Document Template Design User Guide

Introduction

This document is focused on document template layout design and related features. Document layout and related data settings are stored in a Document Template Version record. If you are not yet familiar with basic Document Engine setup please read the Document Engine Admin Guide first.

The Layout section of a Template Version should contain a valid HTML document layout. Add related CSS classes in the Styles tab. We won't cover HTML basics here, as it's a well-known markup language with extensive documentation available online.

Document Engine's power comes from defining dynamic content and referencing Salesforce record data via Handlebars blocks and expressions. This guide focuses on those features.

Handlebars Overview

Handlebars is a template engine that lets you add dynamic content and logic to your document templates. Handlebars template syntax is described in detail on our website https://document-engine.io/docs/ as well as the official handlebars guide https://handlebarsjs.com/guide/ Handlebars uses double curly braces `{{ ... }}` to mark dynamic content. Dynamic content can be a plain expression that renders its value to a document:

```
<div>{{ expression }}</div>
```

or it can be a block with opening and closing statements:

```
{{#if expression}}
    <div>block content</div>
{{/if}}
```

Blocks and expressions can be nested, expressions use parentheses `( )` for subexpressions.

Block Helpers

Handlebars block helpers control template flow and create scoped contexts. They use opening and closing tags.

#if - Conditional Rendering

Renders content when the expression is truthy. Optional **`{{else}}`** clause.

```
{{#if expression}}
    <div>content when true</div>
{{else}}
    <div>content when false</div>
{{/if}}
```

#unless - Inverse Conditional

Renders content when the expression is falsy.

```
{{#unless expression}}
    <div>block content</div>
{{/unless}}
```

#each - Iterating Over Lists

Iterates over a list or array. Field references inside resolve to the current item.

```
{{#each listExpression}}
    <div>item content</div>
{{/each}}
```

#with - Changing Context

Changes the field context to a related object.

```
{{#with expression}}
    <div>content relevant to the expression object</div>
{{/with}}
```

Salesforce Merge Fields

In expressions, Salesforce object fields can be referenced by their API names, which are prefixed with a colon, such as **`:StandardFieldName`** or **`:CustomFieldName__c`**. These field names are relative to the object that the template applies to. Throughout this documentation, we refer to that object as the root object. Expressions can reference related fields and related lists of the root object, as well as fields within those related objects, up to 5 levels deep. Expression paths are built using dot notation. This works the same way as in Salesforce formula expressions; however, each individual field API name must be prefixed with a colon. For instance, these would be valid field reference expressions for a template designed for the Opportunity object type:

```
{{ :StageName }}
{{ :Account.:Industry }}
{{ :Account.:Owner.:Name }}
{{ :Account.:Parent.:Parent.:Parent.:ParentId }}
{{#with :Account.:Owner }}
    <div>Owner: {{ :Name }}, {{ :Email }}</div>
{{/with}}
{{#each :OpportunityLineItems}}
    <div>{{ :Product2.:Name }}</div>
    <div>{{ :Product2.:ProductCode }}</div>
{{/each}}
{{#each :Account.:Contacts}}
    <div>{{ :Email }}</div>
{{/each}}
```

Note: Merge fields should strictly match the case of corresponding field API names. For instance, **`:Stagename`** or **`:stageName`** will be marked as invalid Opportunity fields as the correct name should be **`:StageName`**.

`#with` and **`#each`** blocks create an inner field context for their content, so **`:Name`** inside an **`#each`** block will refer to the name of a record from the list referenced in the block expression. If you need to reference a field from a parent context inside **`#each`** or **`#with`**, you can use a relative field path with **`../`**, which moves one level up to the parent context.

```
{{#with :Account.:Owner}}
```

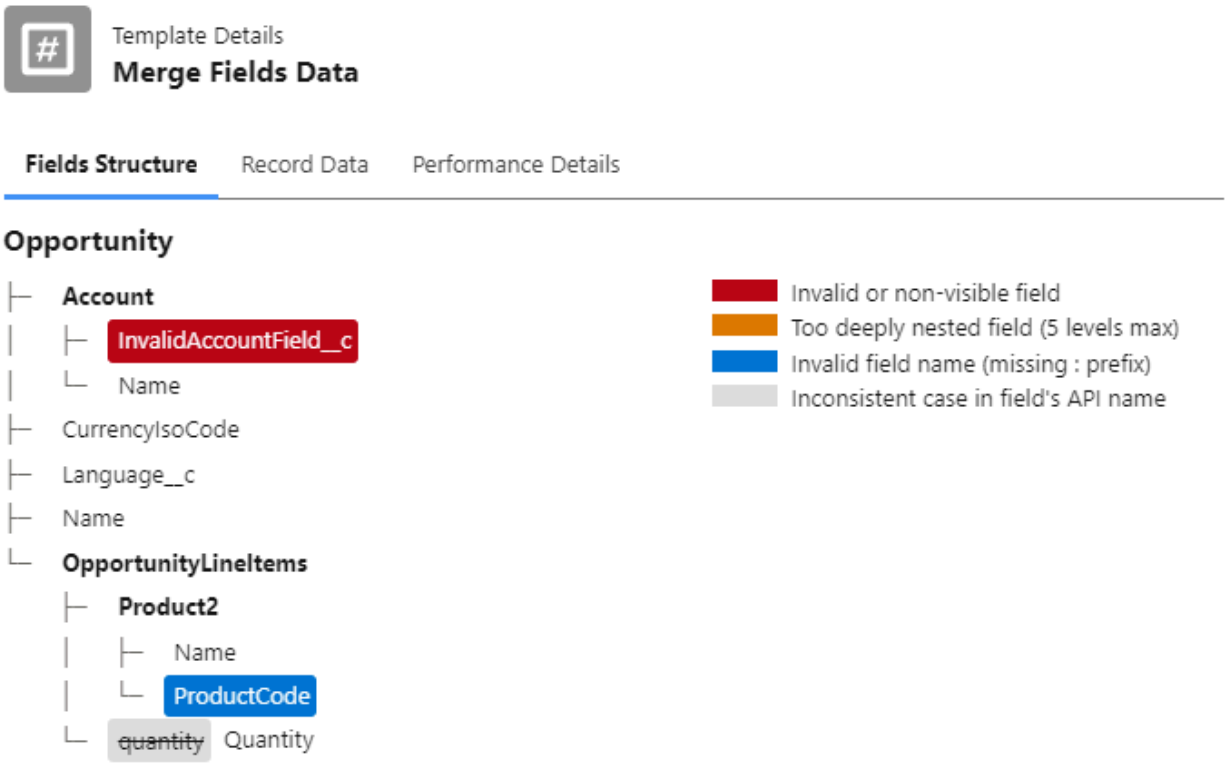
```
    <div>Opportunity Name: {{ ../../:Name }}</div>
    <div>Account {{ ../:Name }} is assigned to {{ :Name }}</div>
{{/with}}
```

You can also reference the root object context using the **`@root`** keyword. For instance, the following expression refers to the root record's name regardless of where it is placed in a template:

```
Opportunity Name: {{ @root.:Name }}
```

During document generation or preview, merge fields are validated against the Salesforce schema. If some fields do not exist or are not properly defined, the system shows a validation message. You can get details on valid and invalid fields using the Review Merge Fields action in the editor's top right corner.

Note: Document Engine follows Salesforce security recommendations and makes all data queries in user mode. Please make sure all merge fields are visible to users who will generate corresponding documents. Otherwise, they will see an invalid field error message and will not be able to generate a document.



Function Helpers

Handlebars helpers are custom functions that you can use in Handlebars templates to format data, perform logic, or transform values. They can be used inline or in sub-expressions. Document Engine provides a set of custom helpers for common operations. The following are some of them. Please refer to the documentation section https://document-engine.io/docs/ on our website for the complete list.

is - Value Comparison

```
{{#if (is value1 operator value2)}}
    ...
{{/if}}
```

The **is** helper takes three arguments:

- **value1**: The first value to compare.
- **operator**: The comparison operator to use. This can be one of the following values:
  - **==** Returns true if value1 is equal to value2.
  - **===** Returns true if value1 is strictly equal to value2.
  - **>** Returns true if value1 is greater than value2.
  - **<** Returns true if value1 is less than value2.
  - **>=** Returns true if value1 is greater than or equal to value2.
  - **<=** Returns true if value1 is less than or equal to value2.
  - **!=** Returns true if value1 is not equal to value2.
  - **!==** Returns true if value1 is not strictly equal to value2.
  - **$>** Returns true if value1 contains value2, case-insensitive.
  - **<$** Returns true if value2 contains value1, case-insensitive.
  - **$$>** Returns true if value1 contains value2, case-sensitive.
  - **<$$** Returns true if value2 contains value1, case-sensitive.
  - **<#** Returns true if value1 is contained in an array value2.
  - **&&** Returns true if arrays value1 and value2 have at least one element in common.
- **value2**: The second value to compare.

Example:

```
{{#if (is :StageName '==' 'Closed Won')}}
    <div>Opportunity is Closed Won</div>
{{/if}}
```

and - Logical AND

The **and** helper takes any number of arguments, and returns true if all of them are truthy.

Example:

```
{{#if (and :IsActive :IsPrimary)}}
    <div>Active and primary</div>
{{/if}}
```

or - Logical OR

The **`or`** helper takes any number of arguments, and returns true if at least one of them is truthy.

Example:

```
{{#if (or :IsActive :IsPrimary)}}
    <div>Active or primary</div>
{{/if}}
```

formatNumber - Number formatting

```
<div>The number is {{ formatNumber number format }}.</div>
```

The formatNumber helper takes two arguments:

- **`number`**: The number to be formatted.
- **`format`**: The string format to use for the number. This can be any valid numeral.js format string.

Examples:

```
The number is {{ formatNumber :Quantity '0,0' }}.<br/>
The number is {{ formatNumber :Discount__c '0.00%' }}.<br/>
The number is {{ formatNumber :Amount '$0,0.00' }}.<br/>
```

date - Date formatting

```
<div>The number is {{ date value format }}.</div>
```

The date helper takes two arguments:

- date: The date to be formatted.
- format: The string format to use for the date. This can be any valid moment.js format string.

Examples:

```
The date is {{ date :CloseDate 'DD/MM/YYYY' }}.<br/>
The date is {{ date :CloseDate 'YYYY-MM-DD' }}.<br/>
The date is {{ date :CreatedDate 'dddd, MMMM Do YYYY, h:mm:ss a' }}.<br/>
```

select - Filtering list of records based on record's field value

```
{{#each (select list field operator value) as |record|}}
     <p>{{ record }}</p>
{{/each}}
```

The select helper takes four arguments:

- list: list of objects
- field: field of object to check
- operator: The comparison operator to use. This can be any valid operator that the **`is`** helper supports
- value: The value to compare the object's field to

Example:

```
{{#each (select :OpportunityContactRoles ':IsPrimary' '==' true) as |role|}}
     <p>{{ role.:Contact.:Name }}</p>
{{/each}}
```

Note: `|role|` is a block parameter. Block parameters are a feature of Handlebars supported by #each and #with blocks. They allow you to assign a name to the block's context object and reference it inside the block.

Data Builder

For complex data scenarios that can't be handled with merge fields and built-in functions, you can create a custom Data Builder Apex class by implementing the **`DocumentDataBuilder`** global interface:

```
global interface DocumentDataBuilder {
```

```
    SObjectType getType();
    String getData(final Id recordId);
    String getFileName(final Id recordId);
}
```

Here is a short example implementation to illustrate a way this could be done.

```
global with sharing class AccountDataBuilder implements nbse.DocumentDataBuilder {

    public SObjectType getType() {
        return Account.SObjectType;
    }

    public String getData(final Id recordId) {
        final Account account = [
            SELECT Name, CurrencyIsoCode, Owner.Name
            FROM Account
            WHERE Id = :recordId
            WITH USER_MODE
        ];
        return JSON.serializePretty(new Map<String, Object> {
            'id' => recordId,
            'name' => account.Name,
            'ownerName' => account.Owner.Name,
            'currencyCode' => account.CurrencyIsoCode
        });
    }

    public String getFileName(final Id recordId) {
        return null;
    }
}
```

Note: The Data Builder implementation class has to be marked as **`global`**; otherwise, it won't be visible for the Document Engine package.

Of course, a real-life example would be more complicated and contain logic that could be implemented using merge fields and helper functions. The **`getData`** method should return a

string with a valid JSON object. The data will be available in the root context of the template and should not be prefixed with a colon in Handlebars expressions:

```
Owner: {{ ownerName }}
Currency: {{ currencyCode }}
```

The **`getFileName`** method can be used to specify a custom filename for generated documents. If it returns a non-blank value for a record, that value takes precedence over the filename from the Document File Name Formula configured for the template.

Once the Data Builder class is created it can be selected on the Data Builder Class dropdown on the Data tab of the template editor.

Localization

The Labels tab in the template editor lets you create text labels that can be translated for different languages.

To configure multi-language labels:

- Pick an object field that defines a record's language.
- Add supported locales: Click "Add Languages" to add the language locales you want to support
- Create labels: Add label names and their translations for each language.
- Reference in template: Use **`{{::labels.label_name}}`** pattern in your HTML to reference labels.

Note: The double colon (**`::`**) in label references differentiates them from merge fields, which use a single colon (**`:`**).

The system automatically selects the appropriate label based on the record's language field value. If no matching language is found, the label values from the locale marked as default are used.

Note: Save your labels before leaving the Labels tab, otherwise you may lose your work. We recommend using the Quick Save button after each label creation.

Language Field

Language [Language__c]

+ Add Language

Name	Locale Value	Custom Value	Default	
English (United States)	en_US	ENG	☑	▾
Spanish (Mexico)	es_MX		No	▾

+ Add Label

	Label Name	Locale	Label Value
▾	**invoice_title**	en_US	Invoice
		es_MX	Factura
▾	**order_number**	en_US	Order number
		es_MX	Número de pedido