Docmosis v2.0 Release Notes

New Features

#	Change
1	**Table row repetition is now defined differently** The old "by-example" mechanism of copying rows is **no longer supported**. Old way: ***ID*** | ***Value*** «id» | «value» «id» | «value» New way: ***ID*** | ***Value*** «rr_idRow» | «id» | «value» «er_idRow» | Notes: 1. The *{section=qualifier}* should be dropped from all merge fields in tables as this information is now provided by the "rr_" tag. 2. The same nesting rules apply as for the previous by-example style but the new mechanism is easier to work out complex structures 3. Alternating row colouring can now be specified by the template instead of calling to a renderer (though renderers can still override the colour from the template). The background colour of the *rr_* row is used as the even row colour and the background colour of each template row within the *rr_*section provides the odd row colour. Thus the above example would render like this: ***ID*** | ***Value*** 1 | Alpha 2 | Beta 3 | Gamma 4 | Omega More complex colour alternation is possible: ***ID*** | ***Value*** «rr_idRow» | «id» | «value» «id» | «value» «er_idRow» |

Would render as:

ID	*Value*
1	Alpha
1	Alpha
2	Beta
2	Beta
3	Gamma
3	Gamma
4	Omega
4	Omega

4. Borders can now be styled from the template. The *rr_* row provides the top border for the first row and the *er_* row provides the bottom border for the last row. This allows bounding boxes etc to be generated by simply bounding the rows as desired:

ID	*Value*
«rr_idRow»	
«id»	«value»
«er_idRow»	

Would render as:

ID	*Value*
1	Alpha
2	Beta
3	Gamma
4	Omega

5. Colouring and border styling are prototyped from the *rr_* and *er_* rows as mentioned. This applies on a cell-by-cell basis if your content rows have the same number of columns as the *rr_* and *er_* rows. Otherwise a best-effort is done.

2 **Template Errors to Output Document**

Wherever possible, errors during analysis and population of templates are sent to the resulting document and highlighted, rather than throwing an exception. The following example shows an error trying to apply a renderer to the <<value>> field in a table. The renderer failed with an ArrayIndexOutOfBoundsException in custom code.

This table is all about Greek

ID	*Value*
bb1	Alpha
bb2	<<value>>[Docmosis1]
bb«id»	«value»

Docmosis1 **Renderer (name="alt1", class=class com.docmosis.example.ExampleRenders$3) failed with the exception below when trying to render field "value". Cause:java.lang.ArrayIndexOutOfBoundsException:com.docmosis.example.ExampleRenders$3.r ender(line:320) Suggestion 1:Look into the Renderer implementation for the cause**

This behaviour can be disabled by setting the property:

docmosis.populator.error.fatal=true

NOTE: this new style of error handling is progressive and currently doesn't cover all errors

3 **Conditional Table Columns**

Table columns can be flagged as conditional which will then be removed from the document if the condition evaluates to false.
This is controlled by a merge field with the prefix *cc_*, such as "cc_abc".
Expressions and variables are now supported so this form of conditioning expression are also supported "cc_{$columns<2}". [Variables and expressions are introduced in notes to follow]. For example:

ID	*Value «cc_hasPrimeValues»*
«rr_idRow»	
«id»	«value»
«er_idRow»	

The second column will be removed if the data source returns a false for "hasPrimeValues". Note that if your table has rows with differing numbers of columns then this will always remove the column at the same relative location from the left as the marked column.

This goes a bit further in that the condition will also be applied to sub (or covered) columns. Consider this table:

ID	*Value «cc_hasPrimeValues»*		
«rr_idRow»			
«id» «value1»	«value2»	«value3»	
«er_idRow»			

The three columns under the Value heading are "covered" by the Value column. As

such, these columns will also be removed if the value column is removed.

4 **Variables**

Simple variables can be defined in the template and be referenced later. Defining a variable requires a merge field of the form *$var=term*. For example:

«$a=hotels.floors[1]»

defines a variable named "a" which holds reference to the value for the second floor of a hotel. This variable may then be referenced later, for example this will get the name of the floor:

«$a.floorName»

and this will repeat over all the rooms in the floor:

«rs_$a.rooms»
  «roomNumber»
«es_$a.rooms»

NOTE: support for literal values and expressions when setting variables is pending.

5 **Built In Variables**

There are some built-in variables to help with referencing data within or without the "current" context of data.

Variable	Description
$current $this	Current data context. Allowed unnamed types of data to be referenced such as arrays or collections of Java primitives. For example, $current[0] might reference the first string in an array of strings
$top $root	The root of the data – regardless of where in the population phase the render process is, the data can be absolutely referenced.
$parent	The data one level up from the current population context. For example, in a table row that is populating rooms of a floor in a repeating fashion, the $parent variable would refer to the parent of the rooms container, which is likely to be the current floor.
$idx	Current population index, that is, index into the current data set.
$itemnum	Current item number which is the position within the current data set where the starting position is 1.

	$size	The size of the current data set.

6 **Expression Support**

Expressions can now be used for conditional sections and conditional columns. The format of an expression is:

«cs_{expr}» or «cc_{expr}»

where supported expressions are:

Expression	Description
{a<b}	Lookup data "b" less than lookup of data "a"
{$a<b}	Lookup variable "a" less than lookup of data "b"
{a='123'}	Lookup data "a" equals String literal "123"
{b!=123}	Lookup data "b" is not equal to Number literal 123
{!c}	Lookup data "c" and logically negate it
{!$a}	Lookup variable "a" and logically negate it

Examples are:
«cc_{$columns<10}»
«cs_{roomNumber<101}»

7 **Enhanced Range Specification**

Ranges can now be specified so that parts of arrays or collections of data may be referenced. For example:

Field	Description
«rooms[0]»	First room
«rooms[F]»	First room
«rooms[L]»	Last room
«rooms[L3]»	First 3 rooms
«rooms[F2,L2]»	First 2 rooms and last 2 rooms
«rooms[1-3]»	2nd to 4th room
«rooms[1,3,L2]»	2nd, 4th and last 2 rooms
«rooms[1-L2]»	2nd to second last rooms
«rooms[*]»	All rooms

8 **Automatic Template Registration**

Paths can now be configured to be automatically monitored for templates and template changes. The properties:

docmosis.template.monitor.sourcepath=path1;path2...
docmosis.template.monitor.period=5

	if set will automatically watch for templates and register them according to their file name and relative location. NOTE: templates in jar files are also supported.

9	**Simplified Data Provision**
	
	All collecting of data to provide to Docmosis should now be performed via the DataProviderBuilder class. This class allows all forms of data to be attached, without needing to know anything about the underlying implementations.
	
	So, to provide data from some a combination of Java objects, a few strings of data and an SQL query, the code might look like this:
	
	DataProviderBuilder dpb = new DataProviderBuilder();
	dpb.addJavaObject(myFriend);
	dpb.addJavaObject(myUncle);
	dpb.add("reportTitle", "Associate Report");
	dpb.add("footerLabel", "Associate Report as at " + myDateStr);
	dpb.addSQL(myResultSet);
	…
	DataProvider dataProvider = dpb.getDataProvider();

10	**Simplified Template Identification** All types of TemplateIdentifier are now represented by the TemplateIdentifier class. All other classes have been remove. The same applies to TemplateContext which now assumes all contexts can be defined by a "/" separated "path". Templates can now be identified with or without a Context and if no Context is provided, the template is assumed to be at the root of the template store. So, to identify a template, we now do: TemplateIdentifier tid = new TemplateIdentifier("myTemplate"); Or TemplateIdentifier tid = new TemplateIdentifier("myTemplate", "project1/release1");
11	**Simplified Rendering** DocumentProcessor has a new simplified method for rending a template that takes a source template file, a destination file and the data source and performs the other tasks automatically. This means the following code is sufficient to render any template (without any data in this case):

	File template = new File("myTemplate.doc"); File result = new File("output/myDocument.pdf"); DataProvider data = new DataProviderBuilder().getDataProvider(); DocumentProcessor.renderDoc(template, result, data);
12	**Improved Core Performance** The core engine has had several improvements to reduce processing effort during document production including: • A new cache that holds ready-to-render templates in memory (default configuration is 5Mb) • Complex data lookup structures are pre-computed in the template registration phase.

API Changes

The following API changes should be noted

Class / Interface	Change
ConversionInstruction	Interface changed into Class
SimpleConversionInstruction	Replaced with ConversionInstruction
FileTemplateContext	Replaced with TemplateContext
FileTemplateIdentifier	Replaced with TemplateIdentifier
TemplateContext	Interface changed to class
TemplateIdentifier	Interface changed to class
FieldDetails	Package change
RenderedField	Package change. Changed from Interface to class
SimpleRenderedField	Replaced by RenderedField
FieldRenderer	Package change

Bug Fixes / Technical Enhancements

#	Change
1	Fixed a small memory leak that can occur when generating over 50000 documents without restarting a Docmosis converter.
2	Fixed an unhelpful error that occurs when renderers are not used correctly.
3	Fixed an unhelpful error that occurs when reflective data providers are not used correctly.
4	SQL Data Providers now handle all the different SQL data types.
5	Booleans and Strings can be treated interchangeably. Affects only String and File based data provision (not Reflective or SQL data).
6	Increased reliability of start-end tag matching despite styling in the document.