HighQ Collaborate
Document Automation Manual

Table of Contents

- Introduction
  - What is document automation?
  - How document automation works in Collaborate
    - Step 1 - Create the Basic Template
    - Step 2 - Create the Associated iSheet
    - Step 3 - Update the Template with iSheet References
    - Step 4 - Generate Document (Repeat)
  - Two Types of Templates
  - Access Rights Respected
- Template Building: General Guidelines
  - Docx Format
  - The Basics
  - Velocity Notation: Placeholders and Directives
    - Placeholders
    - Directives
  - Adding Velocity References to a Word Template and Editing Them
  - Formatting Velocity References
- Building the Template: Specifics
  - Referencing the Source of the Data
    - The Source iSheet Directive
    - The Source View
    - Referencing Columns
    - Grabbing the Data
    - Referencing iSheet Components
- Using Column Data in the Template
  - Site-Wide Template
  - Single-Record Template
  - Inserting Column Information to be Displayed
    - Formatting of Placeholders
  - Types of iSheet Columns that can be Inserted into a Template
    - Which iSheet Columns can be Inserted into a Template
    - Hyperlinks
    - System-Columns
    - Columns Without Values
    - Numbers
    - Dates
    - Text Conversion and Modification
  - Multiple Records and Tables

            - Foreach Loops
            - Tables
            - Footnotes are Not Completely Supported
            - Multi-Value Choice Columns
            - Loop Counter
            - Expanding and Collapsing Content
        - Conditions
            - Comparison Operators
            - Boolean Expressions
            - If-Elseif-Else Branches
            - Checking for Empty Values
            - Advanced Conditions
                - Ignore Case
                - Text Contains or Starts With a Value
                - String Length and Record Size
            - Conditions Involving Number Columns
            - Conditions Involving Dates
        - Helper Tools
            - Conversion Tools
                - Convert a Number Column to a Number Variable
                - Convert a Date Column to a Date Variable
            - Math Tools
                - Basic Math Arithmetic
                - More Complex Math Operations
                - Aggregation
                - Number Formatting
            - Date Tools
                - Reformat the Date
                - Compare Two Dates
                - Date Arithmetic
                - Date Components
                - Other Uses of Date Information
            - Sort Tools
        - Auto Numbering and Table of Contents
            - Hide the entire section if the condition evaluates to false
            - Include every section
    - Associating Document Templates with iSheets
        - Adding Document Templates
            - Site-Wide/Multi-iSheet Document Template
            - Single-Record iSheets
        - Managing Existing Document Templates
    - Generating Documents from iSheet Records and a Template
        - Site-Wide Templates

- Single-Record Templates

Debugging Velocity Templates

- Create the iSheet First
- Build Incrementally and Test Often
- Formatting
- Common Problems

Introduction

What is document automation?

Document automation is a technology that permits a document to be automatically generated from a template by inserting information into the template that is associated with a particular matter, project or transaction. Document automation can be thought of us as “mail merge on steroids,” but document automation differs from mail merge because the information may be gathered through an interview process and the template can utilize complex logic to determine which information to insert and the information can be formatted and modified as necessary.

Due to the work involved in building templates, document automation works best on standardized documents, ones that are used repeatedly, for different clients, matters or other situations. The document that is generated as a result of document automation can either reflect the final version of that document, or merely a starting point for further changes and negotiations.

Document automation is composed of two principal parts:

1. A document template, which contains the substance and structure of the document to be generated, along with any required formatting. Usually, the template is a Microsoft Word document.
2. The data to be used in generating a document from the template for a specific situation. Typically, the data will be obtained through an interview process, where questions are asked and the data to be used in the generated document is provided as answers to those questions.

How document automation works in Collaborate

There are four primary parts to document automation in Collaborate.

Step 1 - Create the Basic Template

First, a Business Analyst, working in concert with an attorney or PSL, will create a template for a specific legal scenario. This document template *must* be designed using Microsoft Word (docx or docm format), and initially can include simple placeholders for the data that will be included later (like “[Company Name]” for the company name). For example, a firm’s clients may need to enter into non-disclosure agreements (NDA’s) with other parties on a regular basis and may wish to use a standard NDA form. Based on the relationship between the parties and the type of other party, each NDA may be customized to include information like whether the NDA is unilateral or bilateral, how strict the language will be, and which jurisdiction’s laws will apply, in addition the basic information about each party being inserted into the generated NDA. The attorney would provide a draft of the NDA, with alternative language, and the Business Analyst would modify the template to include placeholders, conditions, etc..

Step 2 - Create the Associated iSheet

Second, the Business Analyst will identify the information that must be collected to enable a document to be generated automatically from the template. The template creation process will help the Business Analyst in this effort. Once a list of that information has been created, the Business Analyst, working alone or with someone trained in Collaborate, can convert that information into an iSheet form (or series of iSheet forms), which will later be used to collect the information from a user who needs to generate a new document.

Step 3 - Update the Template with iSheet References

Third, once the iSheet form has been designed and the document template created, the Business Analyst (or other trained user) will update the Word template to include references to the data columns that are available in the iSheet. (These references have a very specific format and are based on a technology called Velocity.) This part will involve testing and iteration. Once completed, the Word template will be added to the Collaborate site and, if necessary, associated with the iSheet(s) that had been created in Step 2.

Note that the same iSheet can be referenced in multiple templates. You are not limited to a single template per iSheet.

Step 4 - Generate Document (Repeat)

Finally, a user with appropriate access rights to that Collaborate site can generate a Word document based on the previously uploaded Word template, using information that was added to the associated iSheet(s) by that user or other users. The document automation "engine" will combine the template and the data to generate the document.

As you can see, there are two essential *technical* skills required to take advantage of document automation in Collaborate:

1. Building an iSheet to collect the information. Anyone can learn to do this with only 30 minutes of training. Note that only a Site Administrator on a given site will have access to the tools necessary to build an iSheet. Once such an iSheet has been created, it can be templated and reused in other sites.
2. Creating the Velocity Word template to merge the iSheet data into the Word document. This will likely require a few hours of training, at most.

Two Types of Templates

Collaborate's document automation technology supports two different, high-level scenarios:

1. Site-Wide. The site-wide template may include content from any or all records from any or all iSheets in a site. The most common example is the Due Diligence context, where information about a large number of documents may have been entered by multiple users into one or more iSheets in a single site. A report generated from this information will be global in scope and may include references to many iSheet records in several different iSheets and views. Because any iSheet record in a site can be included, only a Site or Content Admin can generate documents from those templates.

2. Single Record. The single-record template includes content from a *single* record of a *single* iSheet. The most common example is an NDA, where each record in an “NDA Information” iSheet contains all of the information that is necessary to generate a single NDA. Only a user with Edit or View rights to an iSheet record can generate a document from the information in that record.[1]

Access Rights Respected

Regardless of which type of template is used, document automation will always respect the iSheet record-level and column-level access rights granted to the person generating the document.

The rest of this document focuses on how to build a Collaborate document automation template in Microsoft Word using the Velocity notation. This document assumes that the user has a working knowledge of the Collaborate platform, with particularly strong knowledge of the iSheets module.

Template Building: General Guidelines

A starting point for a document template is the document itself. Simply create a document like you would any other legal document. (It’s always easiest to start with a document from an existing matter or transaction, or to use an existing form from a firm’s form library.) The substance of the document should written by a domain expert. This manual assumes that such a template already exists. For demonstration purposes, the “NDA Template.docx”, single-record template, is an example of a non-disclosure agreement that might have been prepared by an attorney, with underlined spaces representing placeholders. This can be found in the Document Automation Training site.

Docx Format

Templates *must* be built using Microsoft Word 2007 or above, and the template files must be saved in the .docx or .docm format. Older versions of Word, which use the .doc format, are not supported. The document that is generated through the document automation process will always have the .docx format.

The Basics

In order to merge information from an iSheet into the generated document, the Word template that is created must contain placeholders for that information. For example, using a generic notation (not Velocity notation), in a template business letter, the salutation would look like this:

Dear [FirstName],

where “[FirstName]” is a placeholder, and the value inserted in place of [FirstName] when the document is generated could come from an iSheet column called “First name”. While there *may*

[1] In a future release, users with View-only rights to an iSheet record will also be able to generate documents from single-record templates.

be additional complexity beyond the notion of a simple placeholder, that is essentially how document automation in Collaborate works. The actual Velocity placeholders you will use will have a slightly different format, like this, highlighted in grey:

Dear «$record.FirstName»,

with the precise placeholder format and how to add them described below.

Velocity Notation: Placeholders and Directives

As noted above, the references to the iSheet data that are added to a template are based on a technology called Velocity. There are two types of Velocity references that will be used in a template: placeholders and directives, each of which uses a specific format.

Placeholders

A *placeholder* tells the document automation engine what iSheet data to insert (for example, insert the last name field, not the first name field) and where in the document to put the data.

A placeholder typically includes a reference to a variable, which holds the value to be inserted. A variable can be given almost any name, but the name should always be meaningful, so you know what the variable represents. When building templates, variables must always start with a $, then a letter, and after that can include any letter or number, but not spaces or most punctuation, except for _ and -. For example, these are permitted variable names:

- $DateOfBirth
- $Party1_Of_5

These are not permitted:

- $1stParty
- $First Name

Directives

A *directive* provides guidance to the template about how to use the data. For example, it may indicate the underlying source of the data, or it may be used to add conditions to the template, such checking to see whether the $Country variable equals “United States,” in which case show one clause, otherwise show a different clause. A directive will usually start with a #.

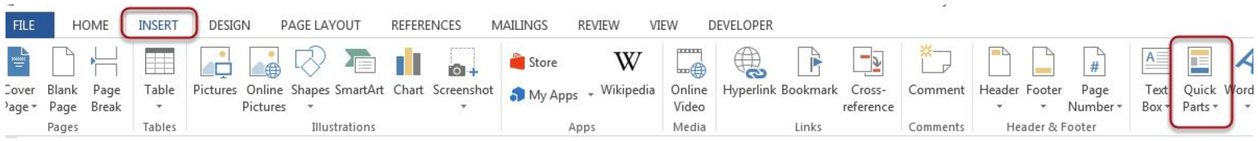
Adding Velocity References to a Word Template and Editing Them

When you need to include either type of Velocity reference to a template, these must to be added to the Word document in a special way, not simply by typing them directly into the document. Indeed, if you try to type these references directly into Word, or edit an existing references directly, those references and changes will be *ignored* by the document automation engine.

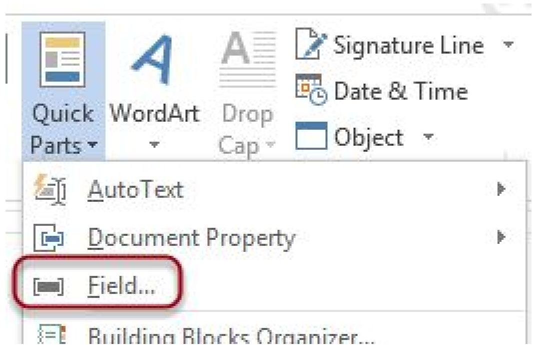
First, create a MergeField in the Word document by following these steps:

1. Go to the location in the Word document where the reference should be added.

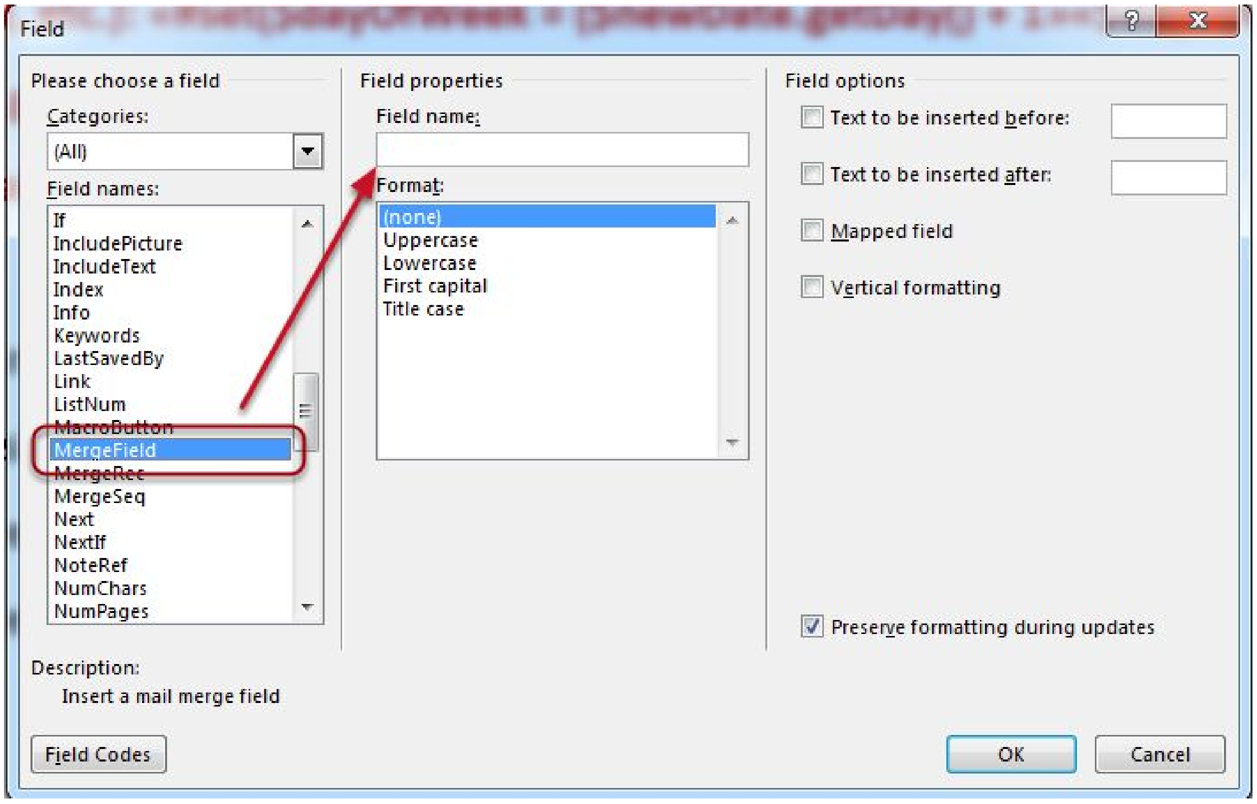
2. Select the “Insert” tab. (The precise manner for selecting the “Insert” tab may change depending on the version of Word you are using. The screenshot below is for Word 2013.)
3. Click on “Quick Parts”.



4. Select “Field...” from the pull-down menu:



5. On the “Field” popup window, scroll down the list of Field names on the left and select “MergeField”:



6. In the “Field name:” field in the middle of the page, enter the Velocity reference.
7. Hit “OK”.

There is a shortcut as well:

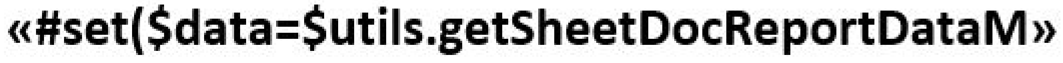
1. Hit CTRL-F9. This will create a set of curly brackets: { }
2. Right click inside the curly brackets and select “Edit Field...”.

That will bring up the “Field” popup window as shown above (#5), then follow the rest of the directions from there.

Once you enter a Velocity reference into the MergeField and hit “OK”, the reference will be inserted into the document with special brackets («») around it. For example, if you had entered “$record.FirstName” it would appear like this in the Word document:

«$record.FirstName»

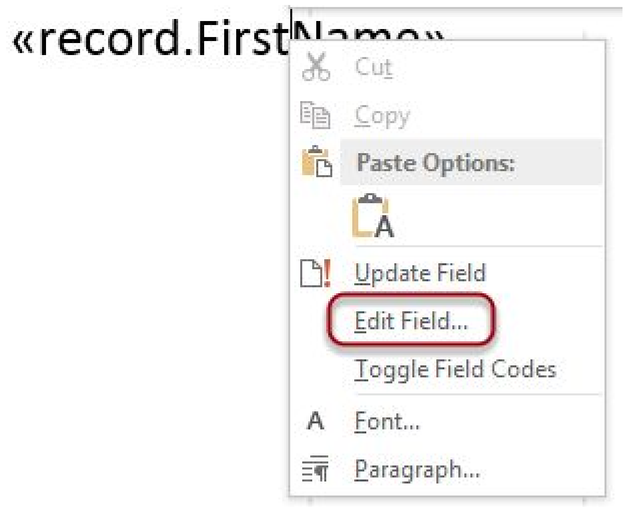
Note that if the text entered into the MergeField is longer than 40 characters, in the Word document you will only see the first 40 characters:



«#set($data=$utils.getSheetDocReportDataM»

(In this manual, when examples of Velocity references are shown, the *full* reference will be displayed, even if only the first 40 characters would typically be shown in Word.)

To view the full Velocity reference and to later edit that reference, right click anywhere in the MergeField and select the “Edit Field…” option:



That will open up the “Field” window shown above (#5), where you can scroll to see the full reference. As noted above, do not attempt to edit the reference directly from the main Word screen.

There is an alternative way to display all of the MergeFields in a template, which allows you to view the full text of the MergeFields from directly inside the Word document, especially those that are longer than 40 characters. It also allows you to more easily search and find the text inside a MergeField.

Simply hit ALT-F9 and the MergeFields will be changed to look like this:

{ MERGEFIELD #set($sheetTitle=\"Velocity1\") * MERGEFORMAT }

The Velocity reference will appear between“{MERGEFIELD” and “* MERGEFORMAT}”. Any double quotes in the reference will have a backslash escape character in front of it.

Although the MergeFields can now be edited directly from inside Word, this is not recommended. First, you would need to make sure to keep the surrounding text and characters, and to put a backslash in front of any double quotes. Also, if you edit a MergeField this way, the underlying reference in the MergeField will be edited, but the text that you see in the Word document when you toggle back to the regular view will be unchanged and will need to be edited directly in the Word document from the regular view. Accordingly, use this approach for editing Velocity references sparingly.

Hit ALT-F9 to toggle back to the regular view.

Formatting Velocity References

When a document is generated from the template, one of two things will happen to the Velocity references that have been inserted into the template:

- Placeholders will be *replaced* with the actual data.
- Directives will be *removed* entirely. This means that the format of the template may look off, but when the document is generated the format will be normal.

As a result, for placeholders, you need to embed them into the template in the *exact* location they should appear in the generated document, and with appropriate formatting.

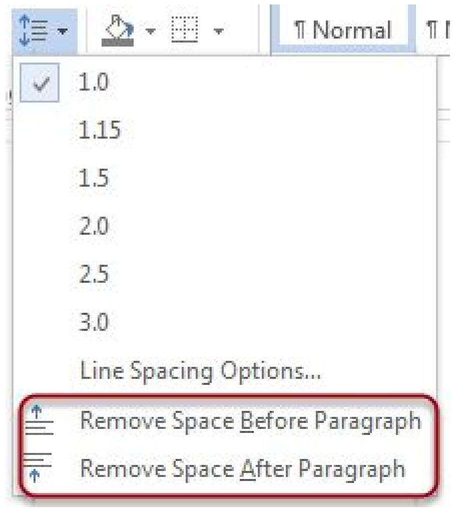
For directives, you should keep them as inconspicuous as possible. If you enter a directive on its own line, when the document is generated, the directive itself will be removed, but any line breaks associated with the directive will remain behind. Accordingly, when adding directives it is recommend that you do the following:

- If you have a group of them, put them all on the same line, with only one line break at the end, with no spaces between each directive.
- IF the directives will appear on their own line, put the directives in a much smaller font than the rest of the text, such as 4 point, otherwise they can be the same size as the regular text. That way, any line breaks that do remain will have less impact on the document’s formatting. For example:

  «#set($sheetTitle='*Employees*')»«#set($viewName='*Current Employees*')»«#set($columns={})»«$!{columns.put("FirstName","First Name")}»«$!{columns.put("LastName","Last Name")}»«$!{columns.put("DOB","Date of Birth")}»«#set($data=$utils.getSheetDocReportDataMap($sheetTitle,$viewName,$columns))»

You can leave the directives in normal size font during the template building phase, then make them smaller once the template logic has been finalized.

- Avoid any unnecessary line breaks. Indeed, many directives can be added to the *same* line as text that will be visible when a document is generated.
- Remove the spaces before and after the paragraph that contains the Velocity directives, using common Word features, to eliminate unnecessary spaces:



- Highlight all directives in a color that makes them standout on the page, but still allows the text to be readable. That way, the directives are easy to find for creation and debugging purpose.

«#set($sheetTitle='*Employees*')»«#set($viewName='*Current Employees*')»«#set($columns={})»«$!{columns.put("FirstName","First Name")}»«$!{columns.put("LastName","Last Name")}»«$!{columns.put("DOB","Date of Birth")}»«#set($data=$utils.getSheetDocReportDataMap($sheetTitle,$viewName,$columns))»

Because the directives will be removed when the page is generated, this color will not be included in the generated document.

There is a file called "NDA Template with Velocity References.docx," which takes the same NDA template referenced above, but inserts Velocity references. (This can be found in the Document Automation Training site.) These references get their data from the "NDA Information" iSheet found in this site, which you should have access to. (Indeed, feel free to go ahead and add an iSheet item and generate an NDA.) The directives, which will be removed when the document is generated, have been highlighted in aqua in the template to make them stand out for review.

Building the Template: Specifics

The section above provided general template-building guidelines. This section walks you through the specifics.

When you create a template, you need to tell the template the source of the data from which it will generate a document. Therefore, you need to tell the template which iSheet(s) it is associated with and also which iSheet view(s).

Referencing the Source of the Data

Use Velocity references to tell the template where the data is coming from: the source iSheet and, typically, the source view of that iSheet. Although it is recommended that these references be included at the top of the document, these reference *must* be added *before* the location of the first placeholder.

The Source iSheet Directive

First, tell the template the name of the source iSheet, by inserting the following Velocity directive:

«#set($sheetTitle="*Name of iSheet*")»

The actual iSheet name will be added in place of *Name of iSheet*, but it still must be included between the set of double quotes. This tells the template that the data will be coming from a particular iSheet. We know that $sheetTitle is a variable because it starts with a $. Note that whenever you need to put text in a Velocity reference in quotes, surround the text in *double* quotes, like: "Name of iSheet", instead of single quotes.

If the template includes data from multiple iSheets, then this type of directive will need to be added for each iSheet, but each iSheet needs a unique variable name, like:

«#set($sheetTitleEmployees="*Employees*")»
«#set($sheetTitleDepartment="*Department*")»[2]

In that case, the variables are $sheetTitleEmployees and $sheetTitleDepartment.

The Source View

Next, indicate which view of the selected iSheet will be the source of the data. The selected view will restrict which records in the iSheet are available in the template. For example, use the "USA Stores" view if the document should only include stores located in the United States. To reference a particular view, next add this reference to the template:

«#set($viewName="*Name of View*")»

with the variable name of $viewName. Again, replace *Name of View* with the actual name of the view. If you need to include multiple views in a template, simply repeat that step for each view, giving each variable a unique name.

[2] The creation of each variable does not need to be together in this way in the template. Each can be defined in different sections of the template.

Referencing Columns

Now that we have identified the iSheet and view to be used as the source of the data, we need to list all of the columns of data in the selected iSheet that *may* be used in the template. (Any columns that will not be used in any way in the template can be ignored.) First, we need to create a "container" that will hold the list of columns. To do so, insert the following directive:

```
«#set($columns={})»
```

where $columns is a variable that will hold the list of columns.

Next, we need to assign each column its own variable name, by using this format:

«$!{columns.put("*variableName*","*Column Name*")}»

for example:

```
«$!{columns.put("FirstName","First name")}»
```

where *FirstName* is the name of the variable that will be used to reference the "First name" column. (In this particular case, do NOT add the $ to front of the variable name.) Note that the column name that is entered here must match the actual column name in the iSheet EXACTLY. The first value will be a variable name (and subject to restrictions noted above on naming variables), and the second value is the precise column name.

The variable name should be meaningful, so that at a glance you can quickly recognize what it stands for. That means use "FirstName" rather than "Variable1," for example. Also, for ease of readability, using so-called CamelCase may be the best approach, where each distinct word in the variable name has initial caps, like: "FirstName". Remember, spaces are not allowed in variable names.

For every column in that iSheet that may be used in the template, add the directive shown above.

Note that every variable name that is associated with the same iSheet must be unique. You *cannot* reuse the same column variable name for two different columns, like this:

```
«$!{columns.put("Name","First name")}»
«$!{columns.put("Name","Last name")}»
```

Grabbing the Data

Finally, there is another step to actually obtain the data from the iSheet and make it available to the template. Insert this directive:

```
«#set($data=$utils.getSheetDocReportDataMap( $sheetTitle,$viewName,$columns))»
```

Each italicized word is a variable, and should have the names you have actually given to each of the three variables: the iSheet title, the view name and the list of columns. $*data* is the variable that holds all of the data for every record in the referenced iSheet and view.

Referencing iSheet Components

Note that if the name of the iSheet, the view or a column changes in Collaborate, then the existing Velocity references in the template will no longer work, as they will reference something that no longer exists. In that case, the template will need to be corrected to match the new names given to those components in Collaborate.

Using Column Data in the Template

Wherever the actual data from the iSheet will be inserted into the template, take one of the following steps.

Site-Wide Template

If the template is accessing more than one record from an iSheet or records from multiple iSheets, then you will need to create a loop that will loop through every record in the selected view of the iSheet, with the following two directives. The first directive (foreach) creates the loop and the second one ends the loop.

```
«#foreach($record in $data)»
«#end»
```

Between these two directives you will include any Word document text and any placeholder references that are necessary to generate the document, as shown below.   The "Model Multi-Record Document Automation Template.docx," template available here provides an example of a multi-record template and uses these directives. $data is the variable created to hold the data brought in from the iSheet.

Single-Record Template

If you have a single-record template, where you only need to access a single record in the iSheet, there are two choices. First, you can include the loop statement above, and the loop will only process the single record. In that case, the «#foreach» directive should appear at the top of the template and the «#end» directive near the bottom.

The alternative is include the following directive before you need to access the record:

```
«#set($record=$data.values().iterator().next())»
```

The “NDA Template with Velocity References.docx,” template available here provides an example of a single-record template and uses this directive. (Note that if for some reason the $data element was empty, this directive might cause the template to fail, which would be acceptable, as there would be no data to populate the template.)

In both cases, site-wide or single record, the $record variable holds all of the data for a *single* iSheet record. Each record will likely contain information about multiple iSheet columns, like First name, Last name and Date of Birth, each of which will have had a Velocity variable associated with it, like $FirstName, $LastName and $DOB, respectively.

Inserting Column Information to be Displayed

In the example above, once you have defined all of the columns and created the $record variable, in order to insert a particular column of data into the template, simply add a placeholder like this to the template:

«$record.FirstName»

When the document is generated from the template, the First name field from that record of the iSheet will be inserted in place of «$record.FirstName». For example, for the salutation portion of a letter, the Velocity template would look like this:

Dear «$record.FirstName»,

and in the generated document you would see:

Dear Mary,

Note that when you reference any variable, you must reference the variable name exactly as it was defined, including the case of letter. For example, the variable $FirstName is not the same as variable $firstName.

Formatting of Placeholders

Any formatting that is applied to a placeholder will be applied to the text that is included in the generated document in its place. For example, the placeholder reference shown below is red, bolded and uses 24 point font:

«$record.FirstName»

As a result, the outputted text that appears in the generated documents would look like this:

Mary

If the placeholder is included among other text and bolded, like:

> The lease expires on **«$record.LeaseExpirationDate»**, after which date the company will need to find a new location.

Then the replacement text (in bold) will simply be included in the middle of the sentence as well:

> The lease expires on **30 Jun 2016**, after which date the company will need to find a new location.

Therefore, apply the appropriate formatting to the placeholder, as well as the correct placement and spacing between the placeholder and other text on the page.

Types of iSheet Columns that can be Inserted into a Template

Every iSheet column that can be included in a template will be outputted to the generated document as *regular* text.

Which iSheet Columns can be Inserted into a Template

Most but not all types of iSheet columns can be inserted into a template. If you try to use a column that cannot be inserted, an empty string will simply be displayed.

Here is a list of the supported column types:

- Single line text
- Multiple line text -- Note that if a multiple line text field is set to use rich text, the template will *ignore* any rich text formatting that is included.
- Choice -- If there are multiple choice values entered in a column, the choices will be presented in the template as a *single* value, with each choice separated by a comma, like: red,white,blue
- Number
- Date and Time -- Note that there are four different date formats, each of which can optionally include the time as well. The format in which a date column is displayed in a template will match the format of the date field in the iSheet.
- User lookup -- The user lookup field can have three different formats: the name of the user, the email address of the user, or the user's name and their organisation. The format in which a user lookup column is displayed in a template will match the format of the field in the iSheet.
- Hyperlink -- the display name of the hyperlink column will be outputted, *not* the URL (if no display name was provided in the hyperlink column, then the URL will double as the display name)
- Calculation
- Auto increment

The unsupported column types are:

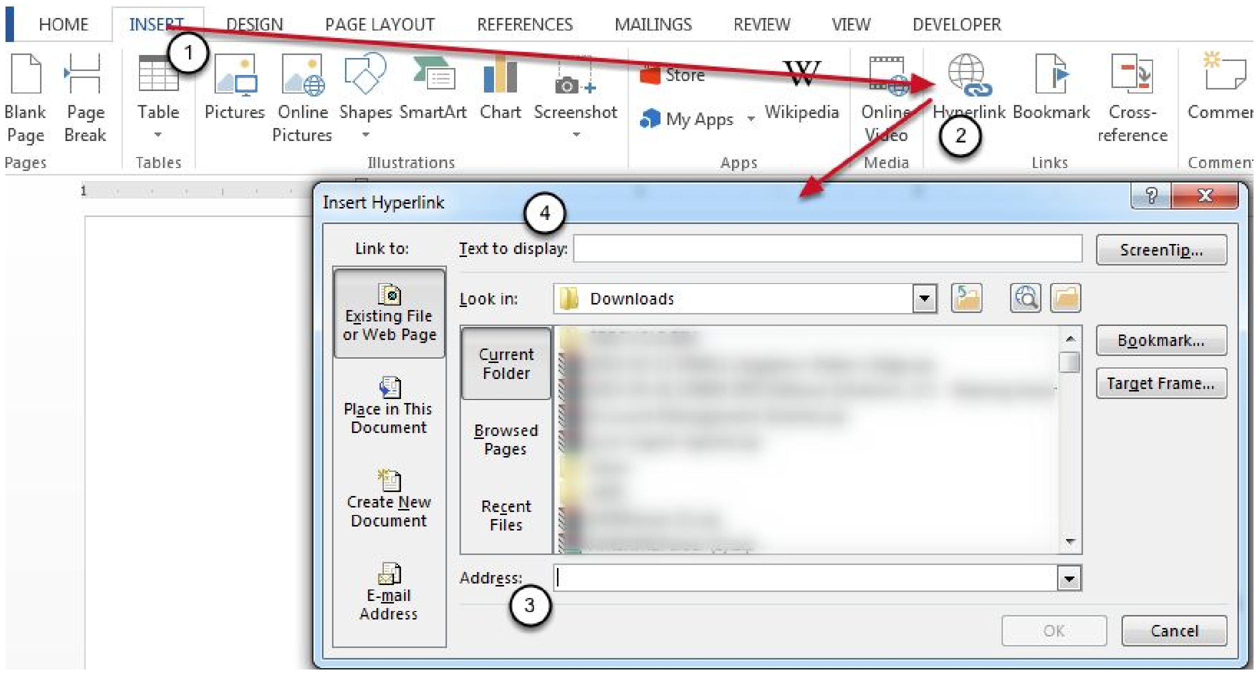
- Image -- both types
- Lookup -- both basic and advanced
- Join
- File Link
- Folder Link
- Attachment

Hyperlinks

Theoretically, it is possible to take a URL that has been added to a column (such as to a single line text column, a hyperlink column or from a choice column) and turn it into a Word hyperlink, so if a user clicks on the link, it will open in a browser window. (Note that if a hyperlink column is used as the source of the data, only the display name portion of the hyperlink is available to the template. In a hyperlink column, if the display name is left blank, Collaborate uses the URL portion of the hyperlink as the display name. Therefore, leave the display name blank when entering hyperlink values, so that the value available to the template is the URL.) The URL must include the "http://" or "https://" prefix for the link to work in Word.

Here is how a Word hyperlink that includes a URL and/or display name from iSheet data can be added to a template:

1. In Word, select Insert. You can also select some text in the document that will become the text of the link.
2. Then select Hyperlink, and you will see the "Insert Hyperlink" window:



3. In the "Address" field, enter the Velocity placeholder for the variable that contains the URL, like: $record.Link

4. In the ”Text to display” field, the value will default to the value you entered in the Address field. This display name will be used for every link, so if your template is looping through multiple records, this is the display name that will be outputted for each one. You have three choices of values to use here:
    a. If you wish for the URL itself to appear here, then leave this field as it is and it will take that value automatically.
    b. Alternatively, enter static text, which will appear the same every time a hyperlink is inserted into the document.
    c. Enter another variable name, like $record.CompanyName, or highlight that variable to begin with when the link was inserted.

   After hitting “OK,” the document will simply show linked text with the value that was entered in the “Text to display” field, like:

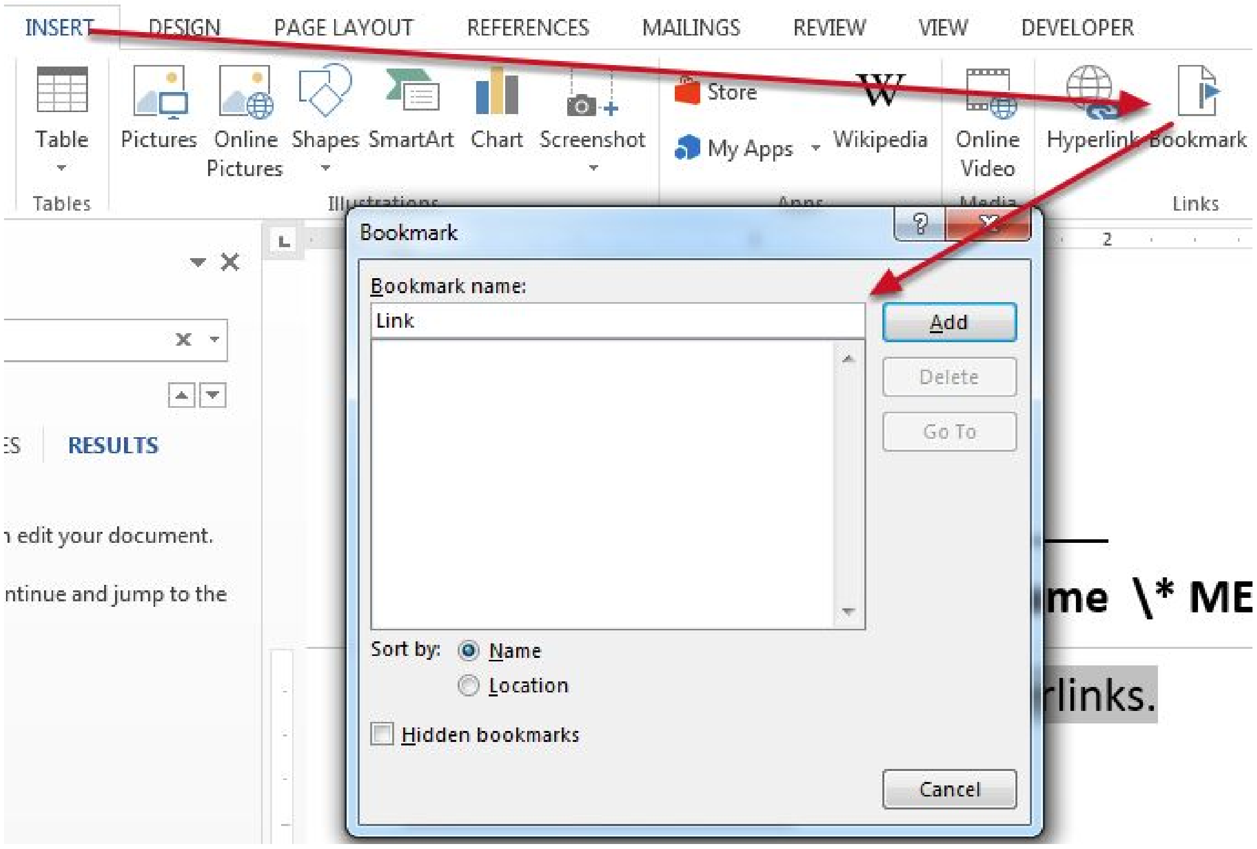
   $record.CompanyName

5. Next, hit ALT-F9 and you will see the code that is being used for the link itself:

   ```
   { HYPERLINK "$record.Link" }
   ```

6. Position the cursor immediately after the second double quote (before the closing bracket). Then go to Insert → Bookmark and insert a bookmark. Give the bookmark any

name:



Bookmark

Bookmark name: Link

Sort by: [✓ Name] Location

Hidden bookmarks

7. Finally, hit ALT-F9 again to toggle the codes off.

(If the document will contain several hyperlinks like this, then each one must have a unique bookmark.)

If you want to create a mailto link, in the situations where you have a column where an email address is entered, manually edit this text to include the word “mailto:” inside the quotes:

```
{ HYPERLINK "mailto:$record.EmailAddress" }
```

Then hit ALT-F9 to toggle the code view off.

System-Columns

Note that the four system columns that appear in every iSheet can also be referenced in a template:

1. Created by -- a user column
2. Created date -- a date column
3. Modified by -- a user column
4. Modified date -- a date column

Just like for regular user and date columns, the display format of each of these columns can be modified at the iSheet level.

Columns Without Values

If a column in a particular row does NOT contain a value, in the template that column will have a value equal to an empty string. Therefore, it is recommended that certain columns in an iSheet be mandatory, so they cannot be empty. In a condition, an empty value can be tested by comparing the variable to the empty string, like this:

```
«#if($record.FirstName == “”)»
```

Conditions are discussed in more detail below.

Numbers

A number column will simply be outputted as that number, so:

```
«$record.FirstName» is currently «$record.Age» years old.
```

becomes:

```
John is currently 18 years old.
```

There are also helper methods that can be used to format numbers, such as adding commas as thousands separators, currency symbols, etc.

Note that it is possible to work with number columns in more complex ways, such as in calculations and comparisons, as discussed below.

Dates

The format that a date will be displayed in is based on the date format of the corresponding date column in the iSheet. Excluding date fields that include time, there are four supported date formats:

- DD MMM YYYY, like 08 Nov 2016
- DD/MM/YYYY, like: 08/11/2016
- MM/DD/YYYY, like: 11/08/2016
- DD.MM.YYYY, like 08.11.2016

If time has been included in the column configuration, then that will be added in the format of HH:MM (24 hour time), like: 21:18.

As discussed in more detail below, it is possible to utilize dates in more complex ways, such as by comparing two dates, doing date arithmetic (like calculating an age given a date of birth), pulling out parts of a date (like month or year) and displaying a date in a different format from the four listed above, like: November 8, 2016 @ 9:18pm.

Text Conversion and Modification

In addition to the formatting changes you can make by applying Word styles (such as bold, underline, etc.), you can also change the text of an inserted value in numerous ways, using functions, which are similar to Excel formulas. For example:

- **Upper case**. To convert a value to upper case, simply do the following: «$record.FirstName.*toUpperCase*()»
- **Lower case**: «$record.FirstName.*toLowerCase*()»
- **Remove white space**. Although typically not needed, you can remove the white space on both sides of a value using this format: «$record.FirstName.*trim*()»
- **Substring**. You can extract any part of a column value, such as the first 3 characters, the middle 8 characters, or the last 2 characters. This also requires knowledge about the length of the value and whether the value is empty or not. If you are certain about the length of a value, then apply the substring() function by simply passing in two parameters: the location of the character *before* the part you want to extract, and location of the *last* character you want to extract. For example, if the value is: “08 Jun 2014” and you want to pull out the first two characters (characters 1 and 2, or “08”), use this format: «$record.Date.*substring(0,2)*». To obtain “Jun” from the same value, use: «$record.Date.*substring(3,6)*».

Note that these functions must be used in the *precise* way shown above.

Multiple Records and Tables

Foreach Loops

In multiple-record templates, you will need to loop through every record and then display the data from each of those records. As shown briefly above, you must first add a foreach loop to insert the value from each record:

> «#foreach($record in $data)»This document is entitled *«$record.ContractTitle»* and was entered into on **«$record.ContractDate»**.
> «#end»

In the example above, if there were three documents listed in the iSheet view, then the output in the generated document would be:

> This document is entitled *Master Lease* and was entered into on **23/05/2012**.
> This document is entitled *Employment Agreement* and was entered into on **15/06/2010**.
> This document is entitled *Indenture* and was entered into on **02/12/2011**.

Notice how the formatting applied to the placeholders is preserved, and a line break is added after each sentence. Also, the foreach directive is placed on the same line as the sentence, to reduce the number of line breaks in the generated document

Tables

When you are dealing with multiple records in an iSheet, often the best approach is to output the data in a Word table, with each record in its own row of the table. In that case, start by creating a table with two rows. The first row will contain the column headers and will likely have its own formatting. The second row will contain the placeholders for the data. Put a special form of the foreach loop in the first column of the data row, as shown below, with the data in that column between the start and the end of the loop:

«@before-row#foreach($record in $data)»[Insert any data]«@after-row#end»

For example:

First Name	**Last Name**	**DOB**
«@before-row#foreach($record in $data)»*«$record.FirstName»* «@after-row#end»	**«$record.LastName»**	«$record.DOB»

When this template run is run, a single table will be generated with all of the records, with different colors in each column, like this:

First Name	**Last Name**	**DOB**
John	**Smith**	23/11/1960
Mary	**Jones**	14/03/1971
Tom	**Warren**	03/07/1946

Indeed, there is a special way to handle tables so that the data rows will have alternating background colors, to make the table easier to read. You can configure the table in Word to show alternating row colors for every row but the header row. This requires using Word table styles. Such a table might look like this:

First Name	**Last Name**	**DOB**
John	**Smith**	23/11/1960
Mary	**Jones**	14/03/1971
Tom	**Warren**	03/07/1946

Note that same approach can be used to include or exclude rows in a table. For example, if every row in a table include a different iSheet column (instead of a different iSheet record), based on the value of a column, you may not want to include a row for that column, as follows:

Column Name	Column Value
First Name	«$record.FirstName»
«@before-row#if($record.MiddleName != “”)»Middle Name«@after-row#end»	«$record.MiddleName»
Last Name	«$record.LastName»

This checks to see if the “Middle Name” column contains a value and, if it does, then include the row.

Footnotes are Not Completely Supported

Sometimes it is necessary to add a footnote for each record of multiple records, and for those footnotes to contain information about each record. In that case, however, simply inserting the required variable in a footnote will *not* create footnotes with the information from each record. Instead, a more indirect method will need to be used to display this information in the footnotes.

For example, the iSheet might contain a list of employees, and the footnote might indicate the year that each employee started working. This can be done as follows. First, before the loop, create a container (here, $startYears) that will hold the information to be included in the footnote, as well as a counter variable to hold the number of footnotes ($fnCounter), in case not every record will have a footnote generated. Then, as you loop through each record, add the start year information for each record with a footnote: $!{startYears.put($fnCounter,$record.StartYear)}. Finally, reference that container in the footnote and pull out the appropriate record, using the $velocityCount variable. In this case, whether the footnote is displayed can also be put inside a condition, such as only showing the footnoted information for “Platinum”-level employees:

> «#set($startYears={})»«#set($fnCounter=0)»
> «#foreach($record in $data)»The employee’s name is: «$record.EmployeeName»«#if($record.isPlatinumEmployee == “true”)»«#set($fnCounter=$fnCounter+1)»«$!{startYears.put($fnCounter,$record.StartYear)}»[1]«#end».
> «#end»
>
> --------
> [1]This employee started working for the company in «$startYears.get($velocityCount)».

Multi-Value Choice Columns

Choice columns can contain multiple values. For example, if the column is called “Favorite Colors”, the value entered in a record may be red, blue and green. In that case, if you simply try to insert the value of the column using the common placeholder format: «$record.FavoriteColors», it will insert the values in a comma-delimited format, like this:

red,blue,green

Instead, you can choose to display each choice on its own line, or in a separate row of a table, by doing something like the following, where you will split up the entire value on the comma character (using the split() function), loop through each choice, and assign each choice the variable $color:

```
«#foreach($color in $record.FavoriteColors.split(","))»«$color»
«#end»
```

which would be displayed as:

```
red
blue
green
```

Note that if a choice value itself contains a comma, then this approach will not work. For example, if the "Favorite Cities" choice field contains values like: "New York, NY", "Paris, France" and "London, England", the commas in those choice values will cause the split() function will create 6 values:

- New York
- NY
- Paris
- France
- London
- England

Loop Counter

If you are using a foreach loop to loop through every record, the template provides a built-in counter variable, $velocityCount, which starts the count at 1. Therefore, if you wanted to number every record in your generated document, you could do that as follows:

```
«#foreach($record in $data»«$velocityCount». «$record.Name» -- «$record.DOB»
«#end»
```

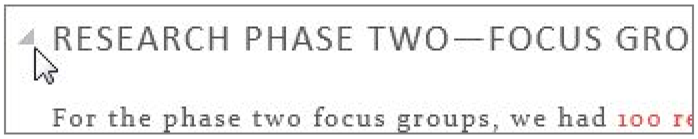
Which would be outputted as:

1. John Smith -- **05/13/1976**
2. Mary Jones -- **11/12/1984**
3. Tom Thompson -- **23/08/2001**

Notice that a period (.) had to be added after the reference to «$velocityCount».

Expanding and Collapsing Content

Microsoft introduced a new feature in Word 2013 that permits styles (like Heading 1) to be used to expand and collapse the text in a DOCX file. Very simply, when a style heading is used for a line of text, any text that comes directly after the heading that is not styled or that has a lower priority style can be hidden by clicking on an arrow next to the heading:


RESEARCH PHASE TWO—FOCUS GRO

For the phase two focus groups, we had 100 re

These headings can also be collapsed by default when the document is opened, so that the arrow must be clicked to expand and view the text below, as shown in the article linked to above.

This technique can be leveraged using document automation, where both the heading text and the text that comes after can be provided dynamically from column data. The only thing to be careful about is ensuring that there is intervening text in the document so that the expand/collapse behavior is stopped when you want the regular text display to start up again in the rest of the document. To make this work, simply do the following (assuming there are multiple records), with notes highlighted in yellow:

> «#foreach($record in $data)»
> «$record.ContractTitle»   Apply the style Heading 1 to this line and then set it to be collapsed by default.
> «$record.ContractSummary»   Make this regular body text.
> «#end»
> Add a line here that has Heading 1 applied, but do not add any text to this line and keep it expanded by default.

Conditions

Perhaps the most powerful way to use data from a record is in a condition. Conditions permit you, among other things, to include or exclude certain language from the document based on the data from the iSheet.

For example, imagine the document template will be used to generate leases based on information in the “Leases” iSheet. There is a column in that iSheet called “Lease renewal” which indicates whether the lessee has the right to renew the lease by giving notice to the landlord. This “Lease renewal” column has a value of either “Yes” or “No”. That column will be mapped to a variable called $LeaseRenewal in the template:

> «$!{columns.put("LeaseRenewal","Lease renewal")}»

In the lease that will be generated using document automation, if Lease Renewal = “Yes,” then an additional clause will be inserted: “Upon giving at least ninety (90) days notice in advance of the Expiration Date, the TENANT may renew this Lease for an additional one (1) year period." This is how that condition would be represented in the template, using Velocity directives:

> Two (2) months security deposit will be required upon signing the Lease. «#if($record.LeaseRenewal == "Yes")»Upon giving at least ninety (90) days notice in advance of the Expiration Date, the TENANT may renew this Lease for an additional one (1) year period. «#end»All Notices must be sent...

(Note that the equals comparator use the *double* equals sign: ==.) If the condition evaluates to true, then any text in the template that is *between* the start of the condition («#if») and its close («#end») will be included in the generated document. If the condition evaluates to false, that text will be excluded. Also notice above that the condition contains not just the text, but white space at the end of the included sentence, to be ensure that the spacing is correct with the text that follow.

The general format of a condition is:

> «#if($record.*ColumnVariableName* == "*value*")»

Comparison Operators

All of the common comparators are available in conditions: equals, greater than, less than, greater than or equal to, less than or equal to and not equals, using these symbols:

```
==  (note that equals uses the DOUBLE equals sign)
>
<
>=
<=
!=
```

When comparing a column value against a static text value, make sure to wrap the static text value in double quotes:

> «#if($record.Country == "Italy")»

Boolean Expressions

It is possible to combine multiple expressions together, which allows you to check the value of multiple columns in a single record at one time, by using the AND and OR operators, and also using parentheses to group conditions together.

The values that will be checked in any condition are those that exist in a *single* record of the iSheet. For example, to display a clause in a document only when (a) a lease is renewable and the lease location is the United States, OR (b) the lease is not renewable and the lease location is Italy, use the following condition:

```
«#if(  ($record.LeaseRenewal == "Yes" && $record.LeaseLocation == "United States") ||
($record.LeaseRenewal == "No" && $record.LeaseLocation == "Italy")  )»
```

The AND operator is entered as && (two ampersands). The OR operator is entered as || (two pipe characters). You can also apply the NOT operator, the ! (exclamation point before the condition). Make sure to properly group complex conditions using parentheses.

If-Elseif-Else Branches

If there are multiple, *exclusive* conditions for the same record that should be checked, use the if-elseIf-else format (always ending with the "end" expression), which can contain as many elseif statements as are required. For example, this will insert a different clause if the country is Italy, or if it is Denmark, or if it is Bolivia, or else a different clause for every other country.

```
«#if($record.Country == "Italy")»
  [insert Italy clause]
«#elseif($record.Country == "Denmark")»
  [insert Denmark clause]
«#elseif($record.Country == "Bolivia")»
  [insert Bolivia clause]
«#else»
  [insert clause for all other countries]
«#end»
```

Checking for Empty Values

If you need to determine whether *any* value was entered for a given column, write the condition in this format:

```
«#if($record.LeaseRenewal == "")»
```

That conditions checks whether the value equals the empty string, or two double quotes together. As noted above, many iSheet columns should be mandatory, so that they cannot be empty.

Note that if (a) column-level security has been applied to a column in an iSheet, (b) that column is referenced in the template and (c) the user running the template does not have at least view access to that column, then that column will not be available *at all* when that user generates a document. In that case, if the template had tried to output the column's value, then the generated Word document will simply show the variable name instead, like: "$record.LastName."

Generally, we advise that you should not include a column in a template if column-level security is applied to that column and a user who will have rights to generate a document does not have at least view access to the column. But in those cases where it is necessary, it is advised that the template first check to see whether the column in fact is available for the user running the template and, if not, have the template respond appropriately. To test to see whether a column is available, simply use this if condition format:

```
«#if($variableName")»
```

like:

```
«#if($record.LeaseRenewal)»
```

That means: "If this column exists…"

Advanced Conditions

There are several advanced conditions that can be applied to a template.

Ignore Case

All text matches are case sensitive. Therefore, if the condition is if($Country == "USA") and the country was entered in the column as "usa", the condition will evaluate to false. But sometimes you need to determine whether if the condition is true regardless of case. In that situation, use this format:

```
«#if( $record.Country.equalsIgnoreCase("Italy") )»
```

by putting the text to be matched inside the equalsIgnoreCase() function.

Text Contains or Starts With a Value

If you need to determine whether the value that was entered for a column contains certain words or characters that is only part of the entire value, you can use a condition in this format:

```
«#if($record.Country.contains("United"))»
```

That will match "United States of America" and "United Kingdom.

Similarly, to see whether the value *starts* with a certain characters, use this:

```
«#if($record.Country.startsWith("Italy"))»
```

Note that both contains() and startsWith() are case sensitive in their matching. To check whether a column contains a certain word regardless of case, use this format, first converting the value to lower case, then seeing whether it contains a lowercased value:

```
«#if($record.Country.toLowerCase().contains(“united”))»
```

As noted above, a multi-value choice column will be outputted as a comma-delimited value. To see whether a particular choice was selected in a record, use the contains() format:

```
«#if($record.FavoriteColors.contains(“blue”))»
```

In that case, make sure one value is not a substring of another. For example, if there are three choices available in the field:

- blue
- blue-green
- slate blue

and you attempt to check if the record contains(“blue”), it will evaluate to true if *any* of those 3 values were selected. There is more complex logic that can be applied to see if a value *exactly* equals a particular choice, which would look like this:

```
«#set($arrays = $utils.getClass().forName("java.util.Arrays"))»
«#set($choices = $record.FavoriteColors.split(","))»
«#if($arrays.asList($choices).contains("blue"))»
        [Text to be displayed]
«#end»
```

As always, if the choice values themselves contain the comma character, neither of these methods will work.

String Length and Record Size

Both in the case of conditions and also for display purposes, you can determine the length of a value that has been inputted and the number of records in the selected view of the iSheet. For example, this condition is true if the length of a column value is more than 6:

```
«#if($record.Name.length() > 6)»
```

This condition is true if there are more than 12 records in the iSheet view made available to the template:

```
«#if($data.size() > 12)»
```

Alternatively, if you wanted to include a statement in the template indicating how many records there were, you could do this, with the size() function:

This report covers the «$data.size()» documents that require further review by the client.

Conditions Involving Number Columns

If a column is a number field, calculated field or a text field that contains numbers (like a choice field), those values will always be treated as text by default in the template. In order to treat a column value as a number -- such as when you need to compare a number in a condition -- the text value must be turned into a “proper” number.

This can be done for any type of number, whether a whole number or one with a decimal. Whenever you need to compare a column value as a number (like: [Age] > 10), do the following, using the special $conversionTool helper variable, discussed in more detail in the next section.

```
$conversionTool.toNumber($record.ColumnName)
```

For example, if you need to compare a column to a number for a condition, do this:

```
«#if($conversionTool.toNumber($record.Age) > 10)»
```

or to compare two columns, both must be converted:

```
«#if($conversionTool.toNumber($record.Age) >
$conversionTool.toNumber($record.MaximumAge))»
```

In addition to conditions, number values can also be used to calculate aggregates and other results, as shown in more detail in the next section.

Conditions Involving Dates

Similar to working with numbers, to do anything with a date column besides display it, the date column must first be converted from text to a date. The next section shows in more detail how to convert a date value into a date variable so that it can be used in a condition (comparing it to other dates) and for other purposes.

Helper Tools

[*Note that this section covers advanced topics and can be ignored for building basic templates.*]

There are a number of “tools” that have been made available to the template which can help you work with column data from an iSheet, particularly numeric and date columns.

Note that if the value of a column in a record is empty, then many of these tools will not work and may in fact cause the document not to be generated. Therefore, you should only want to use these tools on columns with a required value or the template should check to make sure there is a value in the column before using these tools on a particular record.

Conversion Tools

Every iSheet column, no matter what type, is treated as regular text in the template. For example, if you have a number and want to compare it to another number, or perform a calculation on that number, you cannot do so directly. Instead, you first need to convert the "text" number to a real number. The same holds true for date columns. The Conversion Tool can convert these text values to the proper type. (This same type of issue is often encountered in Excel, where Excel treats a number or date as text and Excel formulas will not work until the values have been converted to the proper type.) The Conversion Tool can create a *new* variable from the existing text column variable, like a date or number variable, and the new variable can be used in different ways.

Convert a Number Column to a Number Variable

Using the Conversion Tool, you can convert a number variable to either a whole number or a number with a decimal point. This shows you the Velocity directives you need to include in the template to make this happen:

- Whole numbers: «#set($integer = $conversionTool.toInteger($record.Year))»
- Numbers with decimals: «#set($double = $conversionTool.toDouble($record.Pi))»
- If you are unsure what type of number you are working with, use the toNumber() method: «#set($number = $conversionTool.toNumber($record.SomeNumber))»

if you use one of the whole number utility methods on a number that contains a decimal, the decimal will simply be removed from the variable that is created.

Once you have created a proper number variable, you can perform operations on the number (comparison, arithmetic, etc.), as demonstrated further below.

Convert a Date Column to a Date Variable

Date columns are more complicated to work with than numbers, because you first need to know the format of the date column before you can create a proper date variable from the "text" date value. Date columns can utilize one of four formats, and may include a time component. Those four date formats are:

- DD MMM YYYY, like 08 Nov 2016
- DD/MM/YYYY, like: 08/11/2016
- MM/DD/YYYY, like: 11/08/2016
- DD.MM.YYYY, like 08.11.2016

If the time component has been included in the configuration of the date column, then the time will be added in the format of HH:MM (24 hour time), like: 21:18.

To create a date variable ($date in the example below) from a date column value, use the parseDate() method, as follows:

«#set($date = $conversionTool.parseDate($record.DateOfBirth,”*DateFormat*”)»

The parseDate() method takes two arguments:

- First, a reference to the date column being converted, like $record.DateOfBirth
- Second, the format of that date column, using a special notation described below. The date format argument would look like this for one of the four formats: “dd/MM/yyyy”, which would match a date like this: 12/31/2014. Note that if the wrong date format is used, the proper date variable will not be created.

The special notation for converting date values to date variables uses the following characters to represent each part of the date:

- Day in a month
  - d = a day in a month, with one or two digits, like 8 or 27
  - dd = a day in a month, with two digits, like 08 or 27
- Month
  - M = a month with one or two digits, like 1 or 12
  - MM = a month with two digits, like 01 or 12
  - MMM = a month as an abbreviated word, like Oct
  - MMMMM = a month as a full word, like October
- Year
  - yy = a two-digit year, like 14
  - yyyy = a four-digit year, like 2014
- Hour
  - H = hour in 24-hour clock (0-23), with one or two digits, like 1 or 22 -- note that in Collaborate, time is always shown with a 24-hour clock
  - HH = hour in 24-hour clock (0-23), with two digits, like 01 or 22
  - h = hour in 12-hour clock, with one or two digits, like 1 or 12
  - hh = hour in 12-hour clock, with two digits, like 01 or 12
  - a = AM/PM marker
- Minute
  - mm = minutes in an hour, with two digits, like 05 or 46
- Other characters -- any other characters included in the date format will be shown as themselves, like the colon character, or a comma.

For example, for each of the four date column formats, use the following date format in the parseDate() method:

- “dd MMM yyyy” for “09 Oct 2014”
- “dd/MM/yyyy” for “31/12/2014”
- “MM/dd/yyyy” for “12/31/2014”
- “dd.MM.yyyy” for “31.12.2014”

If the time element is included, just add “HH:mm” to the end of the relevant format, like:

“dd MMM yyyy HH:mm” for “09 Oct 2014 15:32”

This is how it would look, putting it all together:

```
«#set($date = $conversionTool.parseDate($record.DateOfBirth,”dd MMM yyyy
HH:mm”)»
```

Once the new date variable has been created, other tools can be used to display that date variable in a different format, to compare two dates, or perform date arithmetic, as described below.

Math Tools

Once you have your number variable, you can perform operations on it. There are some special utility methods, but you can also use basic operators, like: +, -, *, / and % (modulus).

Basic Math Arithmetic

If you simply need to perform an operation on two numbers, you can use the Math Tools variable to display the result or create a new variable to hold the result. For example:

- Add: «$mathTool.add($record.Number1, $record.Number2)»
- Subtract (the second number from the first): «$mathTool.sub($record.Number1, $record.Number2)»
- Multiply: «$mathTool.mul($record.Number1, $record.Number2)»
- Divide (the first number by the second): «$mathTool.div($record.Number1, $record.Number2)»

Use them like this assign the result to a variable:

```
«#set($result = $mathTool.add($record.Number1,$record.Number2”)»
```

Indeed, there is no need to convert the column variables to number variables to use any of the $mathTool methods.

Alternatively, if you first convert a number column value to a number variable, you can apply the regular math operators (+, -, etc.). This is shown below in one of the “manual” aggregation examples, where all of the values in a column of an iSheet are totalled.

More Complex Math Operations

For more complex math operations, use the $mathTool helper variable. Note that you do NOT need to convert text columns variables to numbers first for these to work.

- Floor -- to obtain the integer portion of a number: «$mathTool.floor($number)»
- Ceiling -- to round the number up to the nearest integer: «$mathTool.ceil($number)»

- Round -- to round the number to the nearest integer: «$mathTool.round($number)»
- Absolute Value -- to obtain the absolute values of a number: «$mathTool.abs($number)»
- Power -- to raise a number to a power: «$mathTool.power($number, *power*)» where *power* is a number, like 2 or 3.

Aggregation

It is possible to perform aggregation on a set of data. There are three approaches you can take:

1. The so-called "manual" approach
2. Word table formulas
3. Using Math Tools

"Manual" Approach. The "manual" approach where you loop through every record, grab the column you need, convert it to to a number and increment a total.

For example, if you are tracking penalties assessed against an opposing party, to sum all of the penalties, you could do something like this, assuming the "Penalty amount" column -- a number column -- corresponds to the $record.PenaltyAmount variable. (Note that all numbers in the iSheet must be whole numbers for this to work.) Comments are shown after each line in brackets.

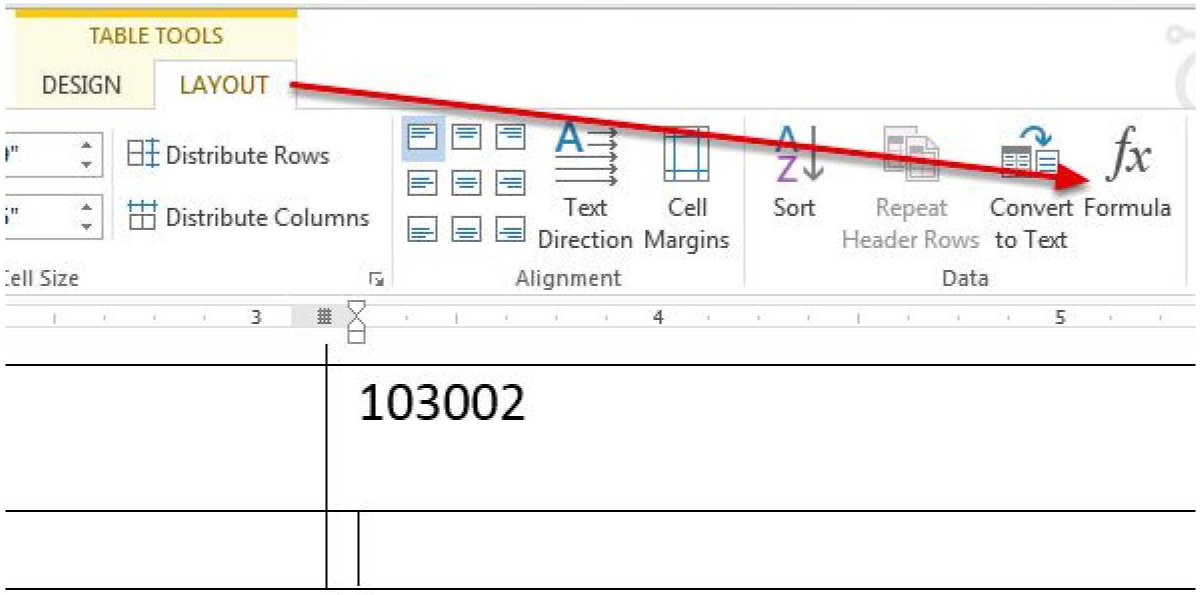
```
«#set($Sum = 0)»  [Create a number variable to hold the sum]
«#foreach($record in $data)»  [Loop through every record in the iSheet]
        «#if($record.PenaltyAmount != "")»  [Check to see if the penalty amount in this
record is empty or not; ignore it if the amount is empty]
                «#set($PenaltyAmountNumber =
$conversionTool.toInteger($record.PenaltyAmount))»  [Convert each penalty amount to
a number variable]
                «#set($Sum = $Sum + $PenaltyAmountNumber)» [Increase the $Sum by
the amount of the penalty in each record]
        «#end»
«#end»
The total penalties are: $«$Sum»  [Output the total]
```

(Note that a similar example is included in the Model Multi-Record Document Automation Template .docx file, referenced elsewhere.)

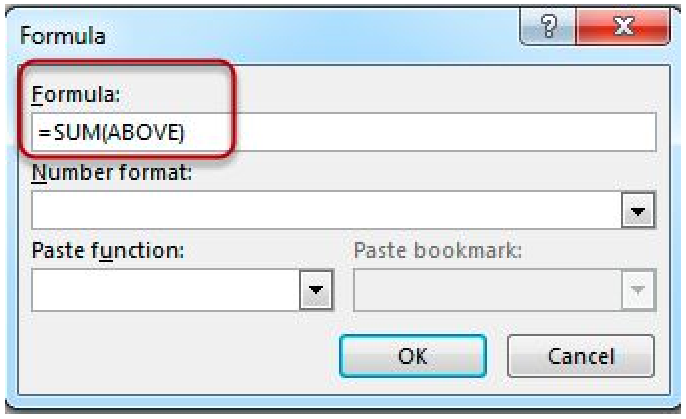
Calculations Using Word Table Formulas. Word has a built-in feature that will perform calculations on number values that appear in a table, such as adding all of the rows in a given column. If you are using document automation to generate a table that contains numeric information, this technique can be used, but this works *only* if the iSheet data has been inserted into a table. This approach is much easier to implement than using Velocity variables (shown above), but the Word table formula approach is also more limited.

Here are the steps necessary to add a table formula to the template:

1. In the template, add a row *beneath* the data row in the table where you want to sum data. This new row will contain the calculation.
2. In the column where you want to perform the calculation, enter the formula. There are more details about how to do this here, but simply place the cursor where you want to insert the formula, select Table Tools → Layout → Formula:



3. In the popup window, enter "=SUM(ABOVE)" in the "Formula:" field:


Formula

Formula:
=SUM(ABOVE)

Number format:

Paste function:

Paste bookmark:

OK
Cancel

do *not* enter a "Number format," and then hit "OK". (Note that there are more advanced formulas that can be used than "=SUM(ABOVE)".)

The benefits of this approach are that it requires no Velocity coding and tracking of variables, and you can sum any values that appear in a table column, not just values that were in the original iSheet.

This approach has a few shortcomings:

1. Formulas need to be updated. When the document is generated, every formula will have a value of zero. The formulas need to be updated after the document has been generated. There are two ways to do this:[3]
    a. Manually trigger an update, by hitting CTRL-A (select all) and then F9. You will need to educate the end user to take this step.
    b. Insert a macro into the document that will automatically update all formulas and fields when the document opens. Although this approach is easier for end users, the document will contain a macro, which is considered a security risk and many users will not be permitted to receive or open such files.[4]
2. Table formulas are very fragile. At the first blank or non-number row that the formula comes across as it moves up from the bottom, the formula will stop calculating the total. Therefore, make sure there are no empty values in the column, or replace empty values with zero using a condition like: «#if($record.Sales == “”)»0«#else»«$record.Sales»«#end»
3. Formatting cannot be applied. Do not use formatting when you insert the formula (see the screenshot above). Unfortunately, the document automation engine cannot handle formatting and if formatting is added, the document will not be properly generated and cannot be opened.

Math Tools approach. The alternative is to use the $mathTool helper variable.

- Total: $mathTool.getTotal($data.values(), “*ColumnName*”) -- to get the sum of all of the numbers in a given column. The first argument is typically the $data object, calling its values() method. The second argument is the variable name assigned to the column.
- Average: $mathTool.getAverage($data.values(), “*ColumnName*”) -- to get the average of all of the numbers in a given column.

These methods can be used to simply output a calculated number, or you can assign the calculation to a variable, like:

«#set($total = $mathTool.getTotal($data.values(), “Salary”))»

[3] Note that either of these two approaches for updating formulas will also update the pagination of any table of contents in the generated document.
[4] The VBA code for this macro is quite simple (and can also be found here):

```
Sub AutoOpen()
  Dim aStory As Range
  Dim aField As Field
  For Each aStory In ActiveDocument.StoryRanges
    For Each aField In aStory.Fields
      aField.Update
    Next aField
  Next aStory
End Sub
```

Number Formatting

It is possible to format numbers to include things like comma separators for thousands, currency and the number of decimals. To do, a number format must be applied, much like a date format is used to format dates. Simply use the format() method in your template, which takes two arguments, the number format and the variable to be formatted. A good practice is to create a variable for this purpose at the top of the template, like:

```
«#set($NumberFormatter  = “”)»
```

and use its format() method to format numbers. Then you can re-use the $NumberFormatter variable anytime you need to display a formatted number value. Note that the format() method can be applied *only* to number variables that have been created by using $conversionTool.toInteger() and similar methods. For example, below $number is a variable created already, whereas in the second example $conversionTool.toInteger() is used to convert the column to a number:

«$NumberFormatter.format(“*NumberFormat*”,$number)»

or

«$NumberFormatter.format(“*NumberFormat*”,$conversionTool.toInteger($record.Revenue))»

The number format that must be entered as the first argument. For example, to show the amount in dollars with a comma as the thousands separate, just this format: “$%,d”, like

```
«$NumberFormatter.format(“$%,d”,$record.Revenue)»
```

Here is what each part of the number format does:

- %d simply tells the engine that a signed (positive or negative) integer will be formatted.
- $ is simply a character that will be added before the formatted number. That could be any character, like £.
- The comma must be between the % and d characters, and tells the engine to add a comma as the thousands separator.

There are other formatting notations that can be used:

- Use “%f” to indicate that a floating point number will be used (a number with a decimal)
- Use “%#f” or “%#d” to indicate the fixed width of the number to be formatted, where # represents a number. For example, “%5d” will pass extra spaces to any number that is shorter than 5 characters long, like: “   32”. This cannot be used to truncate values that are larger than the fixed width amount.

- Use “%.#f” to indicate the precision after the decimal point, like “%.3f” will display 3.14159 as 3.141, with only three precision points.
- Use “%0#d” to indicate the precision for the number of integer characters, or fill them with zeroes. For example, “%05d” will display this: “00124”.

Date Tools

Once you have converted a variable representing a Date column to a date variable, there are several things you can do with the date using the $dateTools variable.

Reformat the Date

To display a date column value in a different format, first you need to create a date variable from it, as described above. Then, using that date variable, you can use the format() method of the $dateTool:

«$dateTool.format(”*DateFormat*”, $date)»

where “*DateFormat*” uses the date format notation shown above. For example, to show a date that was originally in this format: “12/31/2014 15:32” as “December 31, 2014 @ 3:32pm”, simply do this:

«$dateTool.format(“MMMMM d, yyyy @ h:mm a”, $date1)»

Compare Two Dates

Once a date column has been converted into a date variable, you can compare it with another date, to be used in a condition. The two most common use cases are to compare the date with today’s date or to compare it against another date column, which has also been converted to a date variable. For example, if there are two date variables, $date1 and $date2, these could be compared using the before() and after() methods. (For comparison purposes, note that the $today variable includes a time component and a date variable may not.)
To see whether $date1 is *before* today’s date, do the following:

«#if($date.before($today))»

To see whether $date1 is *after* $date2, do the following:

«#if($date1.after($date2))»

Date Arithmetic

Date arithmetic is a bit more complex. The most common scenarios are to determine the number of years between two dates or the number of days. There are two parts to this.

1. First, convert each date variable to a consistent number (the number of milliseconds between that date and January 1, 1970), using the getTime() method.
2. Second, subtract the number of milliseconds in one date variable from the other.

3. Third, divide the difference between the two dates by the appropriate time component (like day or year), by converting between the number of milliseconds and the time component.

For example, to get the number of days between two dates, do the following:

```
«#set($difference = ($date2.getTime() - $date1.getTime())/(24 * 60 * 60 * 1000))»
```

(To convert a quantity of milliseconds to a quantity of days, divide the quantity of milliseconds by the product of:

- 1000 milliseconds per second,
- 60 seconds per minute,
- 60 minutes per hour, and
- 24 hours per day

)

To display the number of years between two dates:

```
«#set($difference = ($date2.getTime() - $date1.getTime())/(365 * 24 * 60 * 60 *
1000.0))»
```

(This uses the same calculation as for days, except the denominator must be multiplied by 365, the number of days in a year. 1000.0 (with a decimal) is used in the formula so that the result includes decimal places, like 3.2 years, which is not necessary when calculating days.)

Note that in order to perform this type of calculation, the result of the calculation must be assigned to a variable (the variable $difference is used above). The the variable can be displayed:

```
«$difference»
```

Often the arithmetic will involve today's date, by using the $today variable.

Date arithmetic can be used simply to output the difference into the template, as shown above, or you can assign the difference to a variable and use that variable in conditions and other ways, such as if you wanted to compute a person's age and then output some text based on whether they are over 18 or not. For example:

```
«#set($dateOfBirth = $conversionTool.parseDate($record.DateOfBirth,”dd MMM
yyyy”))»

«#set($age = ($today.getTime() - $dateOfBirth.getTime())/(365 * 24 * 60 * 60 * 1000.0))»
```

«#if($age >= 18)»You can vote.«#else»You cannot vote.«#end»

Date Components

To display a single component from a date variable, like the day of the month, month (number) or year, use the following approaches:

- Day of month: «$date.getDate()»
- Month: «#set($month = ($date.getMonth() + 1))»
- Year: «#set(year = ($date.getYear() + 1900))»
- Day of Week (1 = Sunday, etc.): «#set($dayOfWeek = $date.getDay() + 1))»

Other Uses of Date Information

There are also more complex ways in which dates could be used, although that would require the assistance of a HighQ consultant. For example, group and count a date column by year, and output it like this:

Year	**Count**
2011	45
2012	32
2013	62

(Note that similar examples are included in the Model Multi-Record Document Automation Template.docx file found in the Document Automation Training site.)

Sort Tools

Sometimes the records that come from a multi-record iSheet are not in the order you would like them to be displayed in a template. The Sort Tools allow you to sort the records on any text columns, but not on number or date columns. Simply use this approach in the foreach loop, listing the column variable name as the second argument of the sort() method below. It is necessary to first create a variable ($records) that will hold the sorted values, which can then be used in the foreach loop.

«#set($records = $sortTool.sort($data.values(), "*ColumnName*”))»
«#foreach($record in $records)»

Indeed, it is even possible to sort on multiple columns, using this format:

«#set($records = $sortTool.sort($data.values(), ["*ColumnName1*”,”*ColumnName2*”]))»
«#foreach($record in $records)»

like:

«#set($records = $sortTool.sort($data.values(), ["LastName”,”FirstName”]))»

```
«#foreach($record in $records)»
```

You can even sort by descending (ascending is the default), by adding ":desc" or ":asc" after the column variable name:

```
«#set($records = $sortTool.sort($data.values(), ["LastName:desc","FirstName:asc"]))»
«#foreach($record in $records)»
```

Note that sorting is case sensitive, so words that start with lower case letters will come after words that start with upper case letters.

Auto Numbering and Table of Contents

Documents will often use automatic numbering for sections and clauses, using Word Headings and similar features. (These headings are often used as the basis for a table of contents.) If there are sections that should only be included if a condition evaluates to true, then have the condition *surround* the section, including the autonumbering component. However, if the section should always appear so that the number of the sections is not changed, the condition should appear inside the section, possibly with alternate text.

Here are the two approaches:

Hide the entire section if the condition evaluates to false

An example template fragment:

1. Scope of Agreement

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec porta ligula a arcu facilisis eleifend. Sed molestie massa eu libero tempor, vel faucibus dui tincidunt. Sed et convallis nunc.
«#if($record.LatinAmerica == "Yes")»

2. Latin America Agreement

Cras sit amet ante at libero fermentum tristique. Quisque vitae turpis at augue vulputate semper et tristique felis. Maecenas sagittis risus purus. Morbi tempus ultricies malesuada. Fusce quis arcu hendrerit, dictum arcu eu, ullamcorper neque. Aliquam libero nibh, suscipit sit amet suscipit ut, feugiat a metus. Aliquam a imperdiet odio.
«#end»

3. Term and Termination

Aliquam id accumsan neque. Donec luctus lacus turpis, sed congue magna bibendum sed. Curabitur sit amet leo mauris. Phasellus sed suscipit magna, quis pharetra erat.

If the condition evaluates to true, the contract will contain clauses 1 (Scope of Agreement), 2 (Latin America Agreement) and 3 (Term and Termination). If the condition evaluates to false, the contract will contain only clauses 1 (Scope of Agreement) and 2 (Term and Termination).

Include every section

An example template fragment:

> 1. Scope of Agreement
>
> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec porta ligula a arcu facilisis eleifend. Sed molestie massa eu libero tempor, vel faucibus dui tincidunt. Sed et convallis nunc.
>
> 2. «#if($record.LatinAmerica == "Yes")»Latin America Agreement
>
> Cras sit amet ante at libero fermentum tristique. Quisque vitae turpis at augue vulputate semper et tristique felis. Maecenas sagittis risus purus. Morbi tempus ultricies malesuada. Fusce quis arcu hendrerit, dictum arcu eu, ullamcorper neque. Aliquam libero nibh, suscipit sit amet suscipit ut, feugiat a metus. Aliquam a imperdiet odio.«#else»[Placeholder]«#end»
>
> 3. Term and Termination
>
> Aliquam id accumsan neque. Donec luctus lacus turpis, sed congue magna bibendum sed. Curabitur sit amet leo mauris. Phasellus sed suscipit magna, quis pharetra erat.

In this case, all three sections will always appear, but section 2 may include a full clause or merely the word "[Placeholder]," depending on how the condition evaluates.

Remember to manually rerun the table of contents (if any) after a document has been generated.

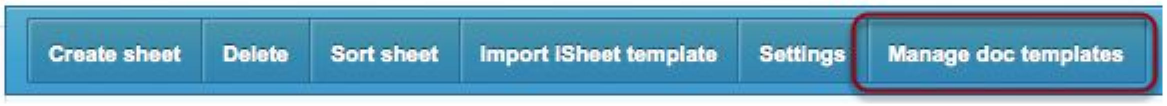
Associating Document Templates with iSheets

Once you have created a document template, how do you associate it with an iSheet so that it can be tested and then used to generate documents. As noted above, there are two principal types of document templates. Below we demonstrate how to associate the different types of templates with iSheets. Note that only a Site Admin (or System Admin) can actually add a document template to a site.

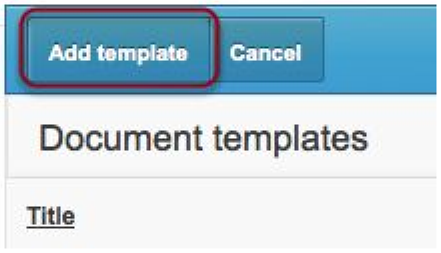
Adding Document Templates

Site-Wide/Multi-iSheet Document Template

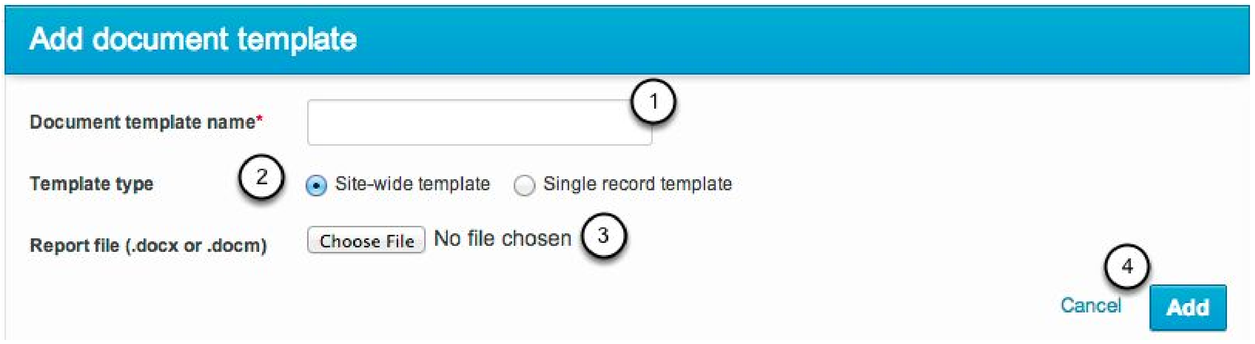
To add a template that is site-wide in scope, a Site Admin should access the iSheets Admin page of that site. At the top of the page, the Site Admin would click on “Manage doc templates”:



which will present a screen with an “Add template” button:



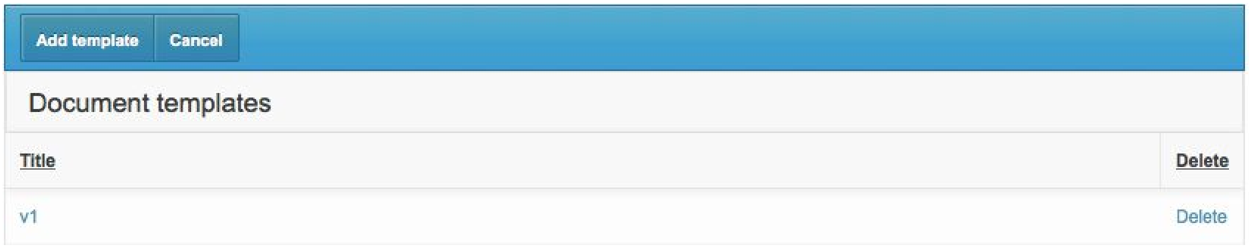
with no templates currently listed. Click on “Add template” to see the “Add document template” page:



There are four steps involved in adding a template:

1. Enter a meaningful name for the template. This is the name that will be displayed to end users. Ideally, the name should be unique, but different templates with the same name can be entered in the same site
2. Determine the type of template. For a site-wide/multi-iSheet template, select “Site-wide template”.
3. Browse to find the .docx or .docm document template file on your computer.
4. Hit “Add”.

After clicking “Add,” the “Document templates” page will refresh and list the template that was just added:



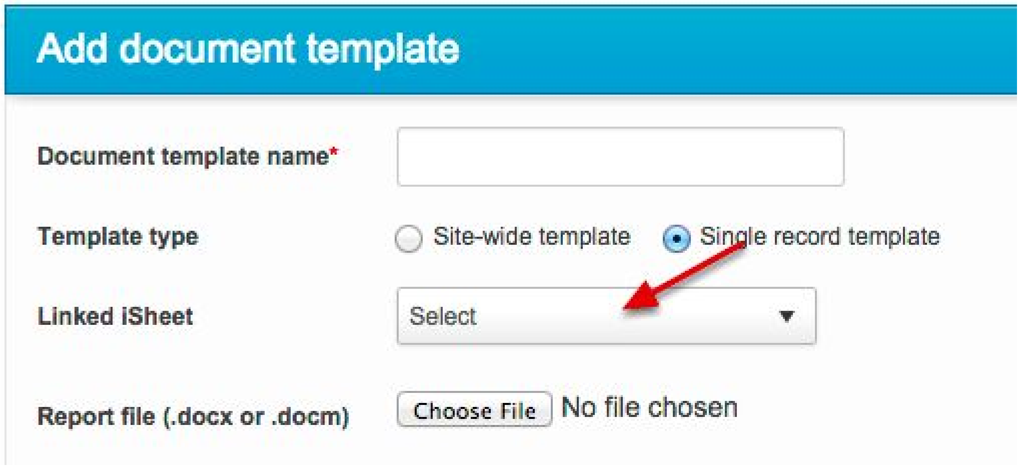
Document templates

Title	Delete
v1	Delete

That new template can now be accessed by a Site or Content Admin from any iSheet in that site.

Single-Record iSheets

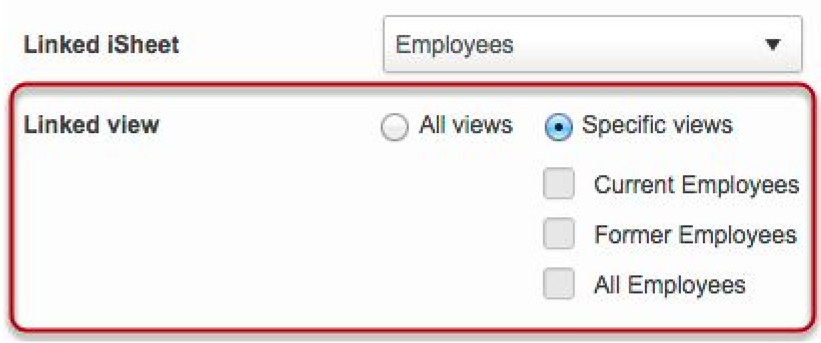
Adding a single-record iSheet is done in the same way, with one difference. Select the template type of “Single record template”, and in that case another field will appear, listing all of the iSheets in the site:



Add document template

Document template name*	
Template type	Site-wide template [✓ Single record template]
Linked iSheet	Select
Report file (.docx or .docm)	No file chosen

Once the iSheet has been selected, you will have a choice to associate that template with all of the views in that iSheet (including any views that are added in the future) or just specific views:



Linked iSheet	Employees
Linked view	All views [✓ Specific views]
	Current Employees
	Former Employees
	All Employees

(When limiting the template to specific views, at least one view must be selected.)

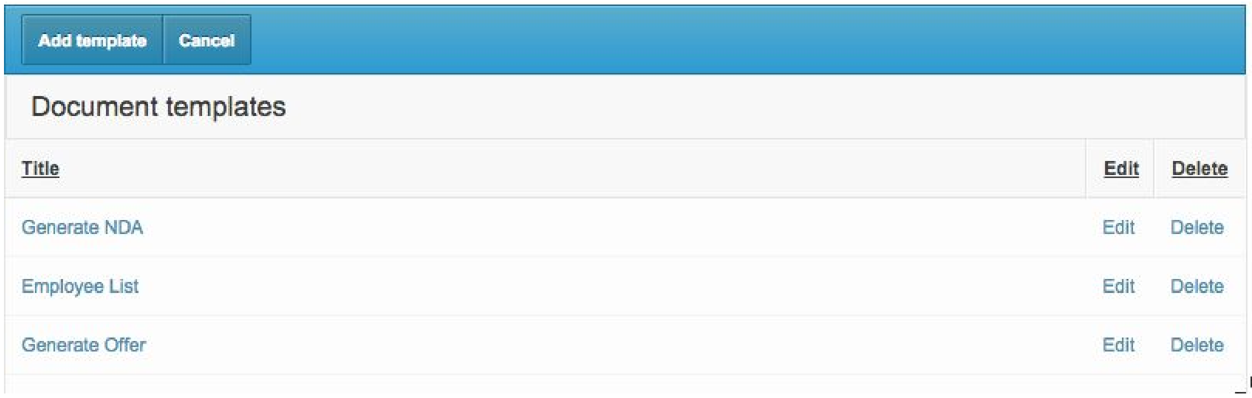
Associating the template with specific views permits you to limit which users have access to a template by also applying security to those views. Therefore, you could have a “Document automation” view, associate a template with only that view, and limit access to that view to a specific security group. That way not every user with access to that iSheet will necessarily be able to generate documents.

Managing Existing Document Templates

Currently, the only options for managing existing document templates are:

- View a list of all templates associated with a site
- Download an individual template
- Delete a template
- Rename a template
- Add a new version of an existing template
- Change the views to which a single-record template is associated and the type of template

To manage existing document templates, click on "Manage doc templates" to view a list of the current templates on the "Document templates" page:

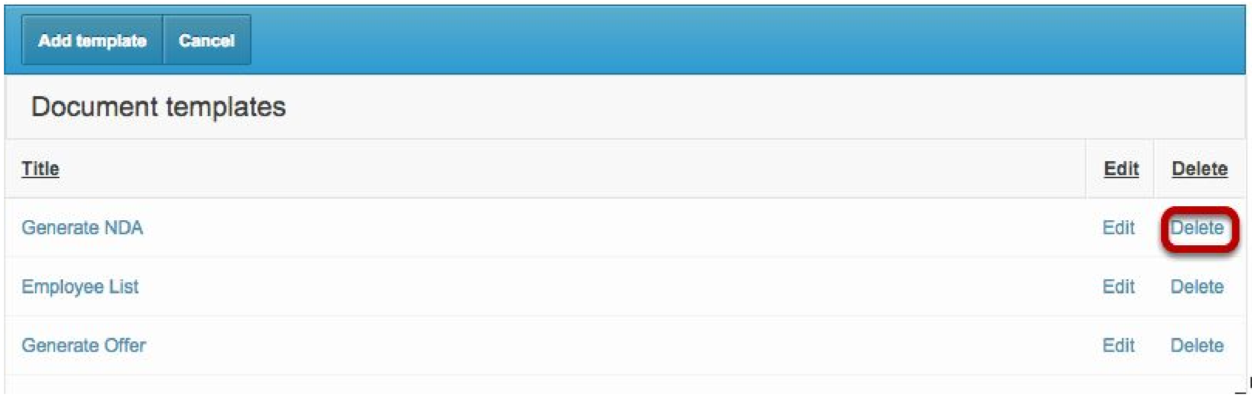


Document templates

Title	Edit	Delete
Generate NDA	Edit	Delete
Employee List	Edit	Delete
Generate Offer	Edit	Delete

To download a template, simply click on the name of the template. (The downloaded file will not have the name of the template, but a unique template ID, like: "483.docx".)

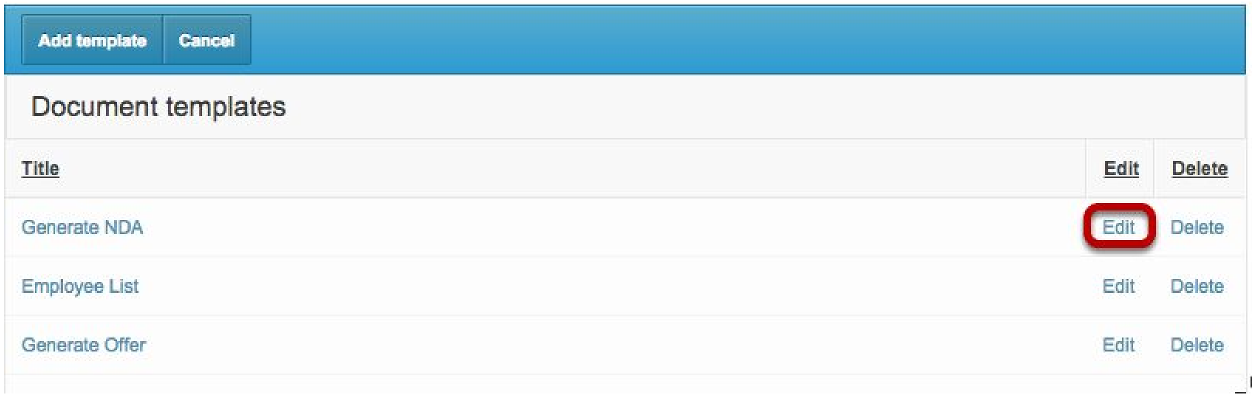
To delete a template, simply click on the "Delete" link across from the template's name:



Document templates

Title	Edit	Delete
Generate NDA	Edit	Delete
Employee List	Edit	Delete
Generate Offer	Edit	Delete

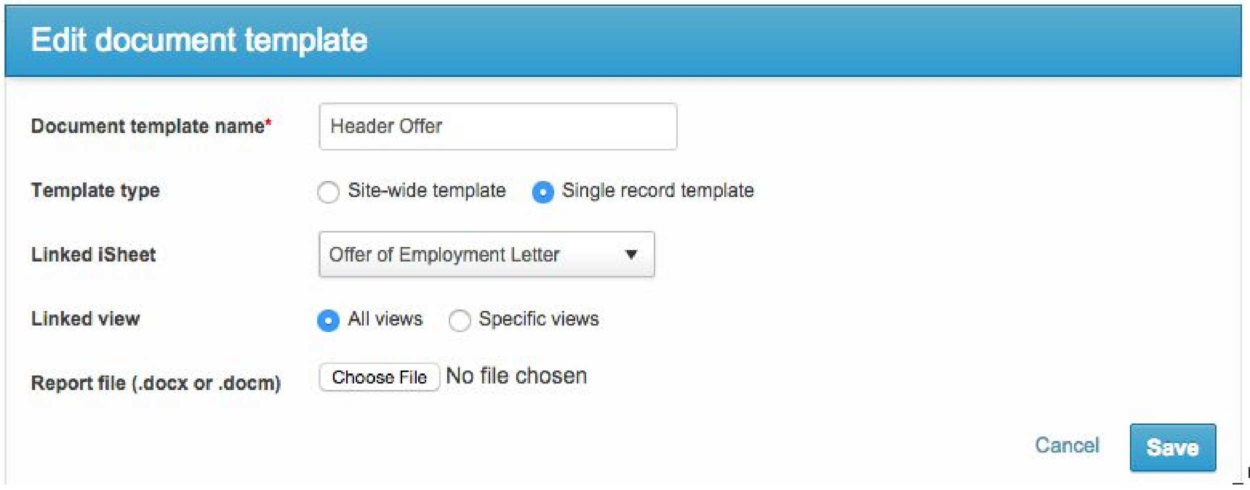
To edit an existing template, click the “Edit” button across from the template’s name:



Document templates

Title	Edit	Delete
Generate NDA	Edit	Delete
Employee List	Edit	Delete
Generate Offer	Edit	Delete

which will bring up the “Edit document template” page:



Edit document template

Document template name*	Header Offer
Template type	○ Site-wide template ● Single record template
Linked iSheet	Offer of Employment Letter
Linked view	● All views ○ Specific views
Report file (.docx or .docm)	Choose File No file chosen

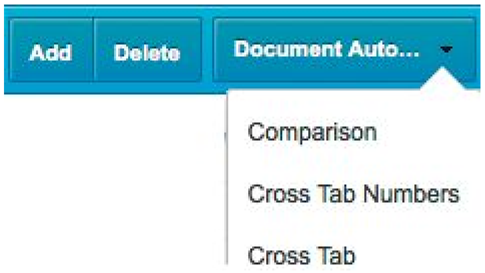
To rename the template, simply change the name in the “Document template name” field and click “Save”. To add a new version of the template, click on the “Choose file” button, select the new template version from the local computer and click “Save”. (Note that there is no way to access prior versions of a template. The new version will fully replace the prior version.) The template will retain the existing name, not the name of the file that was uploaded. It is also possible to change the type of template (site-wide or single record) and to change the views that a single-record template is associated with using those fields.

Generating Documents from iSheet Records and a Template

Each type of document template can be accessed in a different way in order to generate documents from iSheet data.

Site-Wide Templates

For site-wide document templates, at the top of every iSheet in a given site, there will be a pull-down menu (likely called “Document Automation”) listing all of the site-wide document templates (the name of this pull-down menu can be changed from instance to instance by using the System Vocabulary feature in Collaborate, but it will be the same in every site):

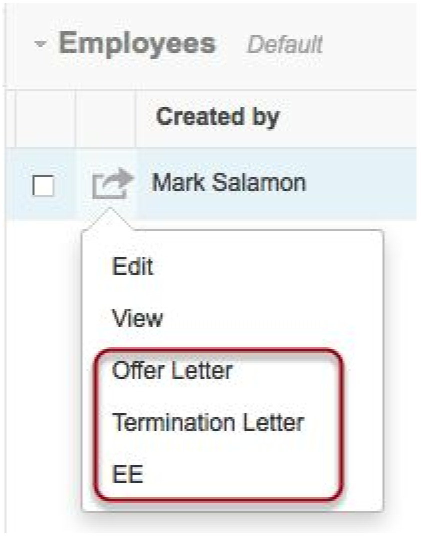


By default, *every record* in every iSheet in the site will be accessible to the template. However, if you check the box next to selected records (currently, these checkboxes are only available for regular iSheets, not document- or file-metadata iSheets), then only those selected records in the currently selected iSheet will be accessible to the template.

Only Site and Content Admins can access this menu.

Single-Record Templates

For single-record document templates, any user with Edit or View rights to a record (in the future, users with View-only rights) will see extra actions on the menu item for that record, listing each single-record template accessible from that iSheet:



Once a document has been generated, it will be downloaded to the user's computer automatically as a .docx file.

Debugging Velocity Templates

Typically, if there is an error in the Velocity references added to a template, an empty document will be generated. Less often, the document will be generated but the output will not be as expected or may show the outputted text as something like: *$data.ColumnVariableName*, instead of the value you expected.

Here are a few tips for building a template using Velocity and also for debugging templates you have built.

Create the iSheet First

Before you add Velocity references to a Word document template, create the iSheet first. Even if the iSheet has not been finalized, you will need the iSheet fields to test the Velocity references and to generate test documents.

Build Incrementally and Test Often

Instead of building the entire template at once and testing at the end, add Velocity references incrementally to your template over time and test each new part before moving onto the rest. That way, if you cannot get the template to work, you will know it must be caused by one of the small number of changes that was just made.

In the Collaborate site, you can add as many versions of a template as you like. It is likely a good idea to keep the last two versions. That way, if you make a change to the template that no longer works and you cannot roll back to that prior version, the prior working version will still be stored in Collaborate.

Formatting

Often the issues encountered when creating a template are simply formatting issues. For example, the document contains extra spaces or line breaks. In that case, make sure Velocity directives are in a small font and grouped together on a single line. The directives could be added to the same line as text that is viewable. Also, make sure to put conditions around any text that should be hidden based on how the condition evaluates. Any spaces or line breaks or other marks inside a condition that evaluates to true will be displayed. And you may want to try to remove the space before and after a paragraph, using the Word features shown above.

Common Problems

Here are some common problems you may encounter that will cause a template to fail.

1. Entering or editing the Velocity reference directly onto the page instead of using the "Field" window. If you right click on the reference and select "Edit field…" you will see the actual text that is being evaluated by the document automation engine, compared to what had been edited directly.
2. When creating a variable, the name of the iSheet, view or column provided does not EXACTLY match the names of those items in the Collaborate site, including letter case and spaces. This may happen if the name of an item was changed later in the Collaborate site.
3. Failing to close a *foreach* loop or *if* condition with «#end». This often happens when there are nested foreach loops and conditions.
4. Using the single equals sign (=) instead of the double equals sign (==) when evaluating conditions.

5. Enclosing a string with a double quote on one end and a single quote on the other end (or vice versa).