

# coreBOS GenDoc documentation

This is a coreBOS GenDoc documentation template.

The purpose of this template is to gather together in one template most of the commands and important functions of the document generator.

As you can see up to this point, all text formatting features are available, as well as paragraph formatting, footnotes<sup>1</sup>, endnotes<sup>c</sup> and special characters: ©®¼½±

We can even use the drawing capabilities as seen next.



Obviously, tables are fully supported also:

Header 1	Header 2	Header 3	Header 4	Header 5
Combined cells				20
				30
	Even formulas like the sum of cells work:			50

Finally, on this line, I am inserting some open office fields like

- date: 11/18/17
- time: 05:01:27 PM
- author: joe
- total number of pages: 18
- current page: 1

So, before getting in to the details of using information from coreBOS in the templates, which is where the power of this extension shines, I trust you get the idea that it is extremely easy to format your documents as you need simply using the knowledge you already have about text editing and formatting using a text processor. **No need to learn HTML nor CSS!!**

Now let's talk about accessing information inside coreBOS in our templates.

Each template you create **must** be designed to be compiled or merged against a **main entity**. This module determines the related information you can access inside the template.

For example, if we create a template that starts with an account, then we will be able to access, not only the fields on the account but also all the information of related entities like contacts, invoices or opportunities of that account.

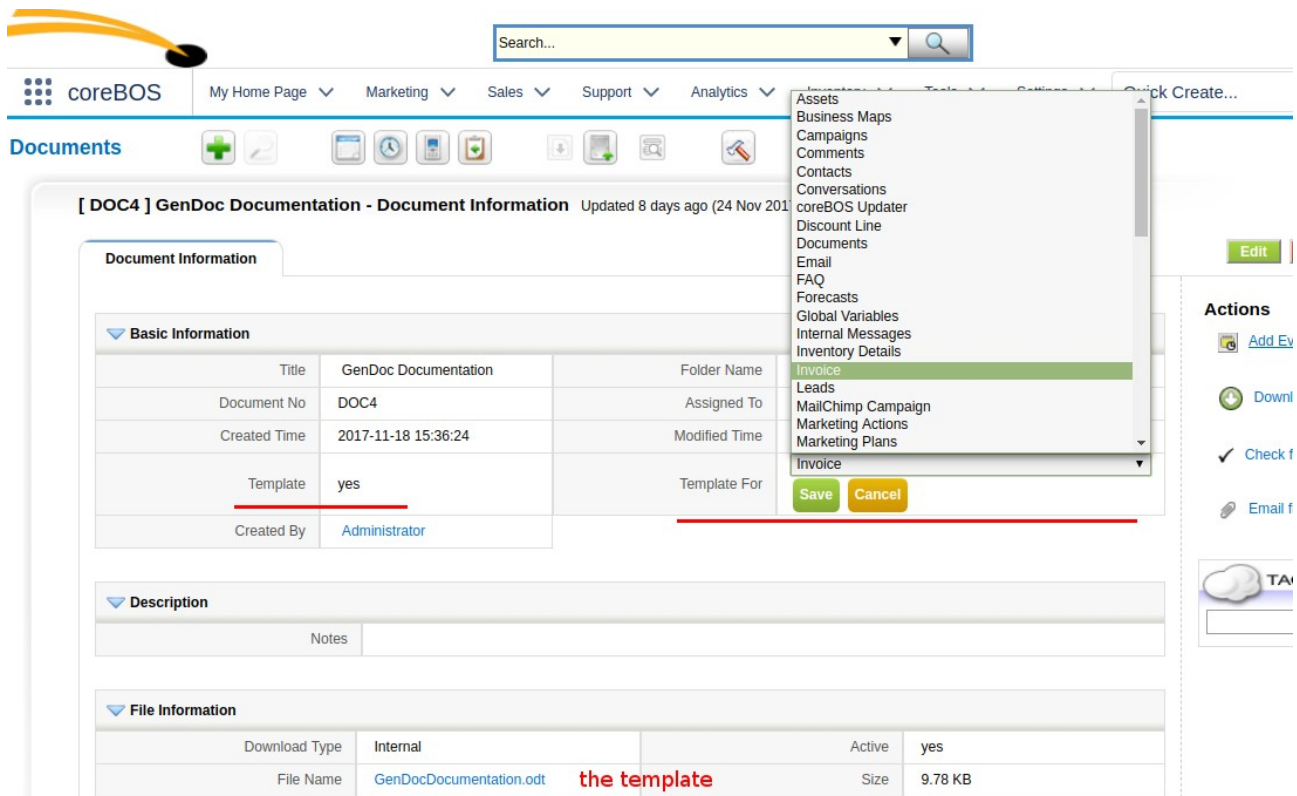
If we start from an invoice, we will be able to access the information of the payments and products but also the account assigned to the invoice and, from there we will be able to access the

---

<sup>1</sup> Even footnotes work with merged labels like Chemex Labs Ltd!!

information of accounts.

This means that each template will need a starting module and record, which is why the documents module, where you upload the templates, now has a module picklist where you indicate this entity. It also has a checkbox that will mark the document as a template, both fields **must** have a value for all templates you upload.



This particular documentation **template is designed to be compiled starting from an invoice** because that will permit us to show **how to access the inventory lines** and also **how to access information from second and third level related entities**.

## Direct Field Access

First let's see how we can access fields in the main module. Since we start from an invoice we will be able to access the fields on the invoice using the syntax `{Invoice.fieldname}` (note here how we can use the marker syntax freely, only labels that are found will be substituted), in general the field substitution marker is `{Entity.field}`, where Entity is the module name and the field is the internal coreBOS field name.

In order to make this easier to type in we have created another extension called GenDoc Labels that will show you all the available fields on each module with their exact marker which you can easily copy into the clipboard and paste into your template.

OK, enough chatter let's see some examples:

Direct Invoice Field	Value
----------------------	-------

Subject:	Sean Cassidy
Invoice Date:	2016-07-16
Status:	AutoCreated
Sub Total:	2.327,72
Total:	2.956,20
Billing City:	Koorlong
Shipping City:	Koorlong
Contact ID:	1,610
Contact Name:	Maxima Brzozowski
Account ID:	74
Account Name:	Chemex Labs Ltd
Assigned To ID:	7
Assigned To Name:	cbTest testymd

Note how we access information on directly related modules by simply using the same marker syntax but putting the name of the module we want to get the information from. It really is that easy, all the information is simply accessible. GenDoc knows which Account or Contact you are referencing.

## User Information Access

Users have a special relation with all the records in the application because there are four different user records we can access at any time: the user who is assigned to the record, the user who created the record, the user who last modified the record and, finally, the current user merging the template.

We can access any of these four users at any time using the virtual modules: Users, CreatedBy, ModifiedBy and CurrentUser.

Assigned User name	cbTest testymd
Creator name	Administrator
Modified by name	Administrator
Current User name	Administrator
Assigned User status	Active
Creator status	Active
Modified by status	Active
Current User status	Active

## Columns and Sections

Columns and sections are also supported as you can see next:

This is the first column:  
Sean Cassidy

This is the second column:  
2016-07-16

This is the third column:  
2.956,20

## Related Module Access

So, now that we know how to access information on the module and record we are merging, our next goal is to access related information. The typical example on the invoice will be to access all of the InventoryDetails records related to the invoice, but we could access the payment records, or the related opportunities of a contact, or project tasks on a project.

Since, in this case, there is **more than one record related** to the main module we are compiling against we cannot just put the module name and field in a marker because the compiler would not know which of all the records we want to show. This is not the case for information on **directly related** modules like accounts (in this case) where there is **only one record** so the compiler knows what to show.

In order to access the information of modules that have many related records, we have to put it inside a **foreach** directive.

Let's get a list of all the InventoryDetails records and some of their fields.

InvDet-000000159	4,00	77,56	310,24
InvDet-000000160	5,00	30,90	154,50
InvDet-000000162	2,00	41,82	83,64
InvDet-000000164	2,00	48,61	97,22
InvDet-000000165	6,00	54,81	328,86
InvDet-000000166	9,00	31,74	285,66
InvDet-000000167	10,00	30,90	309,00
InvDet-000000168	1,00	166,61	166,61
InvDet-000000169	4,00	78,70	314,80
InvDet-000000170	1,00	82,75	82,75
InvDet-000004557	4,00	48,61	194,44

Nice!! Now let's put that inside a table:

Line N°	Quantity	List price	Line Total
InvDet-000000159	4,00	77,56	310,24
InvDet-000000160	5,00	30,90	154,50
InvDet-000000162	2,00	41,82	83,64
InvDet-000000164	2,00	48,61	97,22
InvDet-000000165	6,00	54,81	328,86
InvDet-000000166	9,00	31,74	285,66
InvDet-000000167	10,00	30,90	309,00

InvDet-000000168	1,00	166,61	166,61
InvDet-000000169	4,00	78,70	314,80
InvDet-000000170	1,00	82,75	82,75
InvDet-000004557	4,00	48,61	194,44

Notice how we put one table row inside the **foreach** directive leaving the header outside and also how we have formatted the cells and line tabs to align everything nicely. All that was done using the native text processor functionality, as you would do while creating any other text document you usually create.

So, looking carefully at the syntax we see that the **foreach** directive accepts a module name as a parameter. This indicates the related records we want to retrieve. For example, if we want a list of related payments we would use this construct:

```
Theodore Winchester 2016-07-07 134,00
Madame Mauser      2015-12-11 478,00
Amazon (S.H.E.)    2016-01-10 314,00
Cedaric            2015-05-03 184,00
Magdalena Marie    2016-08-19 462,00
Alex Mitchell      2017-03-03 301,00
Donna Diego        2017-02-20 212,00
```

Using this syntax we can access any related module records.

Now, let's suppose that we want to separate the list of related records on some condition. To accomplish this the **foreach** directive has an extended syntax which permits us to indicate conditions on the records we need the **foreach** to return.

Our first example will retrieve only the not paid payments.

```
Theodore Winchester      2016-07-07      134,00
Donna Diego              2017-02-20      212,00
```

Well, that was easy.

Let's suppose that we want to get all the Inventory Lines with an amount superior to 150. That will be like this:

```
InvDet-000000159      4,00      77,56      310,24
InvDet-000000160      5,00      30,90      154,50
InvDet-000000165      6,00      54,81      328,86
InvDet-000000166      9,00      31,74      285,66
InvDet-000000167     10,00      30,90      309,00
InvDet-000000168      1,00     166,61     166,61
InvDet-000000169      4,00      78,70     314,80
InvDet-000004557      4,00      48,61     194,44
```

Nice!! As you can imagine the normal operators are supported: > < >= <= = !=

But, also there is one additional operator: `in` which can also appear negated: `!in`

Let's suppose we want all inventory lines whose sequence is in the set (1,3,5,7,9). That looks like this:

InvDet-000000159	4,00	77,56	310,24
InvDet-000000162	2,00	41,82	83,64
InvDet-000000165	6,00	54,81	328,86
InvDet-000000167	10,00	30,90	309,00
InvDet-000000169	4,00	78,70	314,80

That last example doesn't seem very useful so let's try to get all the related payments that have a payment mode of cash or credit card.

Madame Mauser	Credit card	478,00
Cedaric	Credit card	184,00

I trust you are as impressed as I am about the power and ease of use of the GenDoc extension, but hold on to your seat, we are just starting!!

Our next step is to see the power of nested **foreach** directives. Let's suppose that we want a list of all the Activities related to the Payments on the Invoice. We simply put one **foreach** directive inside the other, which looks like this:

These are the activities of the payment **Theodore Winchester**

These are the activities of the payment **Madame Mauser**

Pay46 2017-12-02 19:27:00

These are the activities of the payment **Amazon (S.H.E.)**

pay48 2017-12-02 19:16:00

These are the activities of the payment **Cedaric**

These are the activities of the payment **Magdalena Marie**

These are the activities of the payment **Alex Mitchell**

These are the activities of the payment **Donna Diego**

**It is really that easy to go down the whole tree of related records!!**

The nested **foreach** use case is more common than you may think because it has the incredibly powerful side effect of actually starting a new compilation from the record in the current iteration of the loop. Let's see a very typical use case: obtaining product information from the inventory lines.

We have already seen how to loop over the Inventory Details lines and we may be tempted to do something like this:

{Products.productname}	4,00	77,56	310,24
{Products.productname}	2,00	41,82	83,64
{Products.productname}	6,00	54,81	328,86

If you try that you will see that the productname label is not merged. This is because the records are compiled in the context of the Invoice and this record cannot access any product records.

Now we try with a nested foreach:

4Pcs 2.2Bar 32Psi Car Tyre Tire Pressure Valve Stem Caps Sensor 3 Color Eye Air Alert Tire Pressure	4,00	77,56	310,24	
New FULL HD 1080P Car Video Recorder With G-Sensor and 24H Parking mode	41,82	83,64		2,00
Qe0048 Metal Printed Bracelet Watch		6,00	54,81	328,86

With this structure, we will now see the product name correctly. This second level foreach actually launches a whole new merge process starting with the InventoryDetails record. The InventoryDetails record can reach the product fields easily. This is done for each loop with each InventoryDetails record.

Another common use case is to get the related records of a directly related record. Let's suppose that we want a list of all the contacts related to the account on the invoice. To accomplish this we need to use the nested foreach structure even though the first level is directly related. In this case, GenDoc will see that there is no related list for a level and will look for a direct relation. If it finds it, it will use that record as the top level of the second foreach:

Lina  
test

This can even continue nesting inside nesting creating a very powerful and flexible functionality. See if you can understand what the next structure will do before trying the actual merge.

InvDet-000000162 - 83,64

Chemex Labs Ltd

Lina  
test

InvDet-000000164 - 97,22

Chemex Labs Ltd

Lina  
test

InvDet-000000170 - 82,75

Chemex Labs Ltd

Lina  
test

One last thing to notice about the **foreach** directive is that it will be completely eliminated if it cannot be resolved. While field directives will be respected the **foreach** directives will not.

That covers the functionality of the **foreach** directive and we can move on to the next special

directive: **ifexists** and **ifnotexists**.

## Conditional Blocks

The **ifexists** directive permits us to include/exclude whole sections of text depending on the existence of a record or on a condition of some field in the compilation.

For example, this paragraph will appear only on those invoices that have a contact selected. In fact the selected contact is: Maxima Brzozowski

As with the **foreach** directive, we can also use some conditionals like this:

This text will appear if there are one or more inventory lines with a total amount superior to 150

The **ifexists/ifnotexists** directives will permit us to create flexible and adaptable templates that can cover many cases. Instead of creating many similar templates and having the user decide which one to use in each case, we can use these directives to create one template that will decide what to output when needed.

Also, these directives can be combined with the **foreach** directive in order to create extremely functional structures.

This line has a product: 4Pcs 2.2Bar 32Psi Car Tyre Tire Pressure Valve Stem Caps Sensor 3 Color Eye Air Alert Tire Pressure

This line has a service: Annuities

This line has a product: New FULL HD 1080P Car Video Recorder With G-Sensor and 24H Parking mode

This line has a service: River rafting

This line has a product: Qe0048 Metal Printed Bracelet Watch

This line has a product: New Arrival Metal Aluminum Case for iPhone 6

This line has a product: DS-S01 Latest Universal Car Laser Warning Light

This line has a product: U8 Smart Watch

This line has a product: LEGO Enzo Ferrari

This line has a product: Professional USB Factory Logo Printed Colorful Bracelet USB Flash Drive

This line has a product: K101high Pressure Japanese Stainless Steel Berring Turbo Tornado Gun

## Image Field Access

The next directive we are going to visit is **image**. This directive will permit us to add the image contents of any image field in the application directly in our final merged document. The way this works is that we will put on one line by itself the directive with the field specification we want to get the image from and right after the directive, on the next line, we will add an image in the template. The line with the directive will be eliminated and the image contents will be substituted with the image from the application.

The big advantage of this way of doing it is that we can use all the power of the word processor to format our image as we need. Since the GenDoc extension will only change the image contents, all the settings you establish in the image you add will be respected. Note that the image that you



upload is irrelevant, it is just a placeholder, BUT all images you add MUST be different inside the same template, this is because OpenOffice detects that the image is the same and stores a reference for all the repeated ones in order to optimize the size of the final document. When that happens and we substitute the image, the reference gets lost producing an undesired result.

Here are a few examples. One is loading the image of the contact related to the invoice and the other is a table showing the image of the products. Obviously, you will have to upload images to the records to get results in this section.



Product: New FULL HD 1080P Car Video Recorder With G-Sensor and 24H Parking mode



Product:

## Product: Professional USB Factory Logo Printed Colorful Bracelet USB Flash Drive

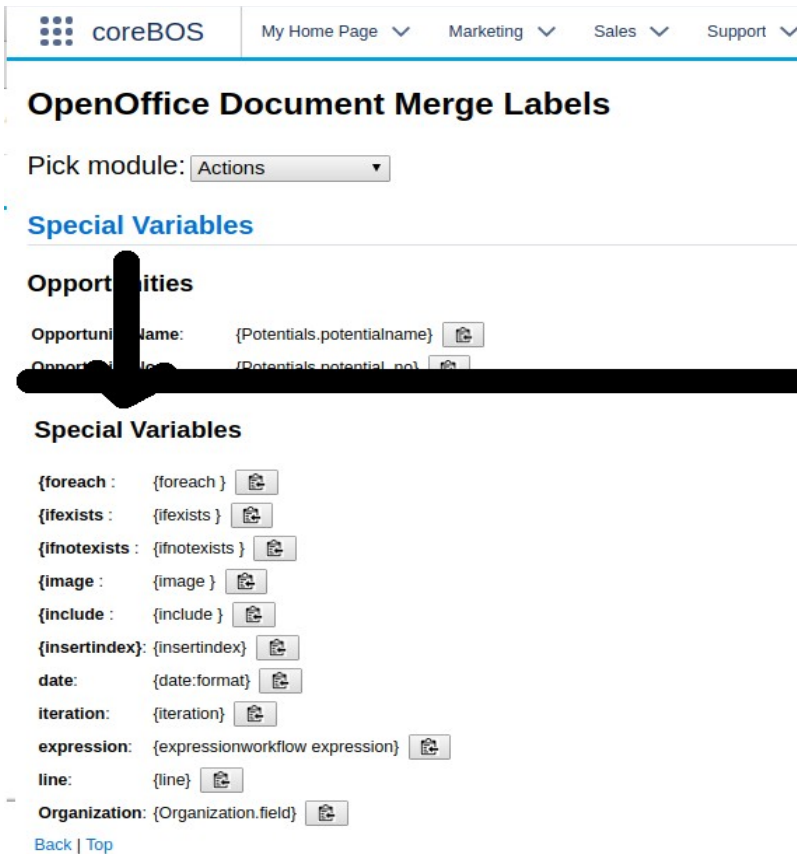


This image inclusion functionality reminds me of just how user-friendly the GenDoc extension is. Note how you can use the existing formatting features available in the word processor to define how your image should appear, the same way you define how your tables, paragraphs and any other elements you want in your final document. There is no need for advanced HTML and CSS knowledge, just useful word processing knowledge which is much more common and easy. The placeholder image I used for the contact image, which is centered and has a curious pink border, is totally on purpose with a very clear message :-)

All image fields are supported. Even the multi-image field of products. For this special non-standard field only one image will be used. There is no way of specifying which one will be used, one will be randomly picked.

### Special GenDoc Labels

Finally, there are also some special labels that we can use that are native to the GenDoc extension. These labels can be found on the Merge Labels extension under the “Special GenDoc Labels” section as you can see in the next image (which also serves as an example that images included in your template will be respected as the one above with Document template fields2).



You can see these labels in action in the next paragraphs.

The **fecha** or **date** directive will output the date the template was merged. All date fields that are on records in coreBOS will be output in the format of the user doing the combination, but this special label will output the format d-m-Y unless indicated otherwise by the extended format specification.

date	24-01-2018
date:Y-m-d	2018-01-24
date:m-Y-d	01-2018-24

Note that the **date** directive has been superseded by the more powerful **expression** directive (see below) but it is maintained for backward compatibility and will be available for as long as the GenDoc extensions exists, so you can use it freely.

<hr />

The next special label es **linea** or **line**. Here you can see one above and below this paragraph.

<hr />

Which is clearly different from the native horizontal line you can add directly using the editor.

Our next special label is **iteration** or **repeticion**, which must be used inside a **foreach** loop directive and will give an ordinal count of the loop iteration we are in. So supposing that the invoice you are merging this template with has 10 product lines this next loop will give you a count to ten:

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11

Finally, our last special label is an extremely powerful one that is a total game changer. The label's name is `expression` and it permits us to tap into all of the functionality that already exists in the coreBOS workflow expression language. So if you know how to define a workflow expression you also know how to add one here. This is basically custom coding right in your template using a language you already know how to work with (or you should know).

Let's see some examples:

<code>expressionuppercase(subject)</code>	SEAN CASSIDY
<code>expressionadd_days(invoicedate,2)</code>	2016-07-18 ; 2016-07-16
<code>expressionif invoicestatus == 'Accepted' then 'GOOD' else 'BAD' end</code>	BAD
<code>expressionhdnSubTotal / 2</code>	1163.86 ; 2.327,72
I think you get the idea :-)	

Now that we know that we can access all the power of the application workflow expressions, we have to revisit the **foreach** directive.

As we can see in the examples above we can access all the fields on the current record (the invoice in this template) directly and also any directly related module field using the workflow syntax, like this:

`HTTP://WWW.CHEMEXLABSLTD.COM.AU`

So it is a slight change in mindset between how GenDoc accesses directly related module fields and how workflow expressions do it, but the goal is accomplished in either case.

Note that you can use the workflow expression editor to help you construct the expressions.

The real problem appears when we try to launch an expression directive inside of a `foreach` directive.

In this case, we may want to access fields on the record that is in the current iteration of the loop, fields from the main record (Invoice) or we may need to access both types of fields.

For example, read this next `foreach`

InvDet-000000159  
InvDet-000000160  
InvDet-000000162  
InvDet-000000164  
InvDet-000000165  
InvDet-000000166  
InvDet-000000167  
InvDet-000000168  
InvDet-000000169  
InvDet-000000170  
InvDet-000004557

The expression above is launched in the context of the invoice, so it looks for the `inventorydetails_no` field on the invoice, it doesn't find it and the workflow expression system, which is rather robust, simply returns an empty string. This is not what we need.

We can't use the GenDoc syntax like this:

```
expressionuppercase(InventoryDetails.inventorydetails_no)
```

because the workflow expression system does not understand that syntax.

The first solution that comes to mind is to launch the expression in the context of the current record of the loop, but that is insufficient because you could need to access information on the main record and not be able to because the workflow system does not know how to access parent records on many to many relations. This is a marginal use case, but still a valid one.

So we decided to pass the expression through the GenDoc compiler before sending it to the workflow system. This way you can access all the same information you can already access inside the `foreach` directive.

For this to work there are two things you need to keep in mind:

- 1.- we must substitute the open and closing keys with two other symbols so they don't get detected on the first compile. We use `~` for opening and `¬` for closing.
- 2.- we are actually substituting the field values with their value, the fields never make it to the workflow system, only the values, so fields cannot be used directly, they must be enclosed by quotes.

Let's see how that looks.

```
InvDet-000000159 INVDET-000000159  
InvDet-000000160 INVDET-000000160  
InvDet-000000162 INVDET-000000162  
InvDet-000000164 INVDET-000000164  
InvDet-000000165 INVDET-000000165  
InvDet-000000166 INVDET-000000166  
InvDet-000000167 INVDET-000000167  
InvDet-000000168 INVDET-000000168  
InvDet-000000169 INVDET-000000169
```

InvDet-000000170 INVDET-000000170  
InvDet-000004557 INVDET-000004557

As you can see we put the field in quotes because the actual value will be sent to the workflow system, not the field and we use the special symbols to create the marker.

## Company Information

We can also access the company information configured in the Settings section of the application using the meta label Organization as you can see in the next table.

Organization.organizationname	Your Company
Organization.address	Your Address
Organization.city	Your City

Since the information available on the company settings section is rather limited and you could need more, the install process created the global variable **GenDoc\_Company\_Account** where you can set the accountid of any Accounts record from which you wish to access your company's information. The idea is that you create an account record for your company and use all the fields there. You can try giving this global variable a value and merging this template again to see that the values below change accordingly.

Organization.siccode	654987321
Organization.bill_address	864 Aupuni St
Organization.ship_city	St Catharines

## Advanced Features

There are a few additional advanced features which I will not cover in this tutorial because they are not supported natively in the basic version. These features require a special openoffice install on a server in order to generate the final document. These features are:

include	Which permits us to include another template inside the one we are merging. This is very useful when generating complex contracts or customer documentation and need to load different types of conditions depending on some field values. When this directive is found the depending template will be loaded and merged inside the same context the main template is and, at the end all the templates will be sent to the open office server where they will be combined into one final document.
insertindex	This directive will update an index that you have defined in your document. If you add an automatic index in your document, obviously all the page numbers are going to change as the different directives get converted into their actual value, so you need to open the final document and update the index. When this directive is found the final document is sent to the special open office server where it will be opened and the index updated for you.

If you think you need any of these services or you need something that cannot be done with the existing functionality, contact us for support.

## Language Support

The GenDoc directives can be easily translated into any language by copying the `modules/evvtgendoc/commands_en.php` file to your language and translating the labels accordingly.

A template will look for directives in the language of the current user. If the template was not designed for that language it will not work.

This makes the extension a little more user-friendly and usable by non-technical users.

You can compile any template in any language using the GenDoc direct compiler functionality which can be found by accessing the `evvtgendoc` module directly.

## Closing Notes.

I hope I have been able to illustrate all the power and ease of use we have created with this extension. Your next step in the process of using GenDoc is to start creating your own templates.

I would like to end this tutorial informing you that there is a GenDoc template store available on the coreBOS documentation site and encouraging you to share your templates with us there.




<http://corebos.org/documentation/doku.php?id=en:gendoc:templatestore>

You can find some additional information about the labels and rules of usage in our documentation wiki:


<http://corebos.org/documentation/doku.php?id=en:extensions:extensions:gendoc>

You can ask for help in the forum <http://discussion.corebos.org> and in the gitter group <https://gitter.im/corebos/discuss> or you can contract us to construct your template or for direct support which always helps to support our effort and make a better product.

I leave here one last example that combines most of the features in one nice inventory lines table.

Product		Quantity	Price	Total
4Pcs 2.2Bar 32Psi Car Tyre Tire Pressure Valve Stem Caps Sensor 3 Color Eye Air Alert Tire Pressure <b>**Discount!**</b>		4,00	77,56	310,24
Annuities		5,00	30,90	154,50
New FULL HD 1080P Car Video Recorder With G-Sensor and 24H Parking mode Add a few more units to get a discount		2,00	41,82	83,64
River rafting		2,00	48,61	97,22
Qe0048 Metal Printed Bracelet Watch <b>**Discount!**</b>		6,00	54,81	328,86
New Arrival Metal Aluminum Case for iPhone 6 <b>**Discount!**</b>		9,00	31,74	285,66
DS-S01 Latest Universal Car Laser Warning Light <b>**Discount!**</b>		10,00	30,90	309,00



<p>U8 Smart Watch  <b>**Discount!**</b></p>		1,00	166,61	166,61
<p>LEGO Enzo Ferrari  <b>**Discount!**</b></p>		4,00	78,70	314,80
<p>Professional USB Factory  Logo Printed Colorful  Bracelet USB Flash Drive  Add a few more units to get a  discount</p>		1,00	82,75	82,75
<p>K101high Pressure  Japanese Stainless Steel  Berring Tubo Tornado  Gun  <b>**Discount!**</b></p>		4,00	48,61	194,44

**Happy merging!**

⌘This is an endnote with a label for the account: Chemex Labs Ltd, that we added on the first page