ZDOCGenerator Template Guide V1.0

Generate Reports and Documents from Templates

1 Getting Started with ZDOCGenerator

ZDOCGenerator is a report/document generation api which allow user to generate reports and document using templates. ZDOCGenerator is a more complete solution that Separate content from presentation, this api produce documents in MS Word formats (docx) .

1.1 Introduction to ZDOCGenerator API

ZDOCGenerator is java based api which is build on top of APACHE POI. ZDOCGenerator allows you to create report and document using input template. You can pass input data through a variety of sources including XML, JSON, java object and publish your reports as DOCX.

1.2 What is template?

Template is typical MS Word documents(docx file) that may also contain ZDOCGenerator custom fields. ZDOCGenerator uses these custom fields to insert data,images,charts,barcode and mark the start and end of content for exclusion or repetition.

1.3 what is data source?

Data source can be XML,JSON or Java object which contains the information about the fields .

1.4 How does document generation work?

ZDOCGenerator transform template to document report using data source provided by user or application. ZDOCGenerator loads the template and merges it with data and creates the resulting document in the desired format(s). Data can be sourced from XML,JSON or Java Object .

2 Template Element

The basic building block of a template is the element. All elements are controlled by fields. ZDOCGenerator provides static text fields , Dynamic text fields, paired fields which has start and end tag, repetition fields which allow users to repeat the content, conditional fields for exclusion the row or content, anchor fields for inserting anchor tag into document, image field to insert the image, barcode field to insert barcode , variable fields .

2.1 Fields

In ZDOCGenerator there are two type of objects that can store value.

a) Field

b) Variable.

ZDOCGenerator uses these objects in data source (XML, JSON, Java Object) for fetching values. In order to use these objects in a template, they must be declared in template. After they have been declared in a template, the objects can be modified or updated during the report generation process.

A field is identified by a unique name, ZDOCGenerator determines the value for a field based on your data source. For example, when using XML or JSON to fill a report, ZDOCGenerator assumes that the name of the field matches the name of an element in XML or key in JSON. You must ensure that the field name in template match the field name in the data source.

2.1.1 Define Field

To render a field in document or report, you must define the field into template. To create a corresponding field you can refer the following syntax. You can specify a field object inside a template, using the following syntax:

<<field-name>>

Example:

Field inside Template : <<documentName>>

Field inside XML Data Source : <doocumentName>HR Policy GuideLine</documentName>

Field inside JSON Data Source : {"doocumentName":"HR Policy GuideLine"}

Download Example

2.1.2 Define Variables

You can use variables to store partial results and do complex calculations with the data extracted from a data source. You can then use these values in other parts of the report, including other variables.

Variable element always starts from **<<@** , you can assign String, Number, Float ,Double ,Long, Boolean ,List,Map and other element also into this variable.

For Creating Map variable element always starts from **<<@map** , you can assign map data into Map variable.

For creating List variable element always starts from **<<@list** , you can assign list data into list variable.

Example :

<<@documentCount=4>>

Assign the number 4 to variable @documentCount

<<@documentStatus=true>>

Assign the Boolean value true to variable @documentStatus

<<@documentName='Policy Guide'>>

Assign the string value to variable @documentName. Append single quotes into string value otherwise it will look up for the data source for data associated with "Policy Guide" reference.

<<@documentName=docName>>

Will look inside the data source for data associated with "docName" and assign it to documentName variable.

Download Example

<<@mapDocuments={docName=policy, author=Sam, version=1.0, type=nothing}>>

Assign the values into map element.

Map Function

Function	Description
get	<<@mapDocuments get 'docName'>> Will fetch the value from map @mapDocuments using 'docName' key
size	<<@mapDocuments size>> Will fetch the size of map @mapDocuments

Download Example

<<@listDocuments={policy Hr, Policy Leave, Policy Company }>>

Assign the values into List element.

List Function

Function	Description
get	<<@listDocuments get 0>> Will fetch the value from list @listDocuments using position .
size	<<@listDocuments size>> Will fetch the size of list @listDocuments

Download Example

2.1.3 Defining Expressions

An expression is a formula that operates on some values and returns a result, in ZDOCGenerator you use expression to render its computed value, inclusion and exclusion the content of paragraph, numbering and table row.

Expression element always starts with **<<{** and ends with **}>>**, will render the computed result of given expression.

2.1.3.1 Expression Operators

Operator	Description	Example
+	Add (it can be used to Add two numbers)	<<{ documentCount + 4}>> Will look inside the data source for data associated with "documentCount" and add 4 into that. If documentCount value is 2 then output will be 6.

		<<@docCountVar = { documentCount + 4}>> Will calculate the expression and assign it to variable docCountVar
-	Subtraction (it can be used to Minus two numbers)	<<{ documentCount - 4}>> Will look inside the data source for data associated with "documentCount" and minus 4 into that. If documentCount value is 10 then output will be 6. <<@docCountVar = { documentCount - 4}>> Will calculate the expression and assign it to variable docCountVar
*****	Multiplication	<<{ documentCount * 4}>> Will look inside the data source for data associated with "documentCount" and multiply 4 into that. If documentCount value is 2 then output will be 8. <<@docCountVar = { documentCount * 4}>> Will calculate the expression and assign it to variable docCountVar
/	Division	<<{ documentCount / 4}>> Will look inside the data source for data associated with "documentCount" and divide by 4. <<@docCountVar = { documentCount / 4}>> Will calculate the expression and assign it to variable docCountVar
gt	greater than (for numbers)	<<{ documentCount gt 4}>> Will look inside the data source for data associated with "documentCount" and if

		documentCount value is greater than 4 then result will true otherwise false . <<@docStatusVar = { documentCount gt 4}>> Will calculate the expression and assign it to variable docStatusVar
lt	less than (for numbers)	<<{ documentCount lt 4}>> Will look inside the data source for data associated with "documentCount" and if documentCount value is less than 4 then result will true otherwise false . <<@docStatusVar = { documentCount lt 4}>> Will calculate the expression and assign it to variable docStatusVar
gteq	greater than or equal (for numbers)	<<{ documentCount gteq 4}>> Will look inside the data source for data associated with "documentCount" and if documentCount value is greater than or equal 4 then result will true otherwise false . <<@docStatusVar = { documentCount gteq 4}>> Will calculate the expression and assign it to variable docStatusVar
lteq	less than or equal (for numbers)	<<{ documentCount lteq 4}>> Will look inside the data source for data associated with "documentCount" and if documentCount value is less than or equal 4 then result will true otherwise false . <<@docStatusVar = { documentCount lteq 4}>> Will calculate the expression and assign it to variable docStatusVar
eq	equal (for numbers and strings)	<<{ documentName eq 'Policy Doc'}>> Will look inside the data source for data associated with "documentName" and if

		documentName value is equal to 'Policy Doc' then result will true otherwise false. <<{ documentCount eq 4}>> Will look inside the data source for data associated with "documentCount" and if documentCount value is equal to 4 then result will true otherwise false . <<@docStatusVar = { documentCount eq 4}>> Will calculate the expression and assign it to variable docStatusVar
notEqual	not equal (for numbers and strings)	<<{ documentName notEqual 'Policy Doc'}>> Will look inside the data source for data associated with "documentName" and if documentName value is not equal to 'Policy Doc' then result will true otherwise false. <<@docStatusVar = { documentName notEqual 'Policy Doc'}>> Will calculate the expression and assign it to variable docStatusVar

Download Example

2.1.3.2 Functions

Function	Description	Example
equalsIgnoreCase	Equal Ignore Case This method compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case, if they	<<{ documentName equalsIgnoreCase 'Policy Doc'}>> Will look inside the data source for data associated with "documentName" and if documentName value is equal to 'Policy Doc'

	are of the same length, and corresponding characters in the two strings are equal ignoring case.	ignoring case considerations then result will true otherwise false. <<@docStatusVar = { documentName equalsIgnoreCase 'Policy Doc'}>> Will calculate the expression and assign it to variable docStatusVar
charAt	The method charAt returns the character at the specified index. The index value should lie between 0 and length()-1.	<<{ documentName charAt 0}>> Will look inside the data source for data associated with "documentName" and if documentName value is 'Policy Doc' then result will be 'P'. <<@docVar = { documentName charAt 0}>> Will calculate the expression and assign it to variable docVar
length	The length method returns the number of characters presents in the string.	<<{ documentName length }>> Will look inside the data source for data associated with "documentName" and if documentName value is 'Policy' then result will be 6. <<@docCountVar = { documentName length }>> Will calculate the expression and assign it to variable docCountVar
startsWith	The method startsWith is used for checking prefix of a String. It returns a boolean value true or false based on whether the specified string is prefix of the particular String or not.	<<{ documentName startsWith 'Policy'}>> Will look inside the data source for data associated with "documentName" and if documentName value starts with 'Policy' then result will true otherwise false. <<{ documentName startsWith suppDocName}>> Will look inside the data source for data associated with "documentName" and "suppDocName", if "documentName" value starts with "suppDocName" value then result will true otherwise false.

		<<@docStatusVar = { documentName startsWith suppDocName}>> Will calculate the expression and assign it to variable docStatusVar
endsWith	The endsWith method is used to check whether the string is ending with user-specified substring or not. Based on this comparison it will return boolean true or false.	<<{ documentName endsWith 'Policy'}>> Will look inside the data source for data associated with "documentName" and if documentName value ends with 'Policy' then result will true otherwise false. <<{ documentName endsWith suppDocName}>> Will look inside the data source for data associated with "documentName" and "suppDocName" , if "documentName" value ends with "suppDocName" value then result will true otherwise false. <<@docStatusVar = { documentName endsWith suppDocName}>> Will calculate the expression and assign it to variable docStatusVar
toLowerCase	The method toLowerCase converts the characters of a String into lower case characters.	<<{ documentName toLowerCase }>> Will look inside the data source for data associated with "documentName" and convert all character to lower case. <<@docVar = { documentName toLowerCase }>> Will calculate the expression and assign it to variable docVar
toUpperCase	The toUpperCase method returns the string in uppercase letter. it converts all characters of the string into upper case letter.	<<{ documentName toUpperCase }>> Will look inside the data source for data associated with "documentName" and convert all character to upper case. <<@docVar = { documentName toUpperCase }>> Will calculate the expression and assign it to variable docVar

contains	The contains method to check if String contains another substring or not. It returns boolean value.	<<{ documentName contains 'Policy'}>> Will look inside the data source for data associated with "documentName" and check if "documentName" value contains 'Policy' or not and return true or false based calculation. <<@docStatusVar = { documentName contains 'Policy'}>> Will calculate the expression and assign it to variable docStatusVar
append	The append method appends the specified string to another string.	<<{ documentName append 'Policy'}>> Will look inside the data source for data associated with "documentName" and append 'Policy' to "documentName" value . <<@docVar = { documentName append 'Policy'}>> Will calculate the expression and assign it to variable docVar
reverse	This reverse method reverses the value of the String.	<<{ documentName reverse }>> Will look inside the data source for data associated with "documentName" and reverse "documentName" value . <<@docVar = { documentName reverse }>> Will calculate the expression and assign it to variable docVar
sort	This method sorts all the words into string in ascending order.	<<{ documentName sort }>> Will look inside the data source for data associated with "documentName" and sort "documentName" value in ascending order. <<@docVar = { documentName sort }>>

		Will calculate the expression and assign it to variable docVar
replaceAll	This method returns a new string resulting from replacing every occurrence of string with a new string.	<<{documentName replaceAll 'Policy' 'HR'}>> Will look inside the data source for data associated with "documentName" and replace all 'Policy' occurrence with 'HR'. <<{documentName replaceAll subDocName refDocName}>> Will look inside the data source for data associated with "documentName","subDocName","refDocName" and replace all "subDocName" value occurrence with "refDocName" value . <<@docVar = {documentName replaceAll subDocName refDocName}>> Will calculate the expression and assign it to variable docVar
indexOf	The indexOf method the index of the first occurrence of character in a String	<<{documentName indexOf 'P'}>> Will look inside the data source for data associated with "documentName" and find the first occurrence of character 'P'. <<@docCountVar = { documentName indexOf 'P' }>> Will calculate the expression and assign it to variable docCountVar
substring	The substring method returns a part of the string. You can pass begin index and end index number position in the substring method where start index is inclusive and end index is exclusive.	<<{documentName substring 1 7}>> Will look inside the data source for data associated with "documentName" and return the substring between index 1 and 7. <<{documentName substring 4}>> Will look inside the data source for data associated with "documentName" and return the substring between index 4 and string length -1. <<@docVar = {documentName substring 1 7}>>

		Will calculate the expression and assign it to variable docVar
lastIndexOf	The lastIndexOf method returns last index of the given character value.	<<{documentName lastIndexOf 'P'}>> Will look inside the data source for data associated with "documentName" and find the last occurrence of character 'P'. <<@docCountVar = { documentName lastIndexOf 'P' }>> Will calculate the expression and assign it to variable docCountVar
replaceFirst	This method returns a new string resulting from replacing first occurrence of string with a new string.	<<{documentName replaceFirst 'Policy' 'HR'}>> Will look inside the data source for data associated with "documentName" and replace first 'Policy' occurrence with 'HR'. <<{documentName replaceFirst subDocName refDocName}>> Will look inside the data source for data associated with "documentName","subDocName","refDocName" and replace first "subDocName" value occurrence with "refDocName" value . <<@docVar = {documentName replaceFirst subDocName refDocName}>> Will calculate the expression and assign it to variable docVar
delete	This method removes the characters in a substring of this String. The substring begins at the specified start and end index.	<<{documentName delete 1 7}>> Will look inside the data source for data associated with "documentName" and delete the substring between index 1 and 7 , render the remaining string. <<@docVar = {documentName delete 1 7}>> Will calculate the expression and assign it to variable docVar

deleteAt	The deleteAt method removes the char at the specified position in given string.	<<{documentName deleteAt 1 }>> Will look inside the data source for data associated with "documentName" and delete the character at index 1 , render the remaining string. <<@docVar = {documentName deleteAt 1 }>> Will calculate the expression and assign it to variable docVar
insert	This method inserts the string into another string at particular index.	<<{documentName insert 3 'Policy'}>> Will look inside the data source for data associated with "documentName" and insert the String 'Policy' at index 3 , render the remaining string. <<@docVar = {documentName insert 3 'Policy'}>> Will calculate the expression and assign it to variable docVar
titleCase	this method changes the string so that the first character of each word is a capital letter.	<<{documentName titleCase }>> Will look inside the data source for data associated with "documentName" and change the string so that the first character of each word is a capital letter. <<@docVar = {documentName titleCase }>> Will calculate the expression and assign it to variable docVar
abs	The method gives the absolute value of the number	<<{documentNumber abs }>> Will look inside the data source for data associated with "documentNumber" and give the absolute value of given number <<@docCountVar = {documentNumber abs}>> Will calculate the expression and assign it to variable docCountVar
ceil	The ceil returns the smallest (closest to negative infinity) double value that is greater than or equal to the number.	<<{documentNumber ceil }>> Will look inside the data source for data associated with "documentNumber" and returns the smallest (closest to negative infinity) double

		value that is greater than or equal to the number. <<@docCountVar = {documentNumber ceil }>> Will calculate the expression and assign it to variable docCountVar
floor	The floor method returns the largest (closest to positive infinity) double value that is less than or equal to the number.	<<{documentNumber floor }>> Will look inside the data source for data associated with "documentNumber" and returns the largest (closest to positive infinity) double value that is less than or equal to the number. <<@docCountVar = {documentNumber floor }>> Will calculate the expression and assign it to variable docCountVar
max	The max method returns maximum of two numbers.	<<{max documentNumber downloadCount }>> Will look inside the data source for data associated with "documentNumber" ,"documentCount"and return maximum of two numbers . <<@docCountVar = { max documentNumber downloadCount }>> Will calculate the expression and assign it to variable docCountVar
min	The min method returns the smaller of two numbers.	<<{min documentNumber downloadCount }>> Will look inside the data source for data associated with "documentNumber" ,"documentCount"and return minimum of two numbers . <<@docCountVar = { min documentNumber downloadCount }>> Will calculate the expression and assign it to variable docCountVar
pow	The pow method is used to calculate a number raise to the power of some other number.	<<{pow documentNumber downloadCount }>> Will look inside the data source for data associated with "documentNumber" ,"documentCount". Suppose "documentNumber" value is 10 and

		"documentCount" value is 2 then it return 10 to power of 2. So output will be 100. <<@docCountVar = { pow documentNumber downloadCount }>> Will calculate the expression and assign it to variable docCountVar
round	The round method returns the closest number to the argument.	<<{ documentNumber round }>> Will look inside the data source for data associated with "documentNumber" and returns the closest number to the "documentNumber" value. <<@docCountVar = { documentNumber round }>> Will calculate the expression and assign it to variable docCountVar
sqrt	The sqrt method returns the square root of a number.	<<{ documentNumber sqrt }>> Will look inside the data source for data associated with "documentNumber" and returns the square root of "documentNumber" value. <<@docCountVar = { documentNumber sqrt }>> Will calculate the expression and assign it to variable docCountVar
dateFormat	The dateFormat method change the format of date based on the given format. User can provide date value, current format and output format.	<<{ publishDate dateFormat 'DD-MM-YYYY' 'YYYY-MM-DD'}>> Will look inside the data source for data associated with "publishDate" and change the format of date to 'YYYY-MM-DD'. <<@docCountVar = { publishDate dateFormat 'DD-MM-YYYY' 'YYYY-MM-DD'}>> Will calculate the expression and assign it to variable docCountVar

Download Example

2.1.3 Non Mandatory Field

This is non mandatory element, non mandatory element always starts with **<<nm_**

Example:

<<nm_documentNumber>>

if "documentNumber" is not available in data source then this element will be removed from report , if "documentNumber" is available in data source then this element will be replaced by data referenced into data source.

Consider the below given template case

Template

Document Name : <<documentName>>

Document Number : <<nm_documentNumber>>

Document Version : <<documentVersion>>

If there is no value for "documentNumber", the output using the above template would look like the following:

Report Output

Document Name : Policy HR

Document Number :

Document Version : 1.0

If entire paragraph contains only non mandatory field and non mandatory field does not available into data source then entire paragraph will be removed .

Template

<<documentName>>

<<nm_documentNumber>>

<<documentVersion>>

If there is no value for "documentNumber", the output using the above template would look like the following:

Report Output

2.1.4 Anchor Field

The element is used insert an anchor link dynamically into report or document. Use the below given field expression to insert anchor link into your report.

<<anchor_documentUrl>>

Will look inside the data source for data associated with “documentUrl” for displayed text for your anchor tag. For anchor url, will look inside the data source for "url" attribute for data associated with “documentUrl”.

Data Source : <documentUrl url="http://www.google.com">Google</documentUrl >

In given data source url value is “*http://www.google.com*" and display value is Google so final output in report will be like:

Report Output:

Google

DOWNLOAD EXAMPLE

2.1.5 Image Field

The element is used insert image at any locations in documents. will look data source field attribute values for width height and image path. Use the below given field expression to insert image into your report.

<<img_imagepath>>

Example :

Field : <<img_imagepath>>

Xml Data Source : <imagepath width="300" height="100">C:\Users\ Desktop\image.jpg</imagepath>

ZDOCGenerator will insert image at given location with 300 px width and 100 px height.

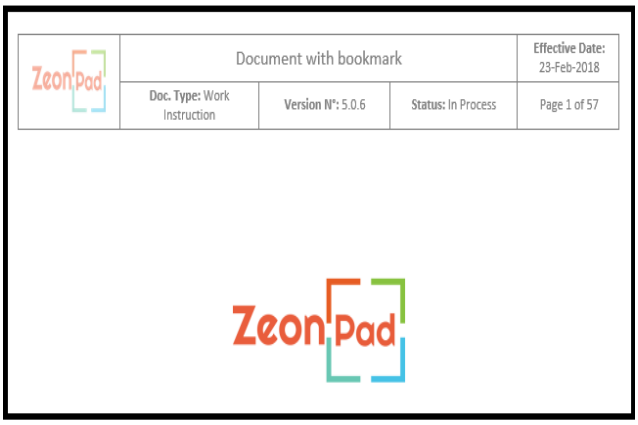
Template:

<<documentName>>

<<img_imagepath>>

Report Output:

Policy HR



Download Exmaple

2.1.6 Conditional field for Section and Paragraph

Conditional element is used for exclusion and inclusion of content inside paragraph an d section in final report. If condition is met then the content inside the section or paragraph is rendered in final report. If condition is not met then the content inside the section or paragraph is removed in final report.

Conditional field is defined using a pair of fields , start field and end field. Start field starts with **<<zifs_{** and ends with **}>>** , start field value is **<< zifs_{some expression}>>** end field value is **<<zife>>.**

The general syntax for a conditional section is:

<<zifs_{some expression}>>

Lorem Ipsum Lorem Ipsum Lorem Ipsum Lorem Ipsum

<<zife>>

The conditional field start and end tags are removed from the resulting document and if each tag is on a paragraph by itself, the entire paragraph will be removed.

Rules to add conditional field:

1. Insert the opening condition field into the empty paragraph or in middle of paragraph.
2. Add the content and other fields into the subsequent paragraphs in the template .
3. Insert the closing condition field into an empty paragraph following the conditional content or in middle of paragraph.

4. Repeat the 1 to 3 steps for adding multiple conditional field.

Example 1:

<<documentName>><<zifs_{ documentCount lt 5 }>> This document is published by author Author Name : <<documentAuthor>> <<zife>> Version Number : <<docVerison>>

If condition is met then the output will be:

Policy HR This document is published by author Author Name : Sam Version Number : 1.0

If condition is not met then the output will be:

Policy HR Version Number : 1.0

Example 2:

<<documentName>>

<<zifs_{ documentCount lt 5 }>>

This document is published by author Author Name : <<documentAuthor>>

<<zife>>

Version Number : <<docVerison>>

If condition is met then the output will be:

Policy HR

This document is published by author Author Name : Sam

Version Number : 1.0

If condition is not met then the output will be:

Policy HR

Version Number : 1.0

DOWNLOAD EXAMPLE

2.1.7 Repeating field for Section and Paragraph.

Repeating sections are groups of fields and plain text that may need to be repeated. Repeating sections repeat the components within the section of the layout based on the occurrence of an element in the data source. Repeating sections are used to create classic banded reports.

Repeating field is defined using a pair of fields, start field and end field. Start field starts with **<<msec_{** and ends with **}>>** , start field value is **<<msec_repeating-star-field-name>>** end field value is **<<msecend>>**.

The general syntax for a repeating section is:

<<msec_repeating-section-name>>

Lorem Ipsum Lorem Ipsum Lorem Ipsum Lorem Ipsum <<docName>>

<<msecend>>

The repeating sections start and end tags are removed from the resulting document and if each tag is on a line by itself, the entire line will be removed.

Rules to create a repeating section:

1. Position the insertion point in an empty paragraph at the starting location of the repeating section.
2. Insert the opening repeating section element into the empty paragraph.
3. Add the boilerplate content and other ZDOCGenerator elements into the subsequent Paragraphs in the document.
4. Insert the closing element into an empty paragraph following the repeated content.

Repeating sections also provide access to some built-in variables such as $secCount which is the current count of the number of times the loop has been repeated starting from one.

Example.

<<msec_documents>>

<<$secCount>>. <<docName>>

<<msecend>>

To produce output that might look like this:

1. Leave Policy

2. Reimbursement Policy

3. Travel policy

Example 2:

<<@mapDocuments={docName=policy, author=Sam, version=1.0, type=nothing}>>

<<msec_@mapDocuments>>

<<$secCount>>. <<$key>> <<$value>>

<<msecend>>

To produce output that might look like this:

1. docName policy

2. author Sam

3. version 1.0

4. type nothing

<<$key>> and <<$value>> are in build variable for map.

Example 3.

<<@listDocuments={policy Hr, Policy Leave, Policy Company }>>

<<msec_@listDocuments >>

<<$secCount>>. <<$value>>

<<msecend>>

To produce output that might look like this:

1. policy Hr

2. Policy Leave

3. Policy Company

DOWNLOAD EXAMPLE

2.1.8 Conditional field for Table's Row

Conditional element is used for exclusion and inclusion of row inside table in final report. If condition is met then the row inside the table is rendered in final report. If condition is not met then the row inside the table is removed in final report.

Conditional field is defined using a pair of fields, start field and end field. Start field starts with **<<zifs_{** and ends with **}>>** , start field value is **<< zifs_{some expression}>>** end field value is **<<zife>>**.

The general syntax for a conditional section is:

Document Name	Version Number	Document Author
<<zifs_{some expression}>>		
<<docName>>	<<docVersion>>	<<docAuthor>>
<<zife>>		

The conditional field start and end tags are removed from the resulting document and if each tag is on a row by itself, the entire row will be removed.

Rules to add conditional field:

1. Insert the opening condition field into the empty row.
2. Add the content and other fields into the subsequent rows in the template.
3. Insert the closing condition field into an empty row following the conditional content
4. Repeat the 1 to 3 steps for adding multiple conditional field.

Example 1:

Document Name	Version Number	Document Author
<<zifs_{documentCount lt 5}>>		
<<docName>>	<<docVersion>>	<<docAuthor>>
<<zife>>		

If condition is met then the output will be:

Document Name	Version Number	Document Author
Hr Policy	1.0	John Snow

Example 2:

Document Name	Version Number	Document Author
<<zifs_{documentCount lt 5}>>		
<<docName>>	<<docVersion>>	<<docAuthor>>
<<zife>>		

If condition is not met then the output will be:

Document Name	Version Number	Document Author

DOWNLOAD EXAMPLE

2.1.9 Conditional field for Table's Columns

Conditional element is used for exclusion and inclusion of column inside table in final report. If condition is met then the column inside the table is rendered in final report. If condition is not met then the column inside the table is removed in final report.

Conditional field starts with **<<zclmc_{** and ends with **}>>** .

The general syntax for a conditional section is:

<< zclmc_{some expression}>>

Insert the condition field into the column of top row

Example 1.

Document Name	Document Version	Document Author **<< zclmc_{documentCount lt 5 }>>**	Effective Date
Policy HR	1.0.0.0	John Snow	12-02-2018
Travel Policy	2.0.0.0	White walker	12-02-2018

Emp Policy	1.0.0.0	John Snow	12-02-2018

If condition is not met then the output will be:

Document Name	Document Version	Effective Date
Policy HR	1.0.0.0	12-02-2018
Travel Policy	2.0.0.0	12-02-2018
Emp Policy	1.0.0.0	12-02-2018

Example 2.

Document Name	Document Version	Document Author **<< zclmc_{documentCount lt 5 }>>**	Effective Date
Policy HR	1.0.0.0	John Snow	12-02-2018
Travel Policy	2.0.0.0	White walker	12-02-2018
Emp Policy	1.0.0.0	John Snow	12-02-2018

If condition is met then the output will be:

Document Name	Document Version	Document Author	Effective Date
Policy HR	1.0.0.0	John Snow	12-02-2018
Travel Policy	2.0.0.0	White walker	12-02-2018
Emp Policy	1.0.0.0	John Snow	12-02-2018

DOWNLOAD EXAMPLE

2.1.10 Repeating field for Table's row.

Repeating row are groups of fields and plain text that may need to be repeated. Repeating row repeats the components within the table of the layout based on the occurrence of an element in the data source. Repeating row are used to create classic banded reports.

Repeating field is defined using a pair of fields, start field and end field. Start field starts with **<<mrow_** and ends with **>>** , start field value is **<<mrow_repeating-start-field-name>>** end field value is **<<mrowend>>**.

The general syntax for a repeating section is:

<<mrow_repeating-row_name>>		
<<somefield1>>	<< somefield2>>	<< somefield3>>
<<mrowend>>		

The repeating rows start and end tags are removed from the resulting document and if each tag is on a line by itself, the entire line will be removed.

Rules to create a repeating row:

1. Insert the opening repeating row element into the empty row.

2. Add the boilerplate content and other ZDOCGenerator elements into the subsequent rows in the document.

3. Insert the closing element into an empty row following the repeated content.

Repeating rows also provide access to some built-in variables such as $rowCount which is the current count of the number of times the loop has been repeated starting from one.

Example 1.

Document Name	Document Version	Document Author	Effective Date
<<mrow_documents>>			
<<docName>>	<<docVersion>>	<<docAuthor>>	<<docEffDate>>
<<mrowend>>			

Output

Document Name	Document Version	Document Author	Effective Date
Policy HR	1.0.0.0	John Snow	12-02-2018
Travel Policy	2.0.0.0	White walker	12-02-2018
Emp Policy	1.0.0.0	John Snow	12-02-2018

DOWNLOAD EXAMPLE

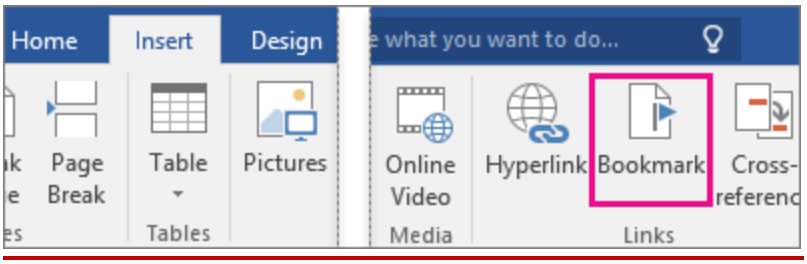
2.2 Bookmark Elements

ZDOCGenerator replace bookmark inside the document.

Step to bookmark.

Select text in your template where you want to insert a bookmark.

Click **Insert** > **Bookmark**.



Under **Bookmark name**, type a name and click **Add**.

Bookmark names need to begin with **bm_**. They can include both numbers and letters, but not spaces. If you need to separate words, you can use an underscore (_).

Example:

bm_documentName

documentName value is available in source data.

2.3 Barcode Elements

ZDOCGenerator can generate barcodes and insert them into your output document. User can specify height , line width ,draw text value , draw border property in barcode/

Supported bar code format.

a) Code128
b) Code128a
c) Code39
d) Code25

e) Codebar

Example :

<<code128_docBarCode>>

<<code128a_docBarCode>>

<<code39_docBarCode>>

<<code25_docBarCode>>

DOWNLOAD EXAMPLE