



## ReporterPE – Quickstart Guide

(Version 2.0 from Oct 20, 2024)

Thank you for your interest in ReporterPE.

This PostExecuter allows you to create PDF, XHTML and ODT documents from Lobster \_data without relying on the use of Jasper Reports. Instead, you use an ODT document (Open Office Format) as a template for creation.

### System requirements:

ReporterPE requires Lobster \_data 4.6.x or later. This version (2.0) of ReporterPE has been tested in Lobster \_data 4.6.9 to 4.6.11. Please note that we cannot guarantee the smooth operation of the PostExecuter in other versions of \_data or with other extensions that may be installed in /extlib.

### Parallel operation with Jasper Reports

This particularly applies to parallel operation with older Jasper Reports installations. Older installations use iText 1.2.7.jar. This library is not compatible with ReporterPE. ReporterPE depends on OpenPDF (a fork of iText), which is now also used when using JasperReports. If there is an iText\_1.2.7.jar in your /extlib directory, we recommend that you update the Jasper Libraries.

### Installation:

Unzip the downloaded archive ReporterPE.zip and place the .jar file contained in the /extlib folder in the /extlib folder of your Lobster \_data installation. Remove all older versions of ReporterPE. In the /etc/admin/datawizard folder of your Lobster \_data installation, edit the **custom\_post\_executer.properties** file. If it doesn't exist yet, create it. Add the entry **de.derbrill.reporter.ReporterPE**.

A restart of your \_data system is then required. On a Lobster \_data TEST system, no further action is required to work with ReporterPE. For a productive system you need a license key.



To achieve the best results when generating PDF files, it is recommended that you also install a current version of LibreOffice on your server. This can then be used by ReporterPE to initiate PDF generation and use the full range of LibreOffice functions. Unicode text (e.g. Arabic) and text frames are only supported by ReporterPE if LibreOffice is used to generate PDFs instead of the internal PDF engine. After installing LibreOffice, it is recommended that you restart your `_data` system. If you have already used ReporterPE and then install LibreOffice without restarting `_data`, you may experience malfunctions when you first use LibreOffice as a rendering engine.

#### General functionality

ReporterPE populates an ODT document you created based on a target tree structure you created in the mapping.

In order to use ReporterPE as a PostExecuter, your mapping must first generate a JSON file. Accordingly, the JSON Integration Unit must **always** be used in phase 5. The structure of the target tree in phase 3 depends on the template you created. The field names must correspond to the variables specified in the template.

#### Provide license for the production system

If you have already received a license key for your Lobster production system from us, store it in the `./conf/ReporterPE/` directory of your Lobster `_data` Prod instance. The license is tied to your Lobster `_data` installation ID. If you operate multiple Lobster `_data` productive instances, you need one license per installation ID used.

If you would like to purchase a license key, contact us by email at: [helpdesk@derbrill.de](mailto:helpdesk@derbrill.de)



## First steps – the demo profile

ReporterPE is delivered with several demo profiles. If you import the package files into your Lobster\_data system, you can test the range of functions immediately. The required configuration files are created automatically by importing the profile. The ReporterPE folder with the Examples subfolder is explicitly created in the conf directory. The required test data is also included in the package.

Profiles are provided, some of which require the installation of LibreOffice on the server. These can be found in a separate folder.

### Mapping settings

In order to fill variables in the template, the target structure must meet certain requirements:

Simple variables are created at the root level of the target tree (more precisely in the node that you specify as “Start at Node” in the IntegrationUnit).

A field named myField refers to the variable \$myField in the template.

Fields in subnodes with the maximum attribute set > 1 are considered list entries. Consider a node named myList. This contains the fields myField1 and myField2.

These fields are referenced in the template by specifying the variables \$myList.myField1 or \$myList.myField2.

### Mandatory fields

Two fields are mandatory in your mapping.

**templatefile** must contain the path to your template ODT file. It is possible to use files from the (accessible) file system or a resource accessible in the network. The nomination is made by specifying the desired protocol (File:/ http:// or https:// ) followed by the URL. E.g. File:./conf/ReporterPe/examples/demo.odt or https://example.org/reporter/demo.odt

**outputformat** must contain the desired output format. The values pdf, xhtml and odt are possible.



## Magic extensions

If your field name (and the variable name in the template) ends with two underscores, followed by styled (e.g. myField\_\_styled), the text content is interpreted as markup styled text.

The following formatting markups are supported:

`<b>Fat</b>`

`<i>italic</i>`

`<u>underline</u>`

`<sub>subscript</sub>`

`<sup>superscript</sup>`

Colors can be set using `<p>` and `<span>` tags:

My mother has `<span style="color:rgb(0, 0, 230);">blue</span>` eyes.

Please note that when using styled text, simple line breaks must be replaced by the `<br/>` tag. It is essential to close the tag well-formed (in the sense of XML). Otherwise display errors may occur.



## References to images:

To set the properties of an image in the template, create a node with the Maximum attribute = 1. The name of the node must end with two underscores followed by img. For example myImage\_\_img. The node must contain at least two fields:

**url:** with a reference to a URL on the web, or a path in the (accessible) file system. If a file is referenced in the file system, the content of the field must begin with the expression **File:.**

Example: **File:./conf/images/myImage.png**

**resize:** true or false

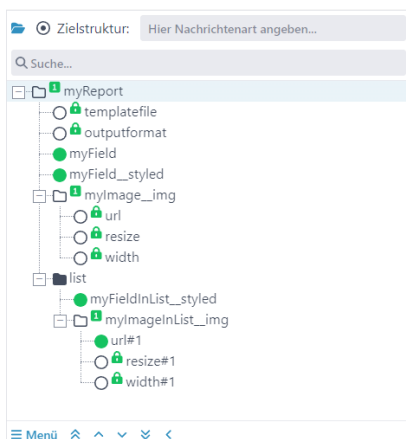
Instead of a url, a data field can be specified. Using this field it is possible to pass base64 encoded data to ReporterPe. The data must be passed in a way that is compatible with the result of a create barcode(a, b, c, d, e) function call. For example:

<https://www.lobster-world.com/online/ data/docs/46/en/BasicCreateBarcode.html>

If the field **resize** contains the value **true**, at least one of the fields width or height is mandatory. If only one of the two fields is specified, the image will be scaled proportionally. If both width and height are specified, the image will be scaled based on the set values and, if necessary, displayed distorted.

It is possible to include images in lists. It is important to ensure that the node that determines the properties of the image is subordinate to the node of the list.

The target tree for filling a template can look like this:

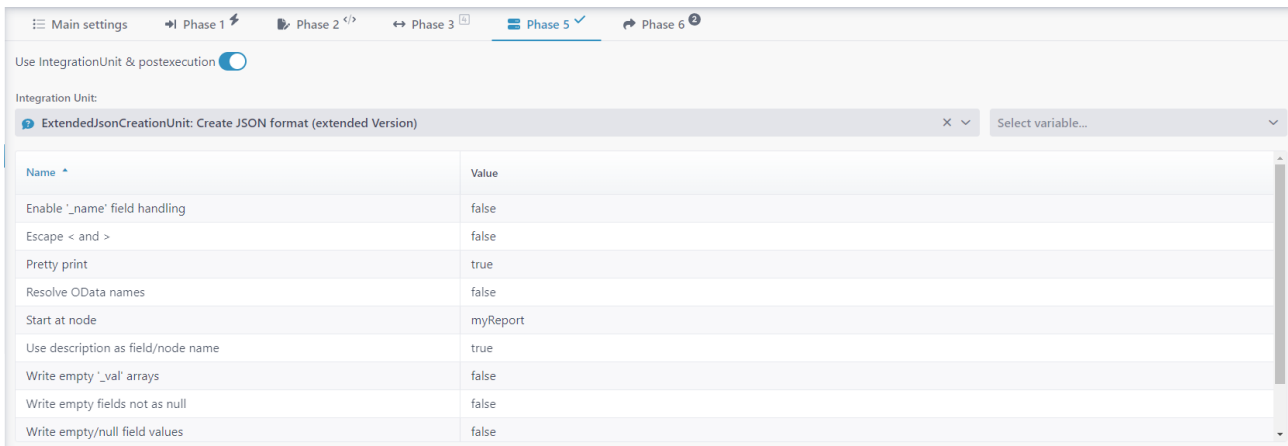


Once your mapping is complete, there are only two more steps left.



## Phase 5 – IntegrationUnit

Select the **ExtendedJsonCreationUnit** here. Set the **Start at node** value corresponding to your mapping.



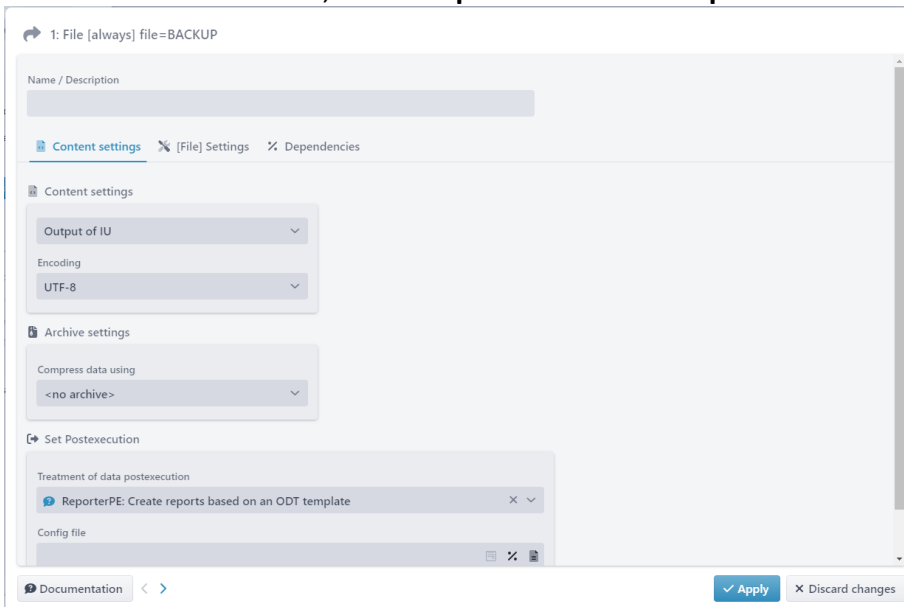
The screenshot shows the 'Phase 5' configuration window. At the top, there are tabs for 'Main settings', 'Phase 1', 'Phase 2', 'Phase 3', 'Phase 5' (selected), and 'Phase 6'. Below the tabs, there is a toggle for 'Use IntegrationUnit & postexecution' which is turned on. Under 'Integration Unit:', a dropdown menu shows 'ExtendedJsonCreationUnit: Create JSON format (extended Version)'. Below this is a table with settings:

Name	Value
Enable '_name' field handling	false
Escape < and >	false
Pretty print	true
Resolve OData names	false
Start at node	myReport
Use description as field/node name	true
Write empty '_val' arrays	false
Write empty fields not as null	false
Write empty/null field values	false

## Phase 6 – Response Units

Create an response unit of your choice. For content settings, select **Output of IU**. Encoding: **UTF-8**.

Under Set Postexecution, select **ReporterPE: Create reports based on an ODT template**.



The screenshot shows the '1: File [always] file=BACKUP' configuration window. It has tabs for 'Name / Description', 'Content settings' (selected), '[File] Settings', and 'Dependencies'. Under 'Content settings', there are dropdowns for 'Output of IU' and 'Encoding' (set to 'UTF-8'). Under 'Archive settings', there is a dropdown for 'Compress data using' set to '<no archive>'. Under 'Set Postexecution', there is a dropdown for 'Treatment of data postexecution' set to 'ReporterPE: Create reports based on an ODT template'. At the bottom, there is a 'Config file' field and buttons for 'Documentation', '<', '>', 'Apply', and 'Discard changes'.



## Creating the ODT template

For this quickstart guide, we assume that you are using LibreOffice (<https://en.libreoffice.org/>) to create your templates. We used LibreOffice version: 7.6.7.2 to create the templates. Certain functions may be found elsewhere in different versions.

### First steps – transfer values from the profile to the template

The transfer to the template takes place via placeholders (so-called template variables). These are announced to the template by using the \$ symbol. The symbol is followed by the field name in the profile, e.g. \$myField. The template variable can be inserted anywhere in the template. Multiple use of the variable is possible.

To give specific formatting to the dynamically inserted text, format the entire template variable expression.

For example like this: **\$myField**

Der aus dem Mapping übergebene Literal wird dann fett und in blau angedruckt.

The literal passed from the mapping is then printed in bold and blue.

Please note that ReporterPE only supports the fonts in the PDF that are installed on the \_data server and are available to the user who started the Lobster \_data service! If a selected font is not found, the system reverts to the standard fonts. Please refer:

[https://en.wikipedia.org/wiki/Portable\\_Document\\_Format](https://en.wikipedia.org/wiki/Portable_Document_Format)

It sounds like nothing could go wrong, right? - Unfortunately no.

For the ODT document, the reference to the variable is a text that can also be formatted. Does the reference to the variable contain various formatting instructions, for example something nice and colorful like **\$myField**, the variable is not recognized as such, but is printed as \$myfield. If there are multiple formatting instructions, the PostExecuter no longer recognizes the specification of the variable.



Unfortunately, these formatting attempts cannot always be recognized in ODT. And most of the time this happens, often invisibly to you, when you make small changes to the template variables.

The first way to solve this is to delete the direct formatting in the template for the variables (highlight, then Ctrl + M, or via the context menu accessible by right-clicking). Then apply the desired formatting to the **entire** variable expression.



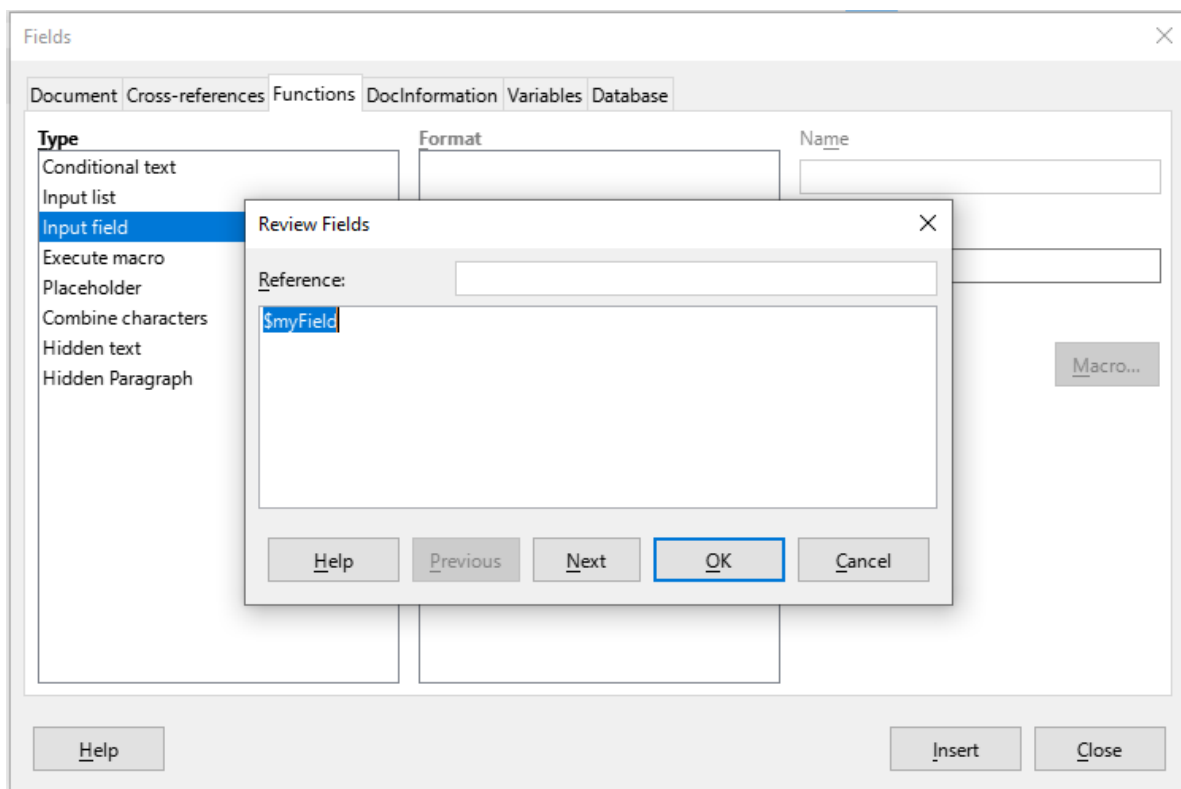




## Using field commands in LibreOffice

You can be on the safe side if you put in a little more work. 😊 Instead of typing the variable name directly into the template, create an input field via

**Insert -> Field -> More fields** or the keyboard shortcut **CTRL+F2**





## Declare template variables as optional

**Important:** All template variables specified in your template are **mandatory** fields by default. If a specified template variable is not filled by the JSON structure generated in the mapping, the variable name is printed.

However, you have the option to mark a template variable as optional by writing an exclamation mark after the \$ symbol.

\$!myField

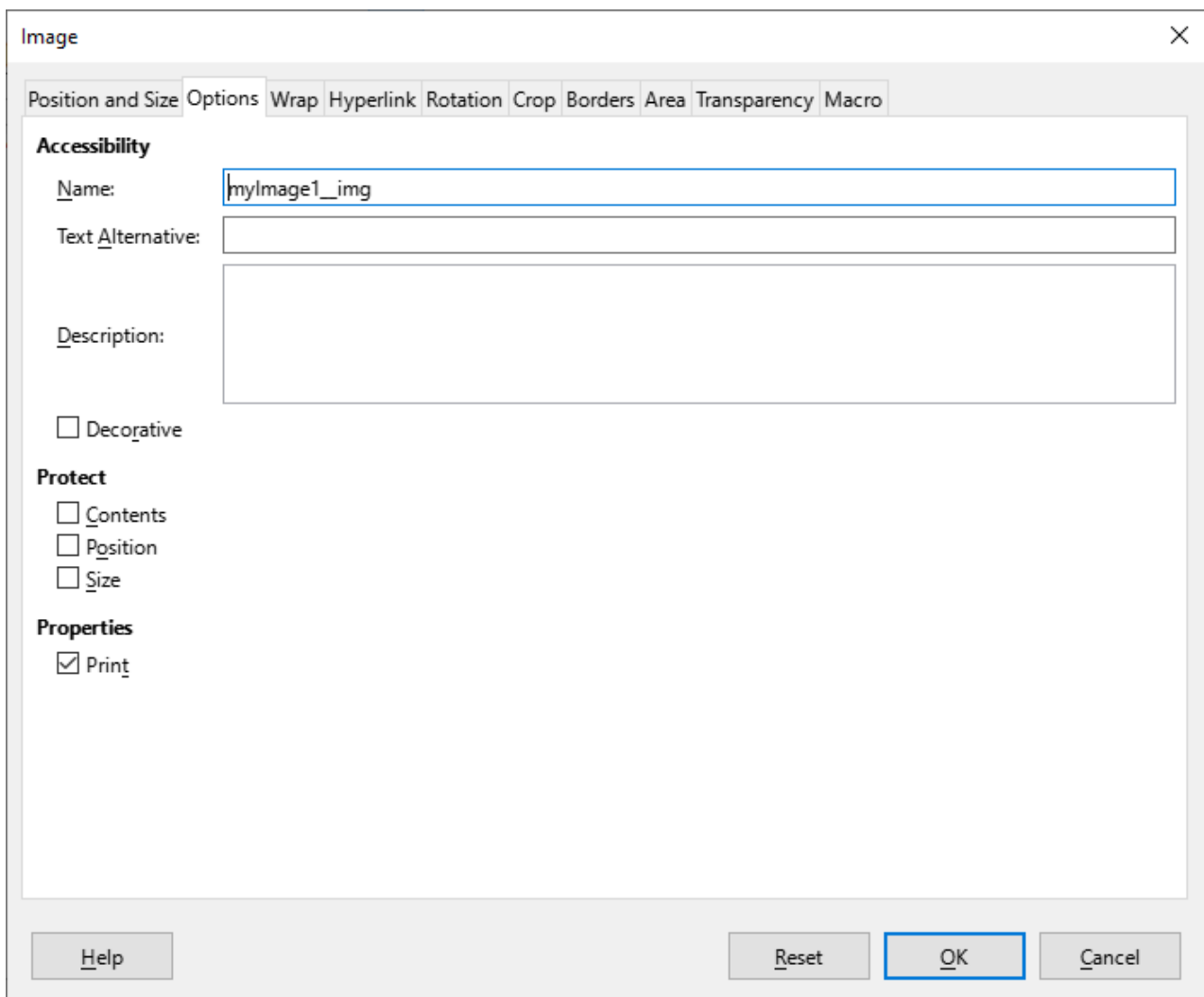
In this case, the variable name is not printed if no value is passed for the myField field in the mapping.



## Insert images

An important task is inserting images. You solve this task by inserting an image into the template as a placeholder. It is recommended that the image you choose is dimensioned as precisely as possible to the images to be integrated using the `_data`. After you have integrated the placeholder image into the template, open the properties dialog (either by double-clicking on the image, or via Format->Image->Properties).

Here you will find the Name property under the Options tab. This must correspond to the node name for the image data in the mapping (without the preceding \$ symbol).



The screenshot shows the 'Image' properties dialog box with the 'Options' tab selected. The 'Name' field is filled with 'myImage1\_img'. The 'Text Alternative' and 'Description' fields are empty. The 'Decorative' checkbox is unchecked. Under the 'Protect' section, 'Contents', 'Position', and 'Size' checkboxes are unchecked. Under the 'Properties' section, the 'Print' checkbox is checked. The dialog has buttons for 'Help', 'Reset', 'OK', and 'Cancel' at the bottom.

Position and Size	Options	Wrap	Hyperlink	Rotation	Crop	Borders	Area	Transparency	Macro
<b>Accessibility</b>									
Name: <input type="text" value="myImage1_img"/>									
Text Alternative: <input type="text"/>									
Description: <input type="text"/>									
<input type="checkbox"/> Decorative									
<b>Protect</b>									
<input type="checkbox"/> Contents									
<input type="checkbox"/> Position									
<input type="checkbox"/> Size									
<b>Properties</b>									
<input checked="" type="checkbox"/> Print									



It should be noted that image objects in the template (unfortunately) require a unique name. It is therefore not possible to use identical image data in different places in the template. If you give the same name twice (at least LibreOffice) the name of the second instance changes. And that silently.

## Working with lists – tables

If you create a JSON array in your mapping, it can be used as a list in the template to fill tables.

Create a table. Starting from the line in which you want to write your list, insert the value `$nodename.fieldname` as a template variable. For example, enter `$myList.myField` in every cell that should be filled in the template.

Static header 1	Static header 2
<code>\$myList.myField1</code>	<code>\$myList.myField2</code>

If you want to use images in the cells, embed a placeholder image and assign the name `node name.field name`, e.g. `myList.myImage__img`. The rules described in the Mapping area apply.

## Working with lists – without tables

If you need a list directly in the text (e.g. for enumerations), you must inform the template that a list should be used at a certain point.

The PostExecuter needs a declaration for the start and end of a list used.

**#foreach**(`$indexVariable` in `$myList`)

• `$indexVariable.fieldName`

**#end**

The declaration starts with the expression **#foreach** followed by an expression in brackets that gives the PostExecuter instructions on how to process the list. This expression consists of an **Indexvariable** that you specify, followed by the specification in which **list** you want to work. The name of the list corresponds to the name of your node in the mapping, preceded by a \$ character.

Within the list, fields from your mapping are referenced by the variable name **\$indexVariable.fieldName**. Example: `$myIndex.myField`

The end of list usage must be announced to the template using the expression **#end**.



A complete example could look like this:

```
#foreach($myIndex in $myList)
$myIndex.myField
#end
```

Lists within lists (without tables)

In principle it is possible to nest lists within each other.

```
#foreach($myIndex in $myList)
$myIndex.myField
#foreach($myIndex2 in $myIndex.myList2)
$myIndex2.myField2
#end
#end
```

The loops are processed in exactly the same way as in the mapping. This is also quite easy to implement in a context outside of table

Lists within lists (in tables)

```
#dragonsLurkHere
```

Things get much more complicated if you want to use nested lists in tables. Tables are, by nature, two-dimensional objects. Introducing further dimensions here is not easy and actually contradicts what we want to achieve with ReporterPE - the most uncomplicated approach to PDF creation.

**Our clear recommendation** is to dissolve the inner loops in the mapping and pass the corresponding values as literals in a variable!

This said: It still works...



Using the `@before-row` and `@after-row` directives allows list entries to be resolved within a table. EACH list that is to be used in the table must then be declared in its OWN table.

This means that if you want to use nested lists in a table, it is necessary to nest tables within each other in the template according to your data structure. Sounds complicated? It is...

The table below shows two lists. The main table has a black frame, the integrated table has a blue one.

@before-row#foreach(\$l1 in \$loop1)\$l1.value1	\$l1.value2
\$l1.value3	<div>@before-row#foreach(\$l2 in \$l1.loop2)\$l2.value@after-row#end</div> <div>@after-row#end</div>

In order to fill cells with values from the nested lists, it is necessary to insert the `@before-row` directive in each of the individual tables and the `@after-row` directive at the end. The directive is followed by declaring the list via `#foreach`, just as you would outside a table.



## Forcing page breaks

If you want to force a page break in your template from the mapping, this is possible using a styled field.

Create a field in your mapping that has the magic extension `__styled` in its name. For example, `pgbrk__styled`. As a fixed value, pass the following content to the field:

```
<p style="page-break-before:always;"></p>
```

Now every time you insert the template variable `$pgbrk__styled` into your template, a page break is forced at this point.

## Scripting within the template

Since version 2.0, basic script commands have been supported within the template. In principle, using a template variable is already a script command. Other options:

If-Then-Else constructs are controlled using the commands `#if` `#else` and `#end`. The condition to be checked is passed to the `#if` command in parentheses. A check to see whether the variable `oofficepath` was passed to the template (including checking whether it is empty) can be implemented as follows:

```
#if( "$!oofficepath"=="")  
oofficepath not set.  
#else oofficepath: $oofficepath  
#end
```

The following expressions are available to you for analyzing the data situation in lists:

`$myList.size()` returns the number of list entries in the list `$myList`.

`$myList.isEmpty()` returns true or false, depending on whether the list is filled with at least one entry or not.

Within a `#foreach` loop, you can check the following values:

`$foreach.hasNext` returns whether there is another list entry after the one you are currently processing.

`$foreach.index` returns the current iteration of the list run, counted from 0

`$foreach.counter` returns the current iteration of the list run, counted from 1

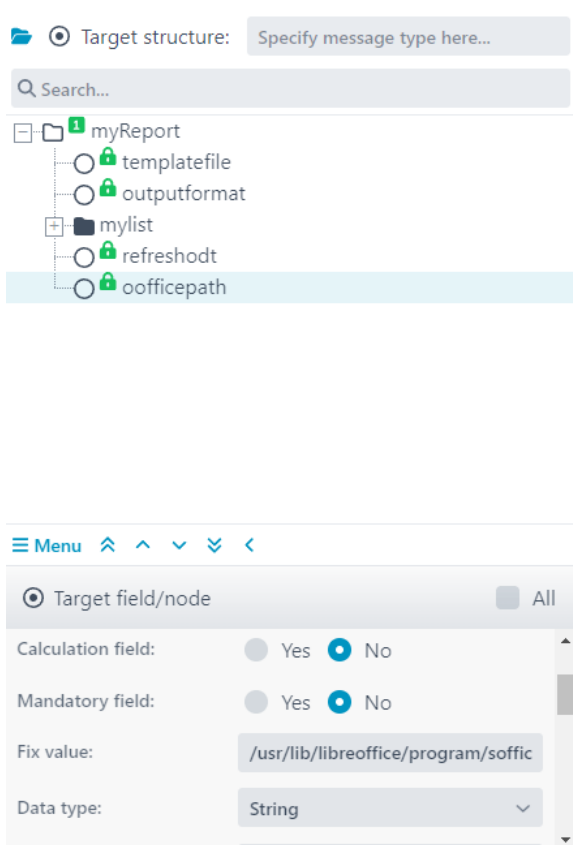
## Dynamic refreshing of the generated ODT document

Sometimes it is necessary to refresh the content of the filled ODT template before generating the PDF. A good example of this is creating tables of contents. ReporterPE allows to automate this process.

For this it is necessary that a LibreOffice instance is installed on the server side. The path to the installation must be known and passed in JSON.

To trigger the server-side refresh of the filled ODT template, expand the mapping by 2 fields:

**refreshodt** with the content true  
**oofficepath** with the path to the installed LibreOffice on your server  
(for example: /usr/lib/libreoffice/program/soffice.bin)



The screenshot shows the ReporterPE configuration interface. At the top, there is a 'Target structure' dropdown set to 'Specify message type here...'. Below it is a search bar. A tree structure is displayed with the following nodes: 'myReport' (expanded), 'templatefile', 'outputformat', 'mylist' (expanded), 'refreshodt', and 'oofficepath'. The 'oofficepath' node is selected. Below the tree is a 'Menu' section with a 'Target field/node' dropdown set to 'All'. The configuration panel includes the following fields: 'Calculation field:' with 'Yes' and 'No' radio buttons ( 'No' is selected ), 'Mandatory field:' with 'Yes' and 'No' radio buttons ( 'No' is selected ), 'Fix value:' with a text input field containing '/usr/lib/libreoffice/program/soffice', and 'Data type:' with a dropdown menu set to 'String'.





## Render engines in comparison

ReporterPE allows you to choose between internal PDF generation and PDF generation via LibreOffice. You will get higher quality PDFs if you let LibreOffice take over the output. You should therefore only use internal PDF generation when processing mass data, i.e. many PDFs in many jobs.

Internal PDF generation requires that you put a lot of effort into creating the templates if you are working with dynamic images. If the dimensions of the placeholder images do not exactly match the size of the actual image data to be used, this can lead to overlaps with the following paragraphs. Deeply nested lists in tables can also lead to display errors in the table frames. Furthermore, not all LibreOffice features are supported when using internal PDF generation.

If you let LibreOffice generate your PDF files, the jobs will run a little slower.

Feature	internal PDF-creation	LibreOffice PDF creation
Tabstops	No	Yes
Unicode Fonts	No	Yes
Textboxes	No	Yes



We hope you enjoy trying out ReporterPE. We will be happy to answer your questions at [helpdesk@derbrill.de](mailto:helpdesk@derbrill.de)

Your derbrill team.