Docmosis Template Guide
Version 2.0

Fast documents from Java!

Docmosis Template Guide

Copyrights

© 2009 Systemic Pty Ltd

This document and all human-readable contents of the Docmosis distribution are the copyright of Systemic Pty Ltd. You may not reproduce or distribute any of this material without the written permission of Systemic.

`http://www.docmosis.com`

The placeholder image provided in the Docmosis distribution is intended for use in document templates and is not restricted by the terms above. You may use the image for the development of document templates and distribute it as required.

Trademarks

Microsoft Word and MS Windows are registered trademarks of the Microsoft Corporation.

`http://office.microsoft.com/en-us/default.aspx`

`http://www.microsoft.com/windows/`

OpenOffice.org is a trademark of OpenOffice.org.

`http://www.openoffice.org`

Adobe® PDF is a trademark of the Adobe Corporation.

`http://www.adobe.com/products/acrobat/adobepdf.html`

Contents

Preface

Welcome to the *Docmosis Template Guide*. This manual is intended for document template developers who will create richly formatted document layouts with the special embedded features that enable Docmosis to produce documents using data generated by Java applications.

The *Docmosis Template Guide* provides information on developing document templates (in either MS Word or OpenOffice.org Writer) that will be used to produce printable documents. This guide assumes a level of competence in using each word processor and is not a reference manual for either.

Note

Don't worry. If you are competent with only one of the two word processors, you don't have to know how to use the other. In general the activities to develop the templates are the same for both tools but where there are differences between the two, this document highlights them and describes the activities for each application.

Conventions used in this guide

This document uses typographical conventions that highlight significant parts of the text to distinguish it from normal text.

Text that looks like this…	Means this…
«fieldname»	A field in the document template. In Word it is a mail merge field. In OpenOffice Writer it is an input field.
docmosis.###	A code instruction: either an individual line, or part of a complete module.
↳	A symbol to show that the line of code has wrapped due to the space restrictions on the page. You should remove this symbol from your code if you copy and paste code snippets from this document.
...	An indicator to signify that the preceding sequence of code instructions will execute incrementally until there is no more data in the data provider.
template.doc	A file name, a file extension or a Web site address.

Table 1: Typographical conventions

Additionally, some parts of the document are written specifically for one of the word processors mentioned. When this is the case, the paragraph has the respective icon in the left margin.

This icon…	Means this…
	The information applies to MS Word only.
	The information applies to OpenOffice.org Writer only.

Table 2: Graphical conventions

Special paragraphs

In addition to the text conventions, certain information is presented in a specific way to emphasise their information.

Note

A note provides additional supporting information that will help you to understand a point that the author is trying to make.

Important

Executed code on a computer rarely cause damage to hardware but may well corrupt data. Information in this form is intended to alert you to the potential for data corruption.

Tip

A tip provides anecdotal information to support the technical information about using the system and might be useful in helping you understand the information being presented.

General Terminology

Several terms are used in this document to identify elements of a document template. The following table provides definitions for those terms. Note that the definitions are general: for more details on these terms, you should refer to the online Help for the particular word processor with which you are developing the templates.

This term...	describes this document element...
field	a placeholder that is used by Docmosis to substitute data or control document flow
boilerplate	graphical and textual content that is added to a template as reusable content to avoid having the document developer recreate the content for each document. Docmosis uses boilerplate components.
header and footer	elements of a printed document that repeat on every page. Information in these elements is usually administrative information about the document.

Table 3: Document terminology

1 Introduction

Docmosis is an easy-to-use document processor which integrates with your Java application to provide the seamless production of printable documents using data generated by the application.

The documents are produced in one or more of the following formats: MS Word; Open Office Writer; Adobe PDF and HTML using your custom-made templates. Docmosis delivers a great reporting engine by providing the following:

- cross platform compatibility;
- portability;
- scalability.

In this chapter, we provide information about the main features of a Docmosis template. Details for incorporating elements and Docmosis logic structures into a template are provided in Chapter 2, Developing Docmosis templates.

1.1 Separating content from presentation

Developing applications that contain presentation logic means that when an organisation's brand image changes (such as a new company logo, different corporate font or company name), so must all the applications. Using Docmosis, all of your presentational features are developed separately from your application in commonly used word processors. This has two distinct benefits:

- The initial development of the document templates can be assigned to those who are experts in that field and they can be developed using one of the two most commonly used word processors; and
- Branding changes do not require software development support, which can be time consuming and expensive.

In addition to these benefits, Docmosis is fast: the core document processor can produce hundreds of documents in minutes in the most popular formats, which is a great improvement on other document processors currently available.

1.2 What are templates?

As far as Docmosis is concerned, templates are typical MS Word or OpenOffice.org Writer documents that may also contain fields. Docmosis uses fields to insert data, and mark the start and end of content for exclusion or repetition. Fields are standard features of these two document tools meaning Docmosis does not require any custom plugins. Docmosis supports a wide range of versions (MS Word 97 onwards, OpenOffice.org Writer 2.2 onwards).

As well as using fields to drive Docmosis, templates in MS Word and OpenOffice.org Writer give ideal control over aspects such as:

- page size, margins, and columns;

- information in running headers and footers;

- typographic characteristics that describe paragraph and character styles; and

- boilerplate text, graphics and embedded field codes.

Documents created using a template inherit this pattern when they are created and no programmatic effort at all is required to benefit from these features. Once the document is rendered, it has no connection to the template from which it came other than that it is based on its characteristics. Templates can be modified as required without any concern to the documents that have been rendered in the past.

Tip

In addition, as the paragraph and character patterns for the output are described in simple formatting terms, there is no real need to develop paragraph and character styles. However, if you are comfortable using styles, it might be advantageous to implement them for future template updates.

1.3 Where are the templates stored?

You can provide your templates to Docmosis in many different ways: from directories, files or Jar files (or if you are a developer, from any InputStream). Docmosis manages a "store" of its own templates, which acts as a cache for the templates. The cache enables Docmosis to analyse the template and to optimise it for later rendering. For more information about registering templates, refer to the *Docmosis Developer's Reference*.

1.4 How does document generation work?

In the simplest terms, Docmosis seamlessly integrates the data available in the application with the element structures in the template to produce documents that may be:

- stored electronically, printed, viewed or any combination of these; and

- published in several document formats.

During document generation, Docmosis loads the template from the template store, merges it with data and creates the resulting document in the desired format(s). Data can be sourced from any combination of locations (databases, files, Java Objects etc), templates can include/exclude any content, tables can grow and shrink and images can be embedded. These features are discussed later in this guide.

If the template document includes an index or table of contents, Docmosis will automatically update these tables in the resulting document.

1.5 Template features

Modern word processors enable the development of documents with support for high-quality typesetting and layouts incorporating inline images. By inheriting these features automatically, Docmosis provides the developer and template author with an extremely powerful automatic document generation capability. Other aspects of the document that are relevant to Docmosis are detailed in this section.

In this section:

- **General features.** Information about the general document features that can be used to create and exploit to deliver high-quality layouts;

- **Advanced features.** Details on the Docmosis features that can be incorporated into your template; and

- **Docmosis elements.** Details of the Docmosis elements that interact with the document generation process.

1.5.1 General features

Many of the integration aspects are achieved simply by using well known documentation techniques. Docmosis understands elements such as lists and tables, so there is no need to learn new techniques to develop templates for use with Docmosis.

The general aspects of a document template include the creation of branding features, layout features and static text elements. In order to create a document template for use with Docmosis, the template author can simply create a document and direct Docmosis using the following features in your word processor:

Word Feature	Writer Feature	Controls
Mail merge fields	Input fields	Data insertion and document flow
Bookmarks	Image properties	Image insertion

1.5.2 Advanced features

To generate sophisticated documents of value to your application, fields are interpreted by Docmosis. These fields can direct Docmosis providing:

- Insertion of text or image data into the body, headers and footers and tables;

- Inclusion or exclusion of content;

- Repeating of content;

- Table row repetition or exclusion;

- Table column exclusion;

- Numbered and bullet list expansion.

> **Tip**
> You might also use other fields that reference the content of the generated elements. For example a *StyleRef* field in a document header that reflects a generated title in the text.

1.5.3 Docmosis elements

All Docmosis "elements" are controlled by fields, except for image insertion which is controlled by bookmarks or image properties. Each element is discussed in detail in this guide. In general, elements may be singular (such as a text insertion) or may be paired having a start and end marker.

1.5.3.1 Field Reference

The following table provides a quick reference to the elements and their syntax. The names of the fields must match exactly for the document generation to succeed.

Element	Description	Closing Element
«name»	Replace this field by whatever data is referenced by "name".	
«$abc=name»	Lookup the data associated with "name" and assign it to the variable "abc".	
«$abc»	Lookup the variable "abc" and render its value	
«cs_name» «cs_{expr}» «cs_$abc»	All content up to the closing element is included or excluded depending on the value associated with "name" or the expression "expr" or the variable "abc".	«es_name» «es_{expr}» «es_$abc» «es_»
«rs_name» «rs_$abc»	All content up to the closing element is repeated whilst there is data associated with "name" or the variable "abc"	«es_name» «es_$abc» «es_»
«cr_name» «cr_{expr}» «cr_$abc»	Include the following table rows depending on the value associated with "name" or expression "expr" or the variable "abc".	«er_name» «er_{expr}» «er_$abc» «er_»
«rr_name» «rr_$abc»	Repeat the following table rows whilst there is data associated with "name" or the variable "abc".	«er_name» «er_$abc» «er_»
«cc_name» «cc_{expr}» «cc_$abc»	Include or exclude the table column containing this field depending on the value associated with "name" or the expression "expr" or the variable "abc".	
Image bookmarked with label "bm_name"	Replace the image with the image data associated with "name".	

Table 4: Docmosis element quick reference

Note

Care must be taken with all fields when using MS Word documents as templates. You MUST AVOID SPACES in a field name as a space will truncate the field and sometimes Docmosis will not be able to detect this. The good news is that most of the time this will turn up as an obvious error reported during document generation.

1.5.3.2 Expressions

The following table shows the valid expressions that Docmosis supports. The examples are for a conditional section ("cs_") but apply to conditional columns also.

Element	Description
«cs_{a<10}»	Lookup data element "a" and see if it is **less than** 10 numerically. If "a" is not numeric, a string comparison is performed automatically.
«cs_{a='fred'}»	Lookup data element "a" and see if it is **equal to** the String literal "fred".
«cs_{$a!=10»	Lookup the variable "a" and see if it is **not equal to** the numeric value 10. If variable "a" does not resolve to a numeric value, a String comparison is performed.
«cs_{!hasElements()}»	Lookup the data element "hasElements()" and **boolean negate** the result. If hasElements() returns something other than a boolean, the result will be evaluated using a best effort (eg a String value of "true" would resolve to true). Note that the use of the brakets "( )" typically implies the data is to be sourced by calling a Java function literally.
Other operators include <= >=	 Less than or equal to Greater than or equal to

1.5.3.3 Nesting

Elements can be "nested" with regards to the way they lookup data. For example, «hotel.floor» typically would refer to the floor within a hotel object. The period "." character represents the delimiter between one level of data and the next. This is described in detail later.

1.5.3.4 Range Specifiers

Data elements can also be referenced by **ranges** of values Docmosis should lookup. This provides a fair amount of power within the template to select the values of interest. It depends on the context of the element as to whether it is allowed to produce multiple values (and Docmosis will flag errors where inappropriate use is made). For example, a repeating section is expected to produce multiple values, but a simple lookup field is not.

The following table details the types of range specifier available.

Element	Description
«hotel[0]»	The first hotel (indexing starts at zero)
«hotel[F]»	The first hotel (equivalent to index zero)
«hotel[L]»	The last hotel
«hotel[*]»	All hotels
«hotel[F3]»	The first 3 hotels

Element	Description
«hotel[L3]»	The Last 3 hotels
«hotel[1,2,4]»	The hotels at indexes 1,2 and 4
«hotel[1-3,L2]»	The hotels at indexes 1 to 3 inclusive and the last 2
«hotel[0-L2]»	All but the last 2 hotels
«hotel[3].floor[L].room[0].name»	The name of the first room of the last floor of the hotel at index 3

1.5.3.5 Built-In Variables

Docmosis provides some built-in variables to assist with common data lookup requirements.

Variable	Description
«$top» or «$root»	The root of the data regardless of the current position or context in the template
«$this» or «$current»	The current source of data in the current position in the template. This allows for anonymous data lookups from arrays or collections such as «$current[0]».
«$parent»	The parent or container of data in the current context of the template. Allows data lookup in the current "hotel" when the current context is a "floor" for example.
«$idx»	The current index into any repeating data set. For example, if we are repeating over all hotels, $idx would report the index of the hotel we are up to.
«$itemnum»	Similar to $idx but is the number of the item which we are currently addressing. Item numbering starts at 1.
«$size»	The size of the current repeating data set. For example if we are repeating over all hotels, $size would be the number of hotels.

2 Developing Docmosis templates

The basic steps for developing a template are:

1. create the layout, boilerplate content and typesetting characteristics of a document;

2. incorporate the Docmosis elements (fields).

The boilerplate content can include sophisticated structures using headings, lists, tables, images, and headers and footers.

> **Tip**
>
> When creating a template, use the word processor of your choice from the two identified.

This chapter provides instructions for the inclusion of the supported fields; it is divided into sections that discuss the basic aspects through to some advanced techniques. In general, the information does not cover typesetting of documents but does provide information where necessary. Most of the information in this chapter is relevant to both word processors: where they differ, information is provided for each case.

Important

All the procedures in this chapter assume that you understand the techniques required for the particular word processor and that you have a document open in the word processor on which you can perform the procedure.

In addition, the procedures use menu-based instructions for consistency.

2.1 Incorporating Docmosis elements

You can add fields at any location in a document template. Each field must have an appropriate property name that identifies it and associates it with an element of the data generated by Docmosis. During document generation, Docmosis expects the application to provide values and logical data structures with the same names and structure as the elements that exist in the template.
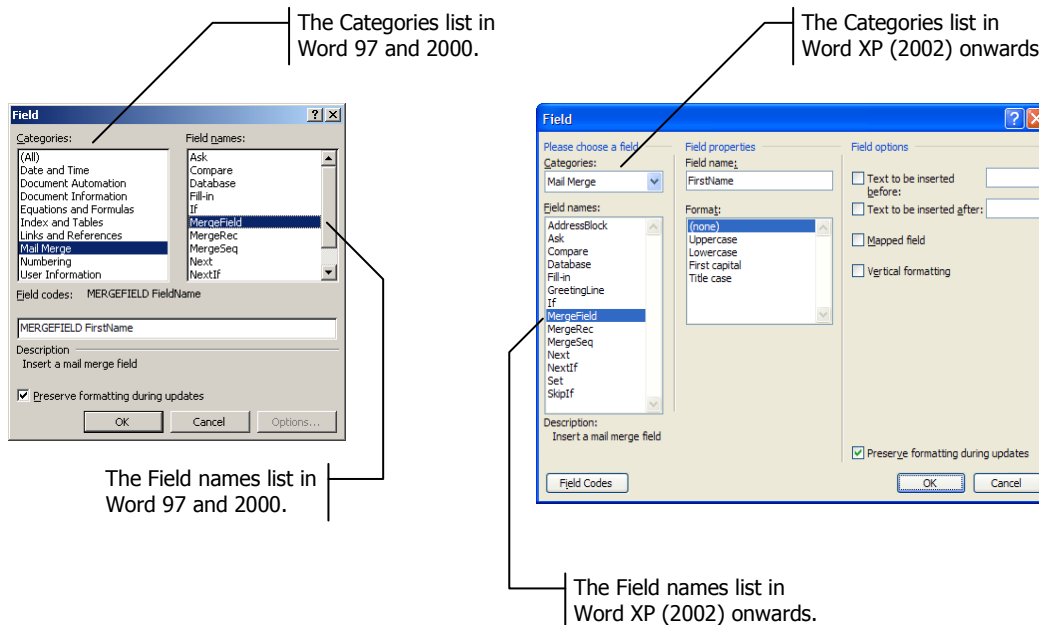
Note

The templates you develop are an integral part of the Docmosis functionality and include some features that are closely linked to the document generation application. During the development of your templates, you might have to consult with one of the application developers for assistance with certain aspects such as element names and program flow.

For more information about the Docmosis elements, refer to the *Docmosis Developer Reference*.

2.1.1 About MS Word versions

In general, the procedures to implement the Docmosis features into templates are consistent in all versions of MS Word. However, the layout of the *Field* dialog box has changed over the years. The following illustrations show the dialog box layouts for the different versions and identifies the areas that are relevant to Docmosis templates.

The Categories list in Word 97 and 2000.

The Categories list in Word XP (2002) onwards.

The Field names list in Word 97 and 2000.

The Field names list in Word XP (2002) onwards.

The Field dialog box layouts in different versions of MS Word

2.2 Basic text elements

Docmosis supports the inclusion of elements that simply match an element of data that is output by the application (essentially, this is a one-to-one match). Wherever an element occurs, Docmosis will substitute the actual data value in the document. The inserted data inherits all the typesetting characteristics that are applied to the field. The syntax for a text field is:

«element-name»

To populate the template element, the Docmosis engine would attempt to source data by the name of the element. In the following procedures, the example element name is `FirstName`.

Note

The procedure for inserting a field is the basis for inserting all the fields and logical structures that are used in Docmosis. The information contained in this section applies to all instances of an element and is not repeated unless it is necessary for clarity. The advanced sections later in this document build on the information in this section.

To insert a text field (MS Word):

3.   Position the insertion point at the location for the field.

4. Select **Insert > Field**.

5. In the *Field* dialog box, select Mail Merge from the **Categories** list.

6. Select MergeField from the Field names list.

7. Type `FirstName` into the appropriate field (see About MS Word versions earlier).

8. Click **OK**.

> **Tip**
> By default, MS Word displays the same text in the merge field as the title you enter in the *Field* dialog box. You may change the text that is displayed without changing the title of the merge field. Simply edit the text that appears between the angle brackets. Be warned though that this is generally not a good idea because if anyone updates the field codes in the document, the "display" name will be reverted back to the real contents.

> **Tip**
> In MS Word documents it is a good idea to frequently ensure the merge fields are displaying what they actually are going to look up. This can be achieved by selecting the fields (or the whole document) and pressing F9 (update field codes).

To insert a text field (OpenOffice.org Writer):

1. Position the insertion point at the location for the field.

2. Select **Insert > Fields > Other**.

3. In the *Fields* dialog box, select the **Functions** tab.

4. Select Input Field from the **Type** list.

5. Type `FirstName` into the **Reference** field.

6. Click **Insert**.

7. When the *Input Fields* dialog box prompts you, in the field below the one that contains the field reference, type the text that you want to be displayed in the document to identify the field. Keeping this text consistent with the text in the Reference is a good idea since it can avoid mix-ups.

> **Note**
> OpenOffice.org Writer presents the field information differently from MS Word: OpenOffice.org Writer does not insert a pair of angled brackets («…») around the displayed field text. You can hover your mouse cursor over the field to see the "real" value that will be used by Docmosis to lookup the data.

8. Click **OK**.

9. To close the *Input Fields* dialog box, click **Close**.

Note

When inserting fields in OpenOffice Writer you may choose to leave the field dialog open whilst you work and whenever you need to add a field, you simply go to the dialog and start adding it. Also, note that ctrl-F2 is a shortcut to the Fields dialog.

2.3 Images

Docmosis is able to insert images at arbitrary locations in documents. Instead of using fields to identify the location for an image substitution, Docmosis uses the word processor's image handling features. By handling images this way, the template can precisely define how the image will be placed and bordered within the resulting document. As each word processor works slightly differently, there are specific methods for setting up the Docmosis code element:

- In MS Word Docmosis uses the bookmarks feature to identify a name for an image; and

- OpenOffice.org Writer supports the identification of images directly, using a *Name* property.

Docmosis Example Template

Staff Profiles

Please contact any of our friendly and experienced staff for more information about your system solution.

«hcrs_IDSets»

Name: «name»
Position: «position»
Tel: «telephone»
Email: «email»

Summary:
«profileSummary»

«hces_IDSets»

In MS Word, bookmarks identify images.

Images can be placed anywhere in a Docmosis template.

Note

In the preceding example, a borderless table is used for layout purposes.

Tip

This activity doesn't discuss the actual images that you will publish, only the placeholder image. You may create and use your own image but for your convenience, a placeholder image is provided as part of the Docmosis distribution.

You are free to use it without restriction.

Use only inline images in MS Word

Docmosis cannot support floating images in MS Word because it uses the bookmarks feature to assign a name an image placeholder. When you position an image using the floating position settings, MS Word removes the bookmark. There are other limitations to how Docmosis supports images, particularly in terms of overlapping with text and other images. This generally will not cause issues for typical documents.

Image file size

When you insert a placeholder image, you will embed the image in the document. This means there is a copy of the image at every location in which it is placed (not simply a single, referenced copy). To limit the overall size of the template file and to improve the performance of a document generation, you should use relatively simple and small placeholder images to identify the locations.

Image placeholder naming convention

In this section, placeholder names are identified using a special prefix "bm_". This prefix is a useful way to distinguish those items that are specific to your Docmosis application and enables you to use the bookmarking and naming features for other items that aren't part of a document generation.

The prefix that you apply to bookmarks and images in the documents is not used during document generation. Using the example in this section, the image would be identified in the document as bm_image1 but the application would use the name image1. The prefix is stripped from the name by Docmosis before the data is populated.

In the following procedures, the example image name is image1.

To insert an image element (MS Word):

1. Position the insertion point at the location of the image.

2. Select **Insert > Picture > From File**.

3. In the *Insert Picture* dialog box, navigate to the location of the placeholder image and select it in the list of files.

4. Click **Insert**.

5. When the image appears in the document, select it and use the reshaping handles to adjust the dimensions of the image.

6. Make sure that the image is selected and click **Insert > Bookmark**.

7. In the *Bookmark* dialog box, type bm_image1 into the **Bookmark name** field.

8. Click **Add**.

Tip
MS Word wraps the content of the bookmark in light-coloured square brackets. To see the bookmark in place, set the **Bookmarks** option in the MS Word *Options* dialog box.

To insert an image element (OpenOffice.org Writer):

1. Position the insertion point at the location of the image.

2. Select **Insert > Picture > From File**.

3. In the *Insert picture* dialog box, navigate to the location of the placeholder image and select it in the list of files.

4. Click **Open**.

5. When the image appears in the document, select it and use the reshaping handles to adjust the dimensions of the image.

6. Make sure that the image is selected and click **Format > Picture**.

7. In the *Picture* dialog box, select the **Options** tab.

8. Type `bm_image1` into the **Name** field.

9. Click **OK**.

2.4 Conditional sections

Conditional content is content in that will be populated in the final document depending upon the data that is generated by the application. If the specified condition is met, the content within the matching conditional section is rendered in the document.

An example of the application of conditional content might be in a product description such as that for a motor vehicle in the following illustration.
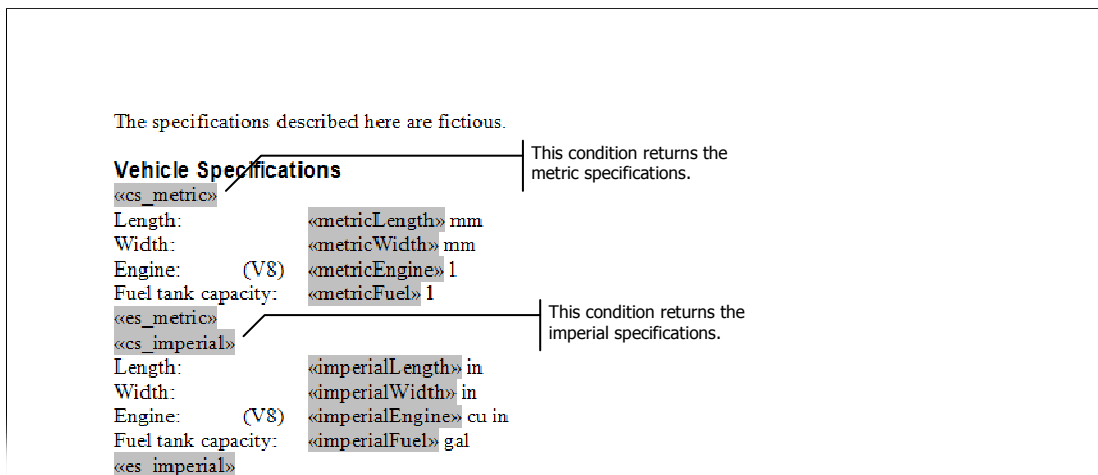
An example of conditional sections in a Docmosis template.

The conditional sections will render the data that is appropriate for each condition. That is, each document will be generated with either metric or imperial specifications but not both. Each conditional section is defined using a pair of fields: a start field and an end field. The general syntax for a conditional section is:

«cs_condition-name»

The text and elements of the conditional section.

«es_condition-name» or simply «es_ »

Conditional sections can use expressions, variables and range specifiers. See the tables in 1.5.3 Docmosis elements for more information.

The conditional 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.

To create a conditional section:

1. Position the insertion point in an empty paragraph at the starting location of the conditional section.

2. Insert the opening condition element into the empty paragraph.

3. Add the boilerplate content and other Docmosis elements into the subsequent paragraphs in the document.

4. Insert the closing condition element into an empty paragraph following the conditional content.

5. Repeat steps 1 through 4 for as many conditions as there are in your application.

2.5 Repeating sections

In a document, a repeating section is a group of elements in succession whose content changes but whose format is the same. Docmosis supports several forms of repeating sections: block-level, tables and lists.

Note

Tables and lists are special forms of repeating sections. They are discussed after this section that deals specifically with block-level repeating sections.

There might be occasions when you want to include repeating sections but do not want to use tables and lists (bulleted or numbered) to present them. In this case you use the Docmosis repeating section elements. Repeating sections can contain any content desired, and it will be repeated whilst there is data to be displayed.

The example below shows a repeating section named `IDSets`, and it contains a table with an image and textual data. This table will be repeated as many times as there is data associated with `IDSets`.
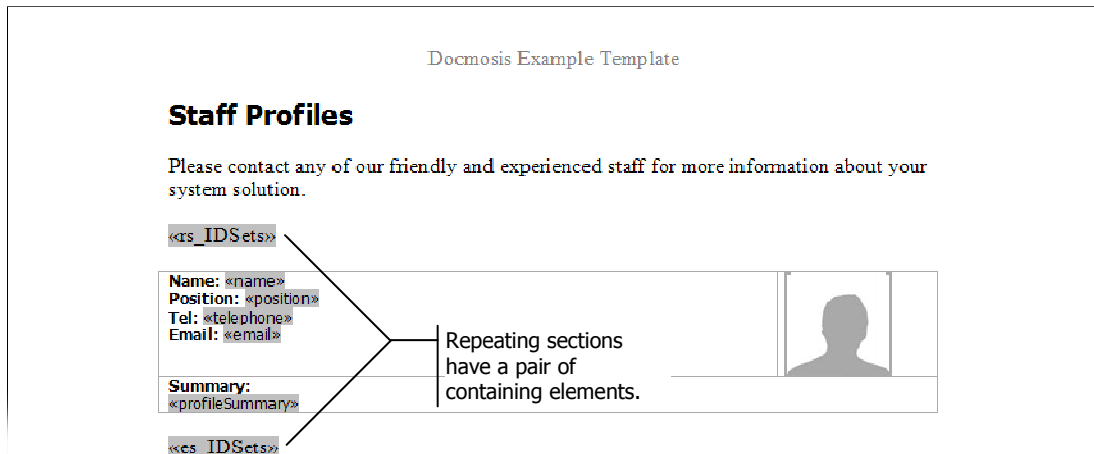
Repeating sections also contain boilerplate content and other Docmosis elements.

The general syntax for a repeating section is:

«rs_repeating-section-name»

The text and elements of the repeating section.

«es_repeating-section-name» or simply «es_ »

Repeating sections can be "nested" inside other repeating sections to any depth desired. Repeating sections can use variables and range specifiers as appropriate. See the tables in 1.5.3 Docmosis elements for more information.

The repeating 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.

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 Docmosis elements into the subsequent paragraphs in the document.

4. Insert the closing element into an empty paragraph following the repeated content.

2.6 Tables

Using fairly simple table markup, you can create sophisticated table layouts in your output documents. In addition to being able to insert text and images using the methods already described, you can use the table-specific Docmosis elements to control:

- including or excluding of groups of rows;

- repeating groups of rows;

- removing columns.

2.6.1 Conditional rows

A set of consecutive rows can be removed from a table using conditional row elements. The following example uses the «cr_hasFriends» and «er_hasFriends» elements to indicate a group of rows in a table that should be excluded if there are no friends.

In this case, if the data indicates that hasFriends is true then the row containing "Jimmy has some friends" would be left in the resulting document, otherwise it would be removed. In all cases, the rows containing the markers «cr_hasFriends» and «er_hasFriends» will be removed:

Note

End markers for conditional rows can also be defined without the name. In the above example, the field «er_hasFriends» could also be simplified to «er_ ».

2.6.2 Repeating rows

Rows of a table can be repeated whilst there is data to repeat. The following example will list of all the friends of Jimmy using one row for each friend showing their name in one column and job in another.

The following example uses the «rr_friends» and «er_friends» elements to indicate a group of rows in a table that should be excluded if there are no friends.

Circle of Friends	
Friend	**Job**
«rr_friends»	
Jimmy has a friend called «friend»	«friendJob»
«er_friends»	

In this case, while the data can supply information for friends the row containing the lookup friend information will be rendered. In all cases, the rows containing the markers «rr_friends» and «er_friends» will be removed.

Circle of Friends	
Friend	**Job**
Jimmy has a friend called Dave	Marketing Manager
Jimmy has a friend called Dee	C++ Developer
Jimmy has a friend called Pete	Roof Tiler

The template for repeating rows also provides some tricks for colouring and borders that can produce impressive results. The rules are as follows:

1. if a cell of a row inside a set of repeating rows has a background colour different to that of the corresponding cell of the starting row (the row with the «rr_xxx» element), then the background colour for that cell will alternate between that of the starter row and it's own background colour. This allows everything from plain tables, to alternating rows to crazy alternating patterns.

2. the starting row (the row with the «rr_xxx» element) determines the top border of the first repeating row. The ending row (the row with the «er_xxx» element) determines the bottom border of the last row to be rendered. This applies on a cell-by-cell basis as for the background colouring. This allows for highly configurable borders to be specified that work pretty much as one would expect.

The following example creates a bounding border encapsulating all the repeating rows (including the marker rows, and alternates the background colour.

Notice in the result below the alternating background colours and the border wraps all cells collectively.

Note

End markers for repeating rows can also be defined without the name. In the above example, the field «er_friends» could also be simplified to «er_ ».

More advanced examples are given in section 2.6.4.

2.6.3 Conditional columns

A template may also indicate columns in a table that are to be conditionally removed. The width of the table remains as fixed in the template and the space recovered by the removal of the column is spread across the remaining columns. The following example shows a Docmosis conditional column element («cc_showJobs») at the top of the second column.

When rendered, this removes the column entirely where the data indicates that showJobs is false.

The following example uses *expressions* to conditionally remove two columns from the table. If you examine the Trial 2 and Trial 3 columns, you will see the conditional column expressions in fields «cc_{ntrials>1}» and «cc_{ntrials>2}».

Lab Results	Trial 1	Trial 2 «cc_{ntrials>1}»	Trial 3 «cc_{ntrials>2}»	All Passed
«rs_specimens»				
«name»	«result1»	«result2»	«result3»	«passed»
«re_»				

If the underlying data says there is only one trial for example, that is, ntrials = 1 then the conditions for the Trial 2 and Trial 3 columns will evaluate to false and the columns will be excluded.   This is shown in the following example output:

Lab Results	Trial 1	All Passed
SP1A-2474	47.241	No
SP1A-2524	23.442	Yes
SP1A-3211	13.672	No
SP1A-3213	53.342	No

If the underlying data says there are 2 trials, that is, ntrials = 2, then the Trial 2 column will remain in the resulting document, but the Trial 3 column is still removed:

Lab Results	Trial 1	Trial 2	All Passed
SP1A-2474	47.241	32.553	No
SP1A-2524	23.442	44.123	Yes
SP1A-3211	13.672	61.153	No
SP1A-3213	53.342	12.223	No

2.6.4 Advanced table structures

Docmosis supports the nesting of repeating and conditional content in table structures. The following example template shows multiple levels of repeating to print out the room details within each floor within each hotel.

The example above is fairly extreme and it would often be more natural to represent the structure in a combination of repeating sections and tables with repeating rows. The following template is equivalent but is not providing the entire structure within a single table:

2.7 Lists

Docmosis infers repetition when there are one or more elements in paragraphs formatted as a list using the 'bullets and numbering' features. As long as the data provider has data to populate, the list will be rendered with items.

In the following example, a field is formatted as a numbered list and will be automatically expanded.

An example of a list item.

Docmosis splits the element into two parts, a repeating component and a lookup component. The repeating component can be limited by a using range specifier covering multiple values. In this example, the range specifier [*] against the friends name means for all friends and the trailing friend name is the lookup of the data to display.

Docmosis Example Template

List Expansion Example

These are my friends:

1. Dave
2. Dee
3. Pete

As another example of how the field is split, consider the following template.

Docmosis Example Template

List Expansion Example

My first friend «friends[F].friend» has the following pets:

* «friends[F].pets[*].type»

The element now is in a bullet list style rather than numbered. It has a repeating component friends[F].pets[*] meaning all pets of the first friend and a lookup component type. The resulting document is shown below where the friend has a dog and a parrot.

Docmosis Example Template

List Expansion Example

My first friend Dave has the following pets:

* dog
* parrot

Docmosis only allows a single component of the element to be a multi-valued range. For example, Docmosis would not allow an element friends[F2].pets[*] since this would repeat at multiple stages and typically would be a mistake.

To create a list:

1. Position the insertion point at the location of the first list item.
2. Format the paragraph as a list item (bulleted or numbered).
3. Add the Docmosis element that will render the data into the list paragraph.

3 Applying a Renderer

To enhance the presentation of your documents, you can apply a "renderer" to the field. This allows various changes to be made to the field including:

- changing the background colour (if in a table cell);
- changing the font style to bold, italics or underline.
- changing the text to be displayed

To associate a renderer with a field, you use the `renderer` qualifier with the name of the renderer that is to be used. The Java code supplying the data will also be responsible for supplying the renderer (which is described in the *Docmosis Developer's Reference*). The only requirement in the template is to associate the renderer with the applicable field. For example:

{renderer=myrenderer}

The field with the renderer would look something like this:

«surname{renderer=myrenderer}»

Note

Remember, you don't need to use a `renderer` qualifier if you want a cell to be permanently shaded. Renderers allow changes based on conditions that exist during document generation.

Tip

You can set the background colour of a cell with no data by using an element whose name has no equivalent (and thus no data) in the application, or by using a field named `dummy`.

Important

To enable the qualifier to work, you must make sure that you edit the code of the field and not the display name/result.

Important

Make sure there are no spaces in the field name or qualifier. Spaces may cause the qualifier to not be recognised, particularly if using Word for the template source. If a space is required, use the underscore character (_).

3.1.1 Renderer Parameters

Renderers may also be passed parameters so that the same renderer can be given different directives depending on the case at hand. For example, the following fields pass different parameters to the renderer named `myrenderer`:

«surname{renderer=myrenderer('strict')}»

«surname{renderer=myrenderer('bold','relaxed')}»

The first sends a "strict" parameter to myrenderer, whilst the second sends "bold" and "relaxed". myrenderer is written in Java code and may respond to these parameters as it sees fit.

3.1.2 Built In Date Renderer

Docmosis has a built in date renderer which is available to all templates without requiring any Java code. To use it simply reference a renderer named "date". It will format the date in the default format (dd MMM yyyy) or as indicated by a single parameter. The parameter is in the format specified by the Java `SimpleDateFormat` class which is described in the Java API Specification. Some examples of suitable formats are shown below. The full documentation for `SimpleDateFormat` can be read on-line at http://java.sun.com/reference/api/ and navigating into the J2SE version of your choice.

The date formatter only applies to date-typed data as comes from Java objects or database queries.

The following example uses the built in date renderer to format a birthdate field into a two-letter year format:

«birthdate{renderer=date('yy')}»

To allow for formats that include a space, the underscore character (_) is transcribed automatically into a space. This is done since fields cannot contain spaces when using Word as the template source. To include an underscore, the backslash character can be used to indicate that the underscore should be left as an underscore (_).

The table below shows some examples of date formatting:

Example Field	Result
«birthdate{renderer=date}»	27 May 2009
«birthdate{renderer=date('E_MMM_yyyy')}»	Wed May 2009
«birthdate{renderer=date('yyyy')}»	2009
«birthdate{renderer=date('dd/MM/yyyy')}»	27/05/2009

To attach a renderer to a field (MS Word):

1. Select the element in the required table cell.

2. Right-click and select **Edit Field** from the context menu (or toggle the field code in earlier versions of Word using *Shift-F9* or *Alt-F9*)

3. Add the `renderer` qualifier inside curly braces to the element name.

4. If you are using the **Edit Field** dialog Click **OK** (or toggle the field code back using *Shift-F9* or *Alt-F9*).

5. If you don't want to see the qualifier in the field result, delete it directly in the field.

6. If you have already created repeating rows or row groups in a table, repeat steps 1 through 5 for each row or group.

To attach a renderer to a field (OpenOffice.org Writer):

1. Select the element in the required table cell.

Note

Don't left-click on the field with the mouse. You won't be able to change the field code in the *Input Field* dialog box.

2. Select **Edit > Fields**. The *Fields* dialog box is displayed.

Tip

You can also right-click the field and select **Fields** from the context menu.

3. In the References text box, add the `renderer` qualifier inside curly braces (as shown earlier).

4. Click **OK**.

5. If you have already created repeating rows or row groups in a table, repeat steps 1 through 4 for each row or group.

3.1.3 Built In Boolean Renderer

Docmosis can format boolean (true/false) data into several presentational styles using the built-in "boolean" renderer. This renderer exists since true/false can often be better displayed in a document by Yes/No, Y/N, tick/cross etc.

The built-in Boolean render takes a single parameter indicating the way the true and false values should be displayed. The following table lists the built in values that may be passed as a parameter.

No parameter	true is rendered as "true" and false as "false"
"yn"	true is rendered as "Y" and false as "N"
"ynlc"	true is rendered as "y" and false as "n" (lower case)
"yesnouc"	true is rendered as "YES" and false as "NO" (upper case)
"yesnolc"	true is rendered as "yes" and false as "no"
"yesno"	true is rendered as "Yes" and false as "No" (mixed case)
"windings1"	true is rendered as a windings tick and false as a cross
"windings2"	true is rendered as a windings checkbox ticked and false as unticked
"dingbats1"	true is rendered as a dingbats light tick and false as a light cross
"dingbats2"	true is rendered as a dingbats heavy tick and false as a heavy cross

For example, to use the `yn` Boolean renderer, a field may attach a renderer named "`boolean`" and give it the '`yn`' parameter:

«isRetired{renderer=boolean('yn')}»

Which will render true as Y and false as N.

3.1.3.1 Using the Wingdings Boolean Renderer

The wingdings renderers are unlike the other forms of renderers in that they rely on the template field actually being in the wingdings font in the first place. This means you would create the fields as required:

Docmosis Template Example

Identity Provided

Passport	«hasPP{renderer=boolean('wingdings1')}»
Driver's Licence	«hasDL{renderer=boolean('wingdings1')}»
Residential Bill	«hasRB{renderer=boolean('wingdings1')}»

Then change the font of the fields that are using the wingdings renderer to the wingdings font:

Docmosis Template Example

Identity Provided

Passport	★≋☺♦☞☜☺◻︎♏■♎♏◻︎♏◻︎♔♌♌◻︎◻●♏☺■☎︎♙♦❩■♈♌♎❩■♈♌♦☜♌◐"☽
Driver's Licence	★≋☺♦♚☹☺◻︎♏■♎♏◻︎♏◻︎♔♌◻︎◻●♏☺■☎︎♙♦❩■♈♌♎❩■♈♌♦☜♌◐"☽
Residential Bill	★≋☺♦♻︎♒☺◻︎♏■♎♏◻︎♏◻︎♔♌◻︎◻●♏☺■☎︎♙♦❩■♈♌♎❩■♈♌♦☜♌◐"☽

When this document is then rendered, the renderer will place the correct wingdings characters into the document.

Docmosis Template Example

Identity Provided

Passport	✗
Driver's Licence	✗
Residential Bill	✗

Important

Using specific fonts may cause issues if you then migrate your system to another platform. You should always confirm the operation of your template in all its intended system environments.

To insert a formatted boolean value element:

1. Insert the element using the basic element procedure (see page 13).

2. When you are happy that the element is correct, select it and format its characters using the font you require for the resulting symbol.

3. Work with the application developer to ensure that the Docmosis data formatter returns the appropriate character values for the true and false conditions.

3.1.3.2 Using the Dingbats Boolean Renderer

The Boolean renderer also provides the ability to render dingbats character replacements. Using dingbats does not require any changes to fonts in the templates as is required for the wingdings formatter.

You may find that Word does not understand the dingbats characters, so if you choose Word as the output document type, you typically would not use dingbats character renderers. The dingbat Boolean renderer can be used with Word or Writer templates, the effect is only an issue in the output document.