParentSectionId == repeatSection.id).OrderBy(s=>s.sortOrder)]>>  <<foreach [item in Repeat]>>  <<if [item.Controls.FirstOrDefault(sc =>sc.tag=="REPEAT\_TEXT")!= Null]>>  <<[item.Controls.FirstOrDefault(sc =>sc.tag=="REPEAT\_TEXT").values.controlValue]>>  <</if>>  <</foreach>> |

#### Markup

If you are uploading a document that is in review, be sure and turn off track changes before you save it to upload. Markup will be considered part of the document, and will impact rendering.