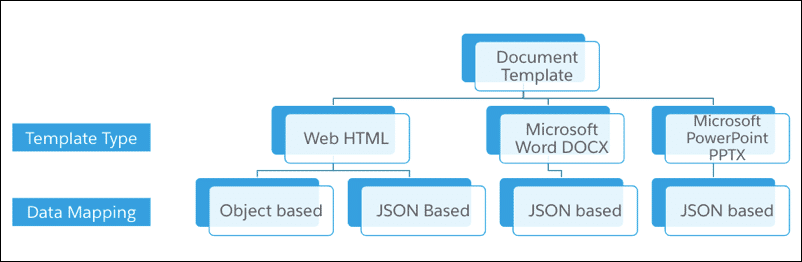
**Document Template Types**

Use the Document Template Designer (Vlocity Document Template tab) to define document templates that generate documents. These can be contract documents that must conform to corporate legal standards or other types of documents relevant to your industry. Document templates define the structure, content, and formatting of these documents.

Functional or business users such as business analysts can create and maintain templates with minimal or no IT support. You can reuse document templates to create documents for multiple contracts.

**Document Template Types**

You can create three types of templates (Web HTML, Microsoft DOCX and PPTX). JSON-based data mapping is available for all types, and object-based mapping is also available for web templates.



**Web HTML**

Build this cloud-based template type in the native HTML format directly in the Document Template Designer. You can define template sections, use the Rich Text Editor to edit and format text, insert tokens, and then merge data into those tokens to generate documents. You can create conditions for when to display specific sections on the generated documents, which are also in the native HTML format.

* Within sections, you can define embedded tokens or merge fields in the following format: %%variable%%.
* You can map the tokens to SFDC objects or to a JSON data structure in DataRaptor (see more below).
* When you merge the data during document generation, the output is HTML, which you can convert to PDF or Word.
* If you only use web HTML templates, it's not necessary to have MS Word installed on your local machine, as the Document Template Designer is cloud-based.

**DOCX**

Build this template in the DOCX file itself, and directly insert tokens into the file. Add the DOCX file to the document template designer, which acts as a container for the template.

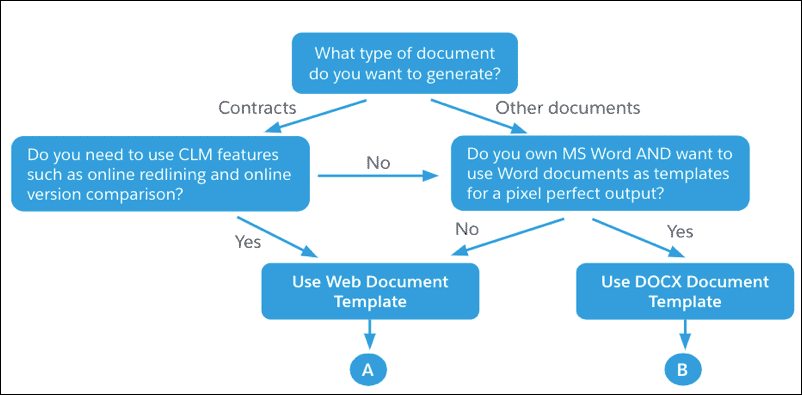
You can merge data from a JSON structure into the DOCX file, and generate an output document.

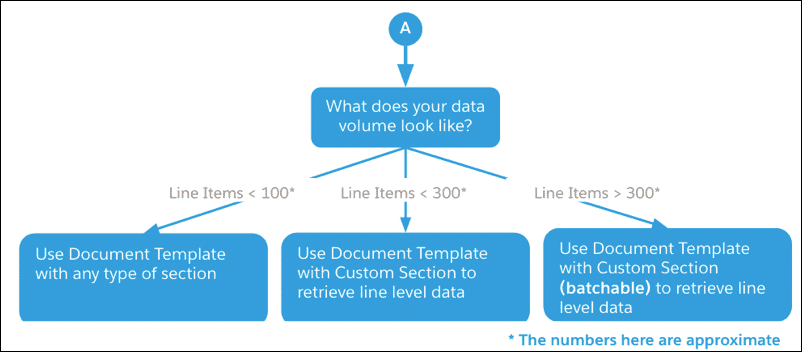
**PPTX**

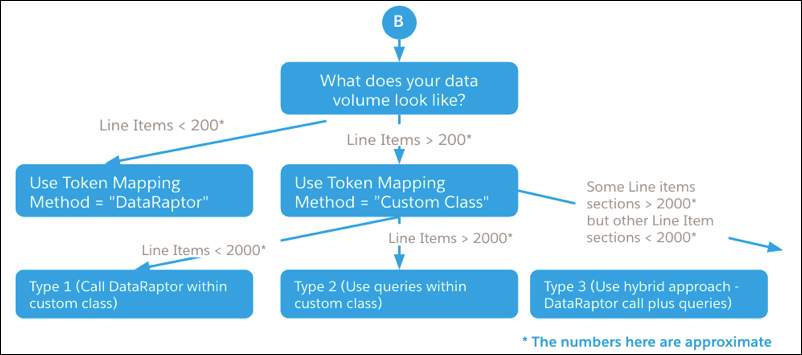
Build this template in the PPTX file itself, and directly insert tokens into the file. Add the PPTX file to the document template designer, which acts as a container for the template.

You can merge data from a JSON structure into the PPTX file, and generate an output document.

Guidelines for Document Template Selection







**When should we use Web HTML Templates?**

* When precise formatting of the output document is not critical (e.g. typical contract documents)
* When you want to track changes by redlining documents online using CLM in the Salesforce application, or use other CLM features such as online version comparison
* When you want to leverage the online clause library and embedded template functionality

Additionally, web HTML templates:

* Support both object-based and JSON-based data mappings
* Generated documents are stored as Salesforce attachments
* Use the following syntax for tokens/merge fields: %%token\_name%%
* Display sections based on conditions that you define using formulas or entity filters
* Currently, support only one level of repeating content
* Support dynamic insertion of hyperlinks

**When should we use Docx Templates?**

* When precise formatting of the output document is critical - for example, two-column formats, custom fonts, background images, and precise placement of text
* When you want to track changes by redlining documents in MS Word rather than CLM
* When you want two or more levels of repeating content blocks in the document

Additionally, DOCX templates:

* Support JSON-based data mappings only
* Generated documents are stored in Salesforce Content
* Use the following syntax for tokens: {{token\_name}}
* Display sections based on Boolean conditions (#IF\_..)
* Support multiple-levels of repeating content
* Currently do not support the dynamic insertion of hyperlinks, images, or charts

# Data Mapping Types

You can design two types of document templates based on how token mapping occurs:

* Object-based
* JSON-based

The data mapping type doesn't impact the end user's experience of generating, redlining, and working with contract documents.

**Object-Based Data Mapping**

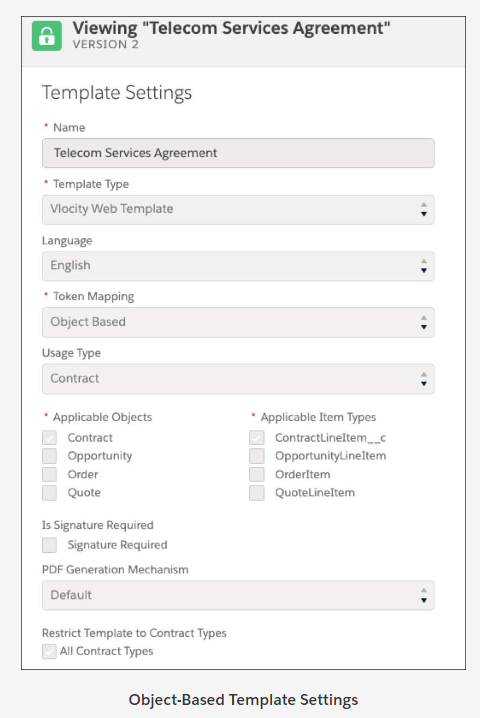
***This data mapping type is currently only available for Web HTML templates.***

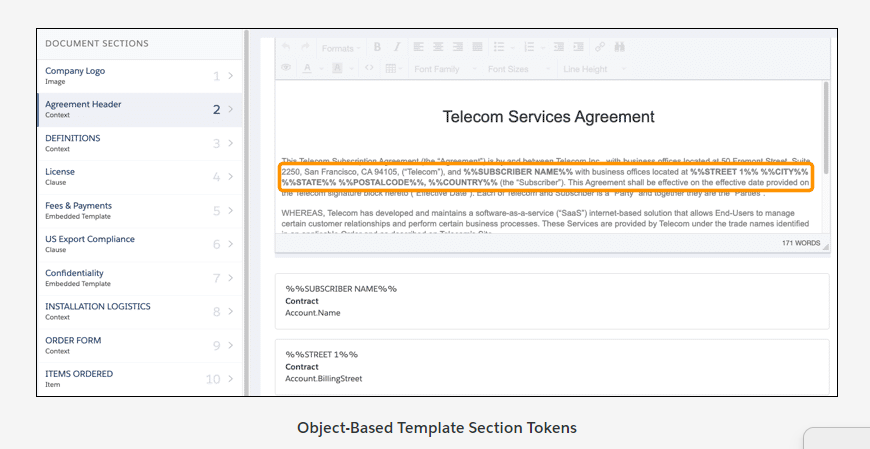
An object-based template is easier to build than a JSON-based template, but is less flexible in terms of the documents it can generate. Use it primarily for contracts rather than opportunities, quotes, or orders, as these objects currently only support PDF generation (therefore, for these use cases, it's best to use the JSON-based template features).

Object-based templates allow you to map tokens directly to Salesforce object fields. You can select applicable Salesforce object data as inputs. You can merge data from these objects into the document sections to produce the final document.

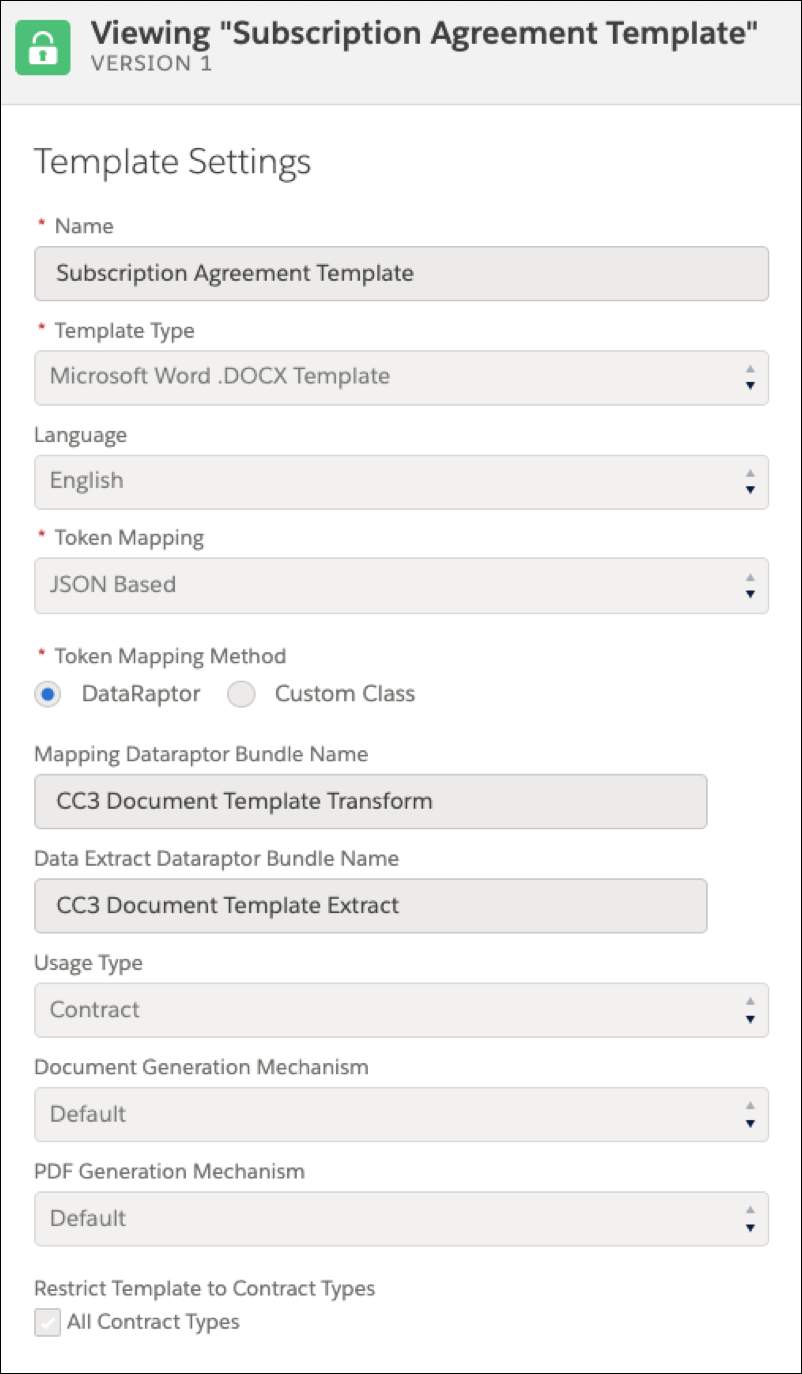
The tokens defined within the template sections are placeholders that are bound to the actual fields in the Salesforce objects. In the image below, note the tokens in the section content, and the definitions below this content.

You can use entity filters for conditional display.

****

****

JSON-Based Data Mapping



***This data mapping type is available for all template types.***

JSON-based data mapping templates allow you to define a document template within CLM, and then use it to generate documents by merging data that can come from any source, such as Salesforce, an OmniScript, or an external system.

Use a DataRaptor as the mechanism to map JSON-based tokens to JSON data. Specify a DataRaptor Transform bundle name, drill down to that bundle, and tokens from the template are pulled into the DataRaptor and mapped to JSON format.

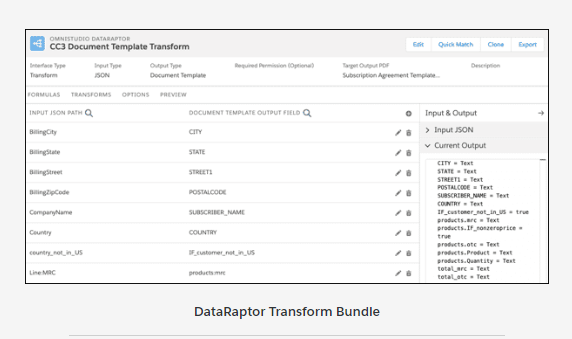
Other mechanisms such as writing an Apex Class to extract or transform data are also possible. However DataRaptor is the preferred mechanism as it's declarative and easily maintainable.

You can generate documents for different use cases across industries. For example, you can link an OmniScript to a sales quote, and the data can come from the OmniScript itself. You can attach the generated document to Salesforce records or email it to recipients.

You can use formulas (rather than entity filters) for conditional display.

Optionally, you can attach another DataRaptor (Extract) to the same template to extract data. However, you need to know which data is merged into which tokens.

In summary, this template type provides you with the maximum flexibility for document generation, but you need to know how to use DataRaptor.

****

Object-Based vs JSON-Based Templates

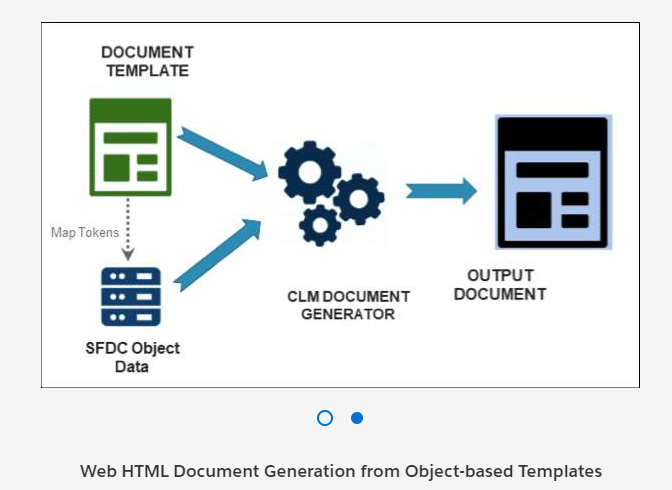
|  |  |  |
| --- | --- | --- |
| Scenario | Object-Based | JSON-Based |
| When should you use them? | * Document Generation within CLM (Contract object) * Limited Support for Quote, Order, Opportunity object (PDF generation only) | * All document generation purposes such as Contracts, Quotes and Insurance Plans * When you want to display line item attributes in the document * Can also be used within CLM |
| Who can build such template? | * Functional/Business-oriented users | * Functional/Business-oriented users (DataRaptor knowledge required) |
| Advantage of each type | * Easier to build as token mapping can be done faster | * Most flexible since data can come from any source. The Repeatable Content section type is also available. |
| Item Section | * Can only operate on lines for opportunities, quotes, and orders, which cannot be filtered for display. | * Opportunity, quote, and order lines, or other data lists such as rate tables, price sheets, and so on. Products attributes from lines can also be displayed. |
| Repeating Content Section | * Not Available | * JSON-based templates have a Repeating Content section type, which allows you to format data as you want it to appear. For example, if you merge data into the document from a quote that has 3 line items, each in a different format, you are not restricted to displaying these line items in a fixed grid or table. We talk about this more in the next lesson about template section types.  Product attributes from lines can also be displayed. |
| Applying Display Conditions to Section | * Simple product-based filter or use of entity filters | * Formula builder for defining conditional sections and repeating content display filter |

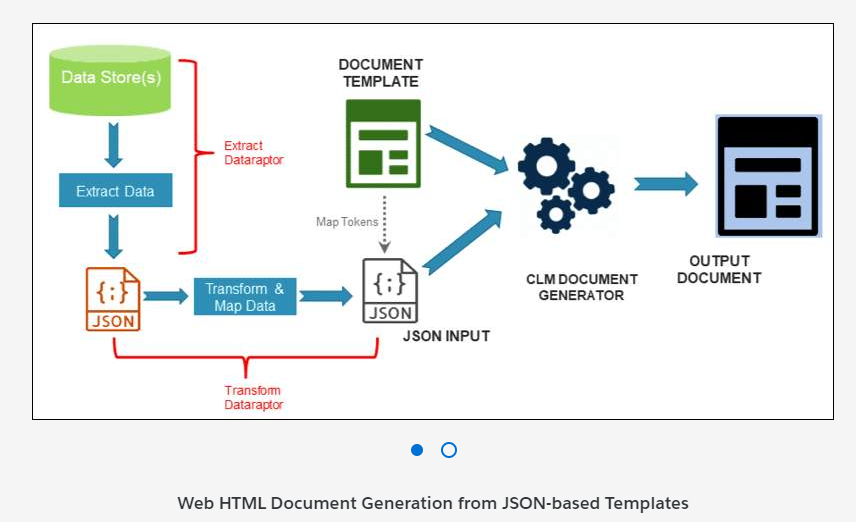
# Web Document Templates

**What are Web HTML Templates?**

Build this cloud-based template type in the native HTML format directly in the Document Template Designer. You can define template sections, use the Rich Text Editor to edit and format text, insert tokens, and then merge data into those tokens to generate documents.

You can create conditions for when to display specific sections on the generated documents, which are also in the native HTML format.

****

****

**When Do You Use Web HTML Templates?**

When precise formatting of the output document is not critical (e.g. typical contract documents)

When you need to use CLM features such as online document customization (redlining) and online version comparison

When you want to leverage the online clause library and embedded template functionality

**Advantages**

* Template design happens 100% in the SFDC cloud – template sections are HTML
* Document generation and storage is 100% in the SFDC cloud
* Allows for online document customization / redlining and comparison between contract document versions
* HTML output document can be converted to Word or PDF

**Disadvantages**

* Advanced formatting capabilities cannot be used to design templates (HTML Rich Text editor)
  + 2 column formats, custom fonts, etc.
* Not a good fit if the output document needs to be “pixel perfect” output

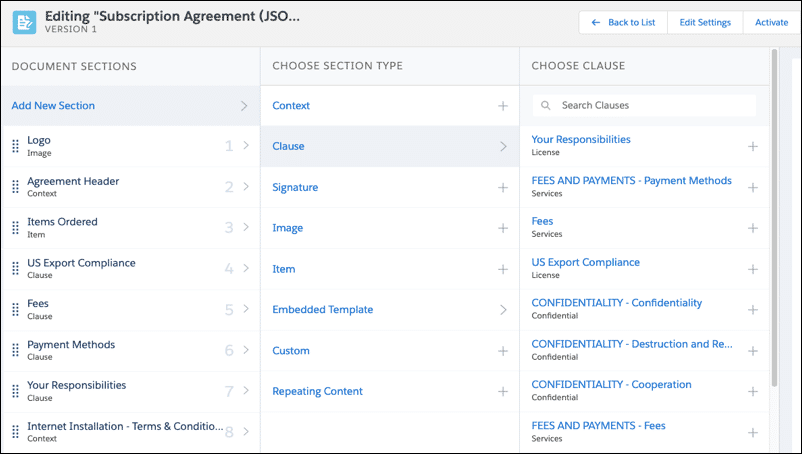
# Create Clauses for Web Document Templates

The Clause Library (Vlocity Document Clauses tab) is where you can store and manage pre-approved legal language and terms, or any kind of language that a document generated from a web template needs to include; for example, payment and warranty clauses. Think of clauses as the smallest building blocks in the construction of a document.

Document clauses are for use with web HTML templates only. You can reuse them across multiple document templates. This helps to make the document creation process more standardized and repeatable.

# Web Document Template Sections

A web document template is an assembly of sections that you order in the sequence you want them to appear in the resulting document. Every section has a type, which has different purposes.



**Add New Section > Choose Section Type.**

**You can also choose existing clauses and embedded templates.**

What do Sections Look Like on the Generated Document?

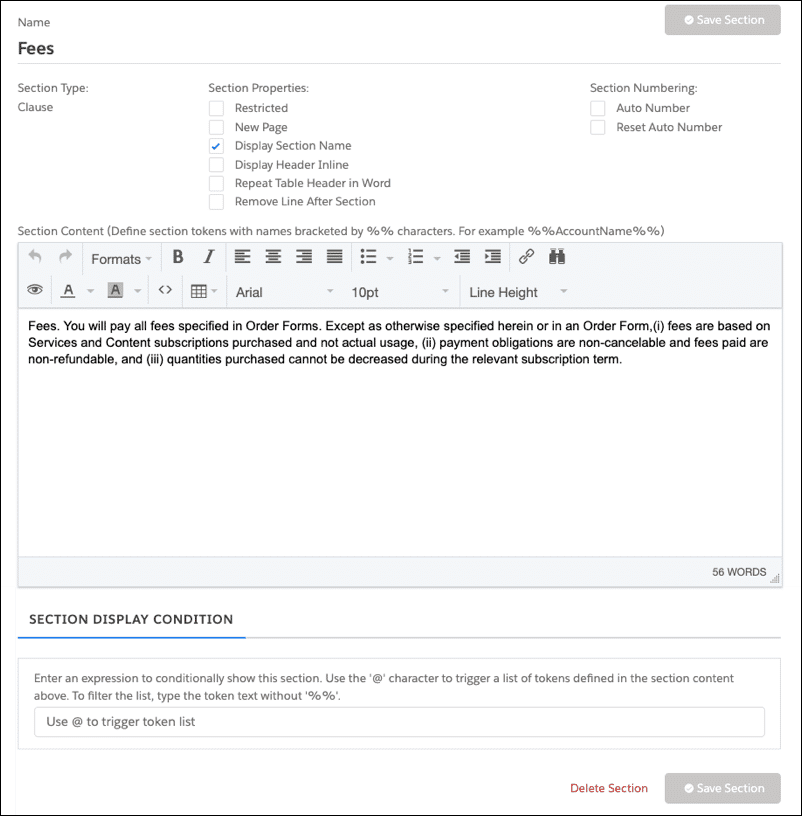
The generated document is displayed in the exact order that you order the sections in its template. However, you can define conditional sections so not all sections are displayed in the generated document if conditions are not met.

We describe each section type below. We've arranged them in alphabetical order as follows:  
Clause, Context, Custom, Embedded Template, Image, Item, Repeating Content, Signature.

**Clause**

When you add a predefined clause from the clause library, the clause text becomes the text in the section. Unless this is a restricted clause, you can use the text as the starting point for making changes.

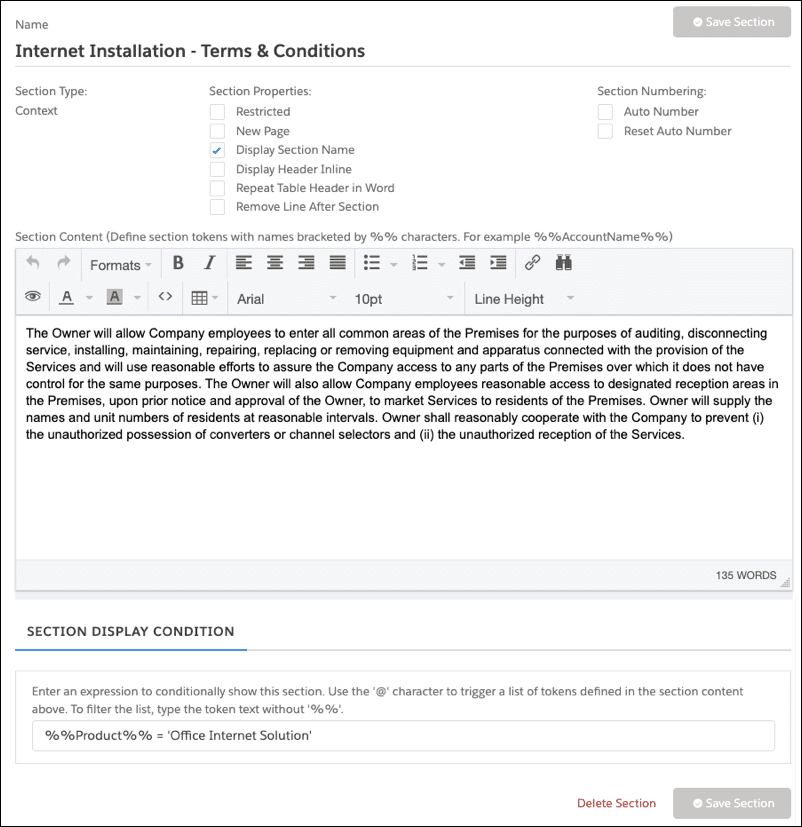
The clause is in plain text but you can use the rich text editor to customize its format.



**Context**

Use a Context section to specify any kind of contextual text you want to include in a contract. For example, you can add a quote or contract header, a Definitions or an Explanation section, and so on.

You can freely type and format text in context sections.

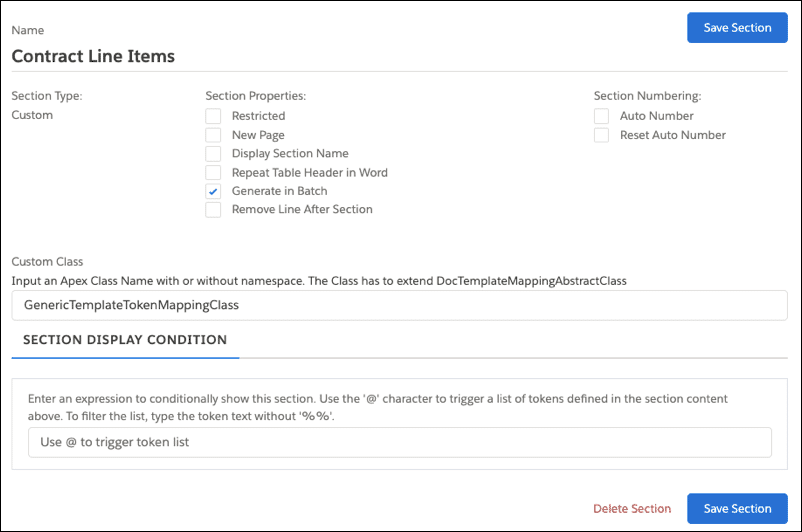


**Custom**

Use a Custom section when none of the other section types meet a business requirement. For example, you need a section for special data with special formatting.

You can write custom logic using an Apex class that implements data fetching and section rendering. You can pull data from custom objects or from sources outside SFDC, write logic to parse this data, and perform calculations if needed. You can then format this data into your desired HTML format. You can see the custom output when the document is converted to Word/PDF.

You may need IT or technical support for this type of section.

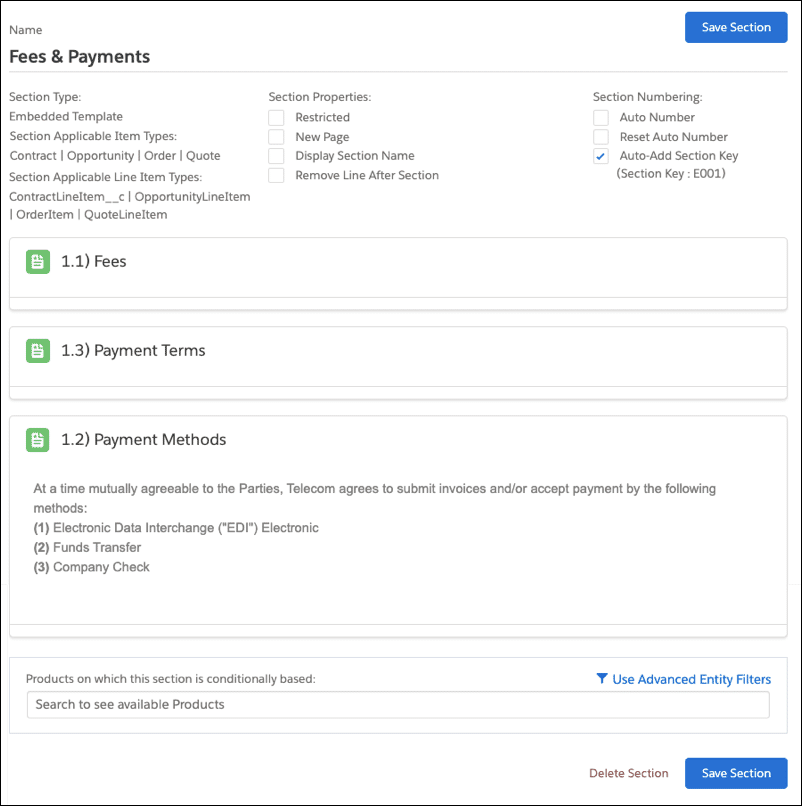


Use a custom section when documents have large numbers of line items to display (for example, multi-site contracts). Select **Generate in Batch Mode**, and write an Apex class to retrieve line items in "chunks".

**Embedded Template**

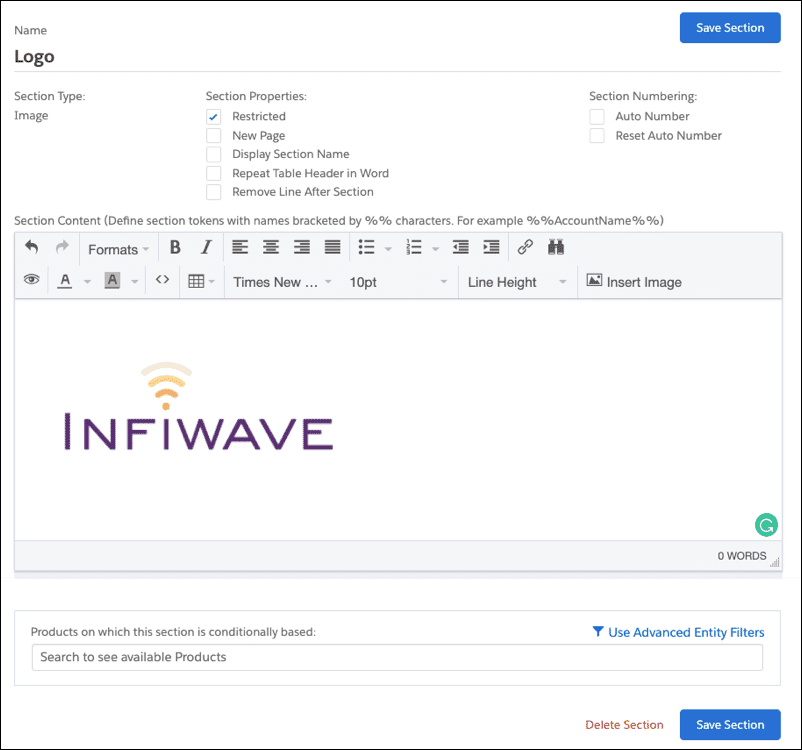
Use this section to reuse an existing template as an embedded template within a parent (or outer) template, based on specific conditions. For example, you can create a template for product-specific language or terms, then include it in a parent template as an embedded template section.

You can create conditions for this section so that it's rendered in the final document only when the product is selected on a contract line.



**Image**

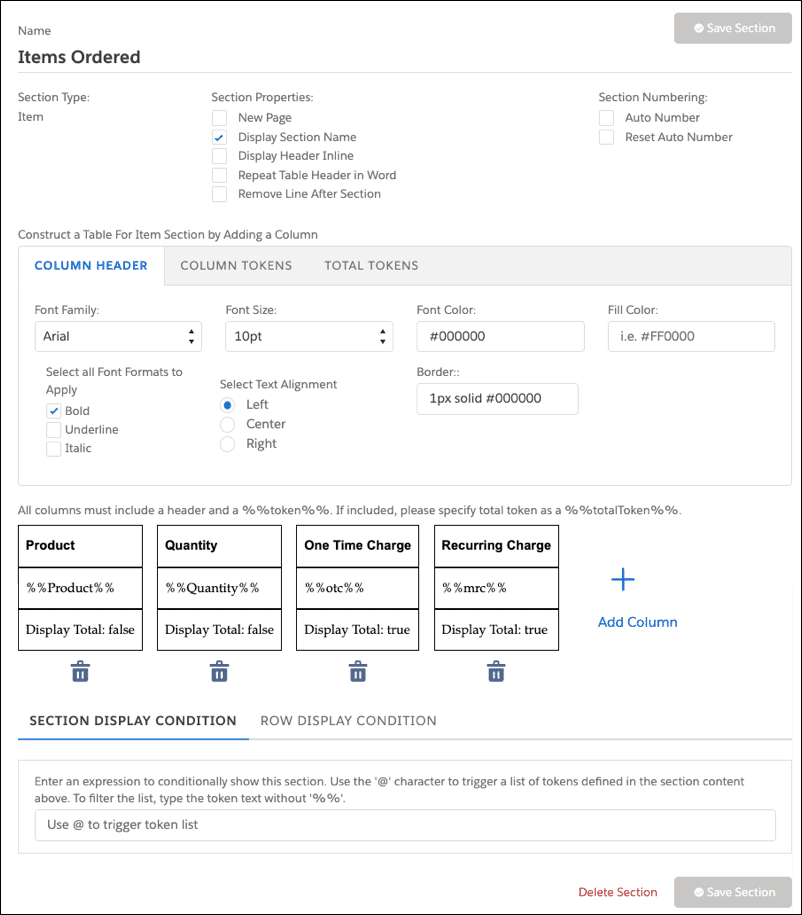
Use this section to insert images such as logos to the body of the document. You can also add text as needed.



**Item**

Use this section to display repeating items in a table format; for example, order lines, rate tables, and so on. You can't use the Rich Text Editor for editing this section, which is always rendered as a table in the contract document. However, you can format the table header and rows, such as selecting different colors.

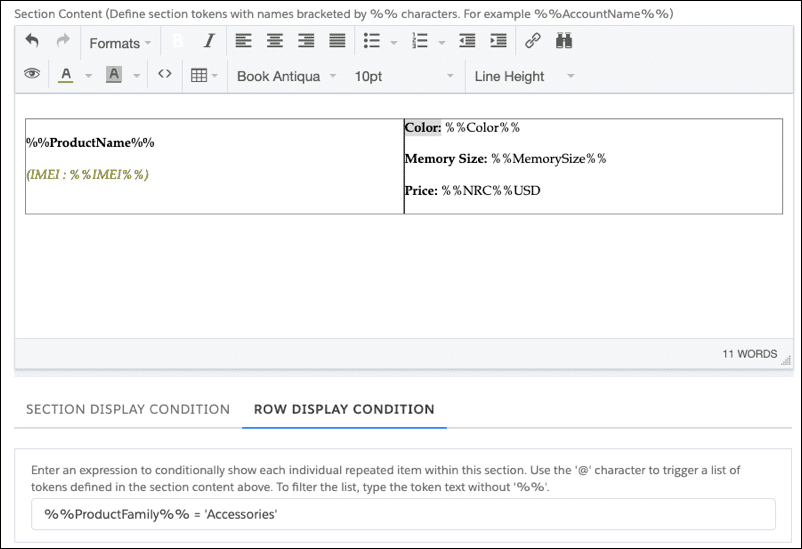
You can create tokens and map them to any type of line item fields.



**Repeating Content**

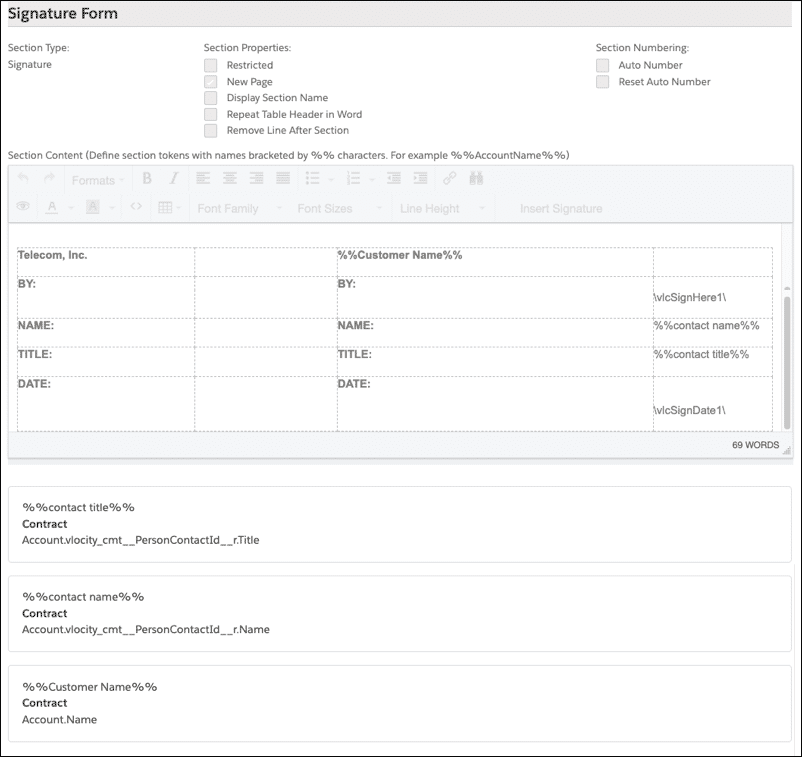
JSON-based templates have a Repeating Content section type, which is similar to the Item section available in object-based mapping document templates, but instead of displaying repeating items only as a table, this section allows you to format data as you want it to appear. For example, if you merge data into the document from a quote that has 3 line items, each in a different format, you are not restricted to displaying these line items in a fixed grid or table.

You can display each line item in an order separately with its own formatting, its own unique attributes, and so on.

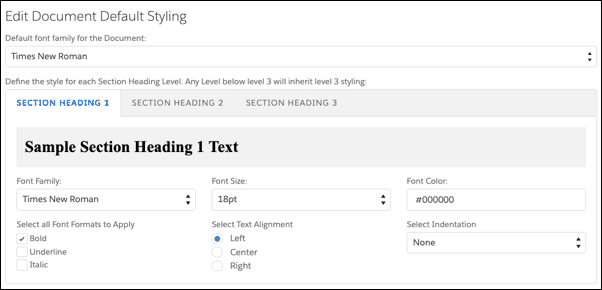


**Signature**

CLM integrates with DocuSign to manage the eSignature flow. Use this section for the DocuSign integration. You can define a signature block and insert tags into the block. DocuSign recognizes the tags when you send generated documents to manage the eSignature process.

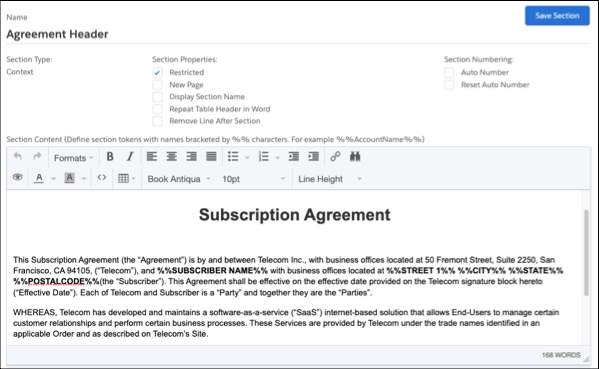


Formatting Section Headers and Text



**Section Headers**

In the Template Settings, you can set the default Section Heading 1, Section Heading 2, and Section Heading 3 styles, including the font family, size, and color.



**Formatting Document Text**

When you edit sections, use the WYSIWYG rich text editor to enter and format the document text, tables, font families, sizes, and color.

The rich text editor is available in most section types.

# Dynamic Document Generation for Web HTML Templates

Applying Conditional Sections to Web HTML Templates

Documents are generated dynamically from document templates. You can apply conditions to template sections to ensure that the generated document is accurate for the context in which it is generated. For example, you can specify the product(s) for which a conditional section will display on the contract document created from the template.

You can use document authoring capabilities to generate dynamic documents for use cases outside of CLM, such as Quote documents, Order forms, and so on.

The template mapping type determines how you can apply conditions:

* **Object-Based:** Simple product-based filter or use of advanced entity filters.
* **JSON-Based:** Formula builder for conditional section display, and conditional row display for repeating content.

# Generating Contracts Using Batch Mode

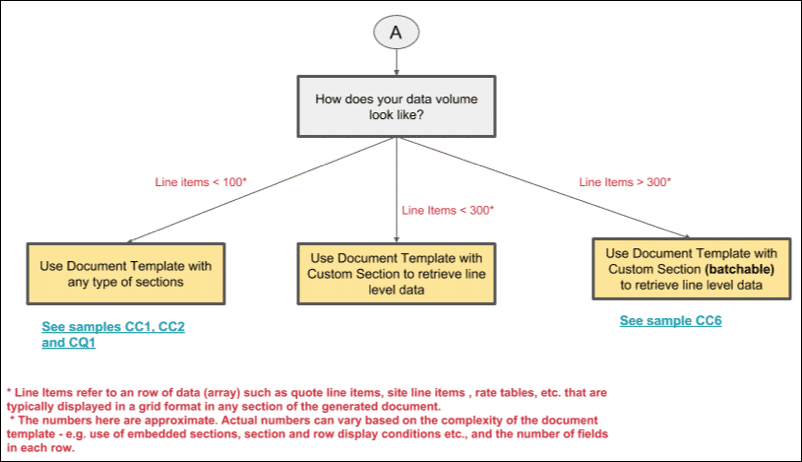
Contract generation can handle up to approximately 100 line items with good performance.  
To create documents with more than 100 line items, generate the contract in batch mode. Batch mode is available for object-based or JSON-based web document templates (not for DOCX or PPTX templates).

You use a **Custom** section to fetch and format lines. In this section, select **Generate in Batch**, and enter the name of the custom Apex class that you have defined (see below).

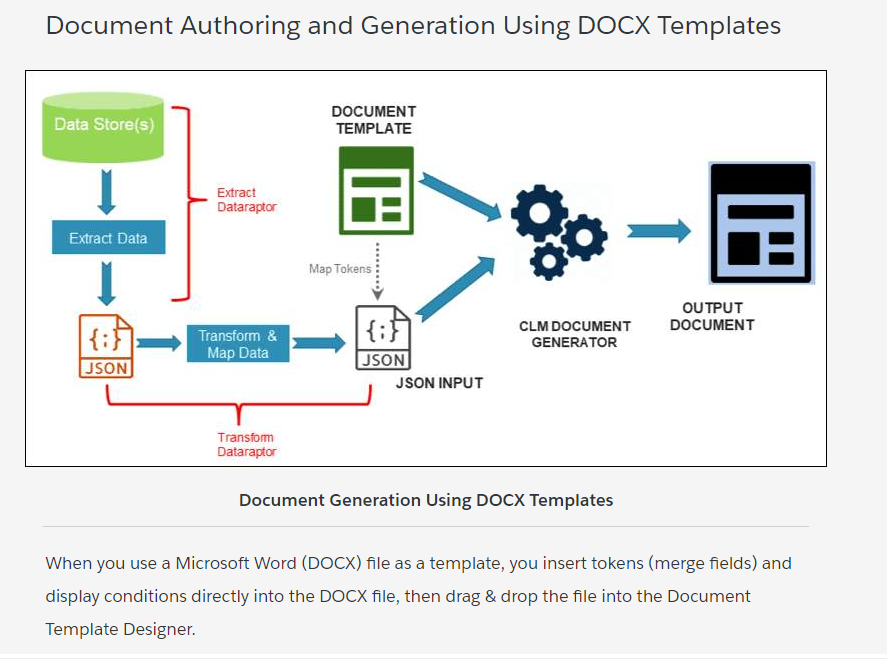
**Writing a Custom Class for Batch Mode**

To implement batch mode, you must define a custom Apex class that extends DocTemplateMappingAbstractClass. The class must implement the methods described below. You can use the Batch Generation Example Class (provided in your documentation) as the basis for your custom class.

|  |  |
| --- | --- |
| public override List getNextSet (String sectionName) | Generates the HTML table content required in each contract document section. The repeatingMapList holds the HTML content for each line item. Add the tableEndTag at the end of the batch job for the last line item. |
| public override Boolean hasNextSet (String sectionName) | Check whether current processing is required. The hasNextStep() method is executed before getNextSet(). To ensure that all records are processed, this method must return true when there are additional records and false when all records have been processed. |
| public override Integer getStartIndex() | Returns the section sequence. |
| public override Integer getBatchDataCount (String sectionName) | Returns the line item count. |



# Microsoft Word DOCX Document Templates

****

As you need a JSON input, you define and specify DataRaptors for data mapping and extraction. You extract the tokens that you create in the DOCX file into DataRaptor, then you can map your JSON to these tokens. The runtime generation calls the DataRaptor, transforms the data, and generates the document.

After you specify the DataRaptors for the template, you can test it from within the designer. You can generate PPTX or PDF outputs.

When Should You Use DOCX Templates?

* When precise formatting of the output document is critical. For example, you want two column formats, custom fonts, background images, and precise placement of text.
* When you don't need to modify the generated document online or you want to modify it in MS Word instead of in Salesforce.
* When you want two or more levels of repeating content blocks in the document.

Create a DOCX Document Template

**In Microsoft Word:**

1. **Create**a “template” file in MS Word or use an existing file.
2. Embed tokens into the file for data that you need to merge in dynamically.
3. **Save** the file in DOCX format.

**In Document Template Designer:**

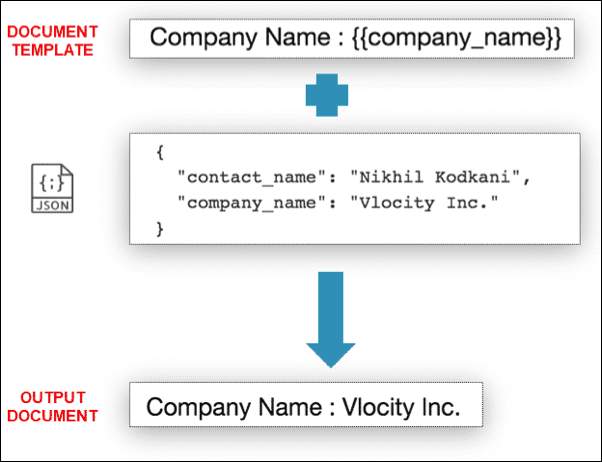
1. **Create**a new document template of type Microsoft DOCX.
   1. To use within contracts, select “Contract” as the usage type
   2. For use outside contracts, you can create your own usage type
2. **Drag and drop** the Word DOCX file into the document template.
3. **Specify** extract and mapping DataRaptors for the document template.  
   (More about this later.)
4. **Test** the document template using the “Test Template” feature.
5. **Activate** the document template.

# Embedding Tokens in Microsoft DOCX Files

You can embed three types of tokens in Microsoft DOCX (or PPTX) files:

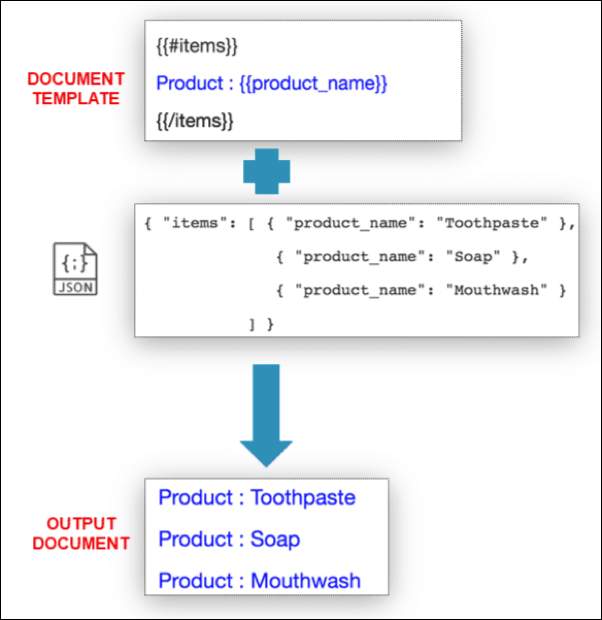
* Variable
* Repeating Content
* Condition Evaluation

Variable Tokens



* **Purpose:** Variable tokens display the value of a variable that is passed into the document template as a JSON element.  
  The data that comes in from the JSON is merged into these tokens.  
  The formatting of the text that replaces the token is the same as that of the token: for example, font type, size, and color.  
  The displayed value uses the formatting of the first curly brace. For example, if the first curly brace in the token is bold, the text in the token is bold as well.
* **Token Notation:** {{token\_name}}

Repeating Content Tokens



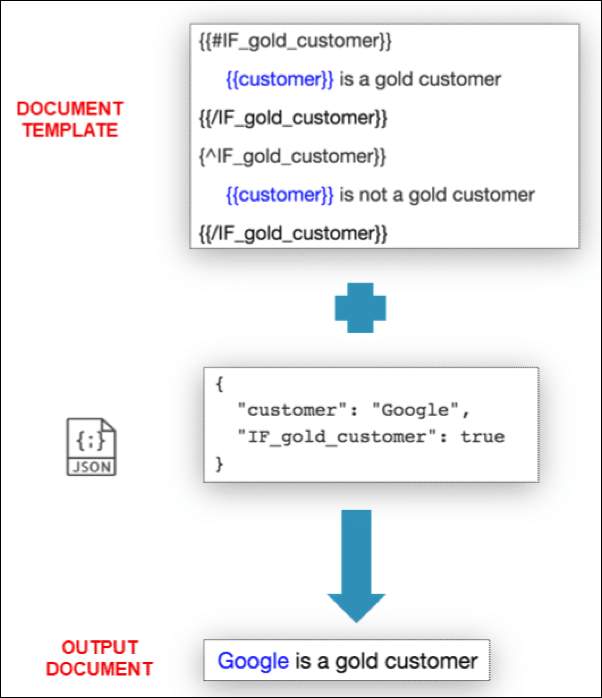
* **Purpose:** Repeating content tokens loop over and display content as many times as there are elements in the JSON array. The content can be a paragraph, numbered list, table, row in a table, and so on. The repeating content tokens themselves are not displayed and the values don't use the token text formatting.
* **Token Notation:**  
  A token beginning with the pound sign (#) indicates that the section should repeat as many times as the item occurs in the input JSON array.

{{#token\_name}}  
<Content to repeat>  
{{/token\_name}}

A token beginning with the caret (^) indicates what should be displayed if the input JSON array is empty or does not exist. In this case, the section is displayed only once.

{{^token\_name}}  
<Content to display>  
{{/token\_name}}

Condition Evaluation Tokens



* **Purpose:**Conditional Evaluation tokens check a Boolean condition to display a section once or to not display the section at all. The token value passed has to be Boolean (true or false). The content can be a paragraph, numbered list, table, row in a table, and so on. The condition evaluation tokens themselves are not displayed, and the values don't use the token formatting.
* **Token Notation:**  
  A token beginning with the pound sign, IF, and underscore (#IF\_) indicates that, if the token value is TRUE, the section appears once, and does not appear if the token value is FALSE.

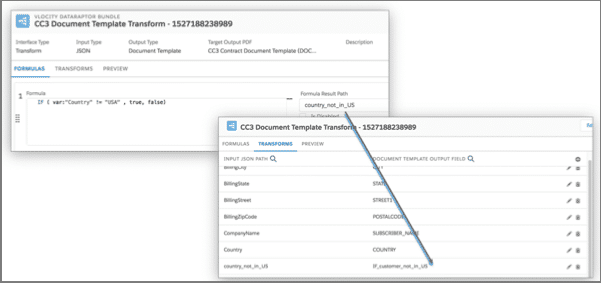
{{#IF\_token\_name}}  
<Content to display>  
{{/IF\_token\_name}}

A token beginning with the caret, IF, and underscore (^IF\_) indicates that, if the token value is TRUE, the section should not appear, but, if the token value is FALSE, the section should appear once.

{{^IF\_token\_name}}  
<Content to display>  
{{/IF\_token\_name}}

**Evaluating Conditions to True or False in DataRaptor Formulas**

Document Templates are logic-less. Therefore, you evaluate conditions to true or false in DataRaptor formulas

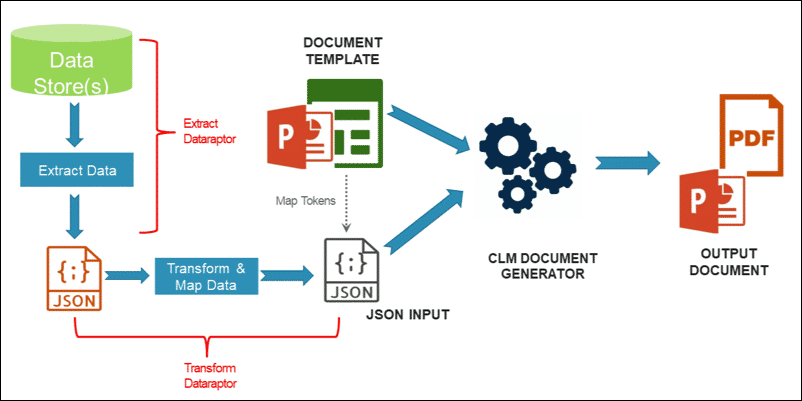


# Working with Microsoft Word DOCX Templates in CLM

Adding a Microsoft Word DOCX File to the Document Template Designer

When you finish defining the DOCX template, drag and drop the file into the Document Template Designer in Salesforce. To do this, go to the **Vlocity Document Template** tab and create a new template.

# Microsoft PPTX Document Templates



When you use a Microsoft PowerPoint (PPTX) file as a template, the process is similar to using a DOCX file as a template. You embed tokens and display conditions directly into the PPTX file, then drag & drop the file into the Document Template Designer.

As you need a JSON input, you define and specify extract and mapping (transform) DataRaptors for the document template.

After you do this, you can test the template from within the designer. You can generate PPTX or PDF outputs (see below).

**Building PPTX Document Templates**

In Microsoft PowerPoint:

1. **Create a** “template” file or use an existing file.
2. **Insert** tokens and display conditions directly into the file for data that you need to merge in dynamically.
3. **Save** the file in PPTX format.

In Document Template Designer:

1. **Create**a new document template of type Microsoft PPTX.
   1. To use within contracts, select “Contract” as the usage type
   2. For use outside contracts, you can create your own usage type
2. **Drag and drop** the PPTX file into the document template.
3. **Specify** extract and mapping (transform) DataRaptors for the document template.
4. **Test** the document template using the Test Template feature.
5. **Activate** the document template.

Inserting Tokens and Displaying Conditions in PPTX Files

You can embed Variable, Repeating Content, and Condition Evaluation tokens in Microsoft PPTX files, as is the case with DOCX files. Therefore, the **Embedding Tokens in DOCX Files** lesson is also relevant for PPTX files.

# Formatting Clauses and Clause Insertion

**Clauses**

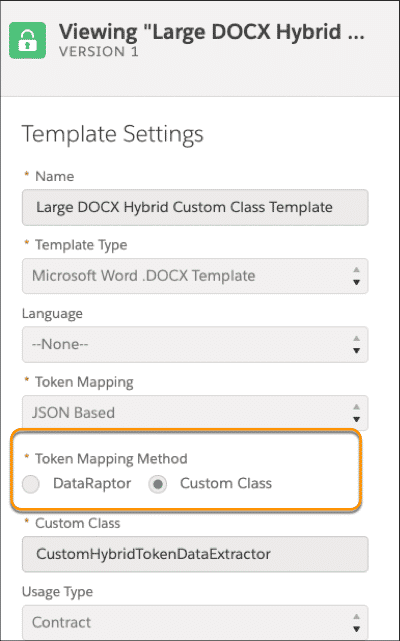
Clauses specify a contract’s legal language and provisions, such as payment and warranty information. A clause typically contains one or more paragraphs of text, which is the clause content. For example, a payment clause may contain text that describes the terms of payment.

* A clause is the smallest building block in a contract. Clauses are the typical contract language you define for your organization, such as terms and conditions and other legal statements.
* You create contract clauses, each with its own unique ID and description. Clauses support variables that are replaced when the clause is added to the contract. You can associate each clause with one or more products and one or more contract templates.
* You can create clauses in any sequence, and then use document templates to place the clauses in the correct sequence.
* You can insert variables in a clause.

# Generating Large Multi-Site Documents Using Word

You may need to generate large documents that have many pages, mainly due to a large number of order or contract line items. This especially applies to a multi-site situation where each site has a lot of line items. In this situation, you will want to display data from each of those sites in one document without violating governor limits (which can happen when you try to query a large data set or process that data set for generating the document).

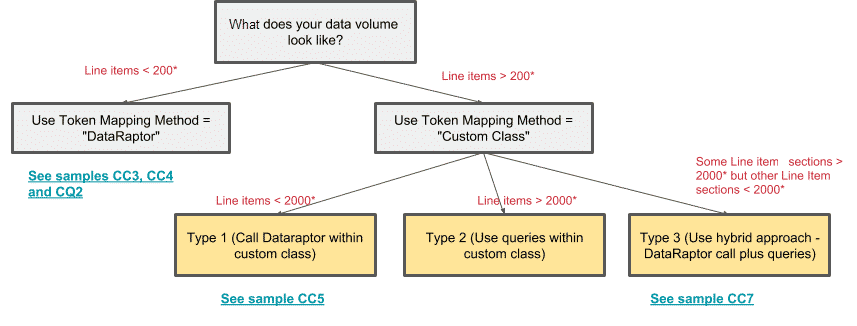
To support this scenario, CLM provides an alternate method to fetch data. The typical method is to use DataRaptors. This is the preferred method because it's completely declarative. Alternatively, you can write a custom Apex class for fetching the data that is fed into the template to generate the document. Within the custom class, you can write code to get the data in chunks (which we refer to as *pagination*). You could also call DataRaptors from the custom class for the data.



You can choose to use a custom class when you create a document template with **Template Type = Microsoft Word .DOCX** **Template**.

The main purpose is to support large data sets, which can't run **getTokenData** using one DR call. The custom class can perform queries in a paginated way to feed token data to the doc gen UI.

DOCX Template Selection Guidelines



\*Note:

* Line items refer to a row of data (array), such as quote line items, site line items, or rate tables, that are typically displayed in a grid format in any section of the generated document.
* The numbers here are approximate. Actual numbers can vary based on the complexity of the document template (e.g. section and row display conditions and the number of fields in each row).