

[Link para a versão original no notion](#)

## Resumo

Neste artigo serão abordadas boas práticas para a geração do BOM utilizando o software KiCad 6.

O objetivo principal é reduzir o tempo de revisão e preparação do arquivo que é enviado para a cotação da PCB e montagem, além de enriquecer a documentação e diminuir a ocorrência de erros no BOM.

Para isto serão abordados os seguintes tópicos:

1. Utilização de campos personalizados nos componentes.
2. Edição do script de geração de BOM do próprio KiCad tornando possível a geração de BOM para variantes de montagem.
3. Utilização do plugin InteractiveBOM.

Executando estas etapas durante o desenvolvimento, o projetista conseguirá obter um BOM confiável e uma excelente documentação para a montagem, com a possibilidade de prever variações na montagem.

## Sumário

1. Pré-requisitos para este tutorial
2. Introdução
3. Configuração da folha
4. BOM .csv
  1. Script tradicional
  2. Script modificado
5. Limitações
6. Interactive BOM
7. Arquivos do projeto
8. Arquivos de output

## Pré requisitos para este tutorial:

- Instalação do KiCad > 6.0.4
  - **Version: 6.0.4-6f826c9f35~116~ubuntu20.04.1, release build**
  - **Atenção: a versão 6.0.3 possui um bug ao importar as ligações do esquemático para a PCB.**
- Adicionar o plugin InteractiveHtmlBom na pasta **/usr/share/kicad/scripting/**

# Introdução

A primeira regra para não ter dor de cabeça ao gerar o BOM é criar seus próprios componentes. Para isto será necessário criar uma biblioteca própria e adicioná-la na lista de bibliotecas do seu kicad.

Para criar uma nova biblioteca:

1. Tools→Symbol Editor
2. File→ New Library
  - a. Global

Para criar um novo símbolo nesta biblioteca:

1. File→New symbol

Dentro do novo símbolo será necessário adicionar campos adicionais após desenhá-lo, para isto basta clicar no fundo com o botão direito e selecionar a opção “symbol properties...”.

Na tela de propriedades do símbolo, será necessário adicionar pelo menos 5 novos campos além dos obrigatórios, para adicionar basta clicar no botão “+” e uma nova linha é adicionada à tabela:

- **PN:** Part number completo do item
- **Fabricante:** De preferência utilizar componentes passivos do mesmo fabricante.
- **Descrição detalhada**
- **Link:** Link de algum fornecedor confiável (ie. Mouser)
- **DNPX:** O valor deve ser DNPX ou em branco, onde X é o número da variante.
- **DNPY:** O valor deve ser DNPY ou em branco, onde Y é o número da variante.

O campo DNP significa “DO NOT PLACE” e o X é um número ou letra que identifica a variante da montagem. O DNP é editado no momento em que o componente é adicionado ao esquemático, pois muda de projeto para projeto.

É importante deixar os campos DNP visíveis para que não seja necessário ter que entrar nas propriedades do componente para ver seu valor. Ele será utilizado tanto para gerar o iBOM quanto para gerar o BOM .csv. O iBOM, como veremos a seguir, será um documento adicional para o departamento de montagem da placa e assistência técnica dos produtos, já o BOM .csv é utilizado para a seleção de PNs (part-numbers) para a montagem.

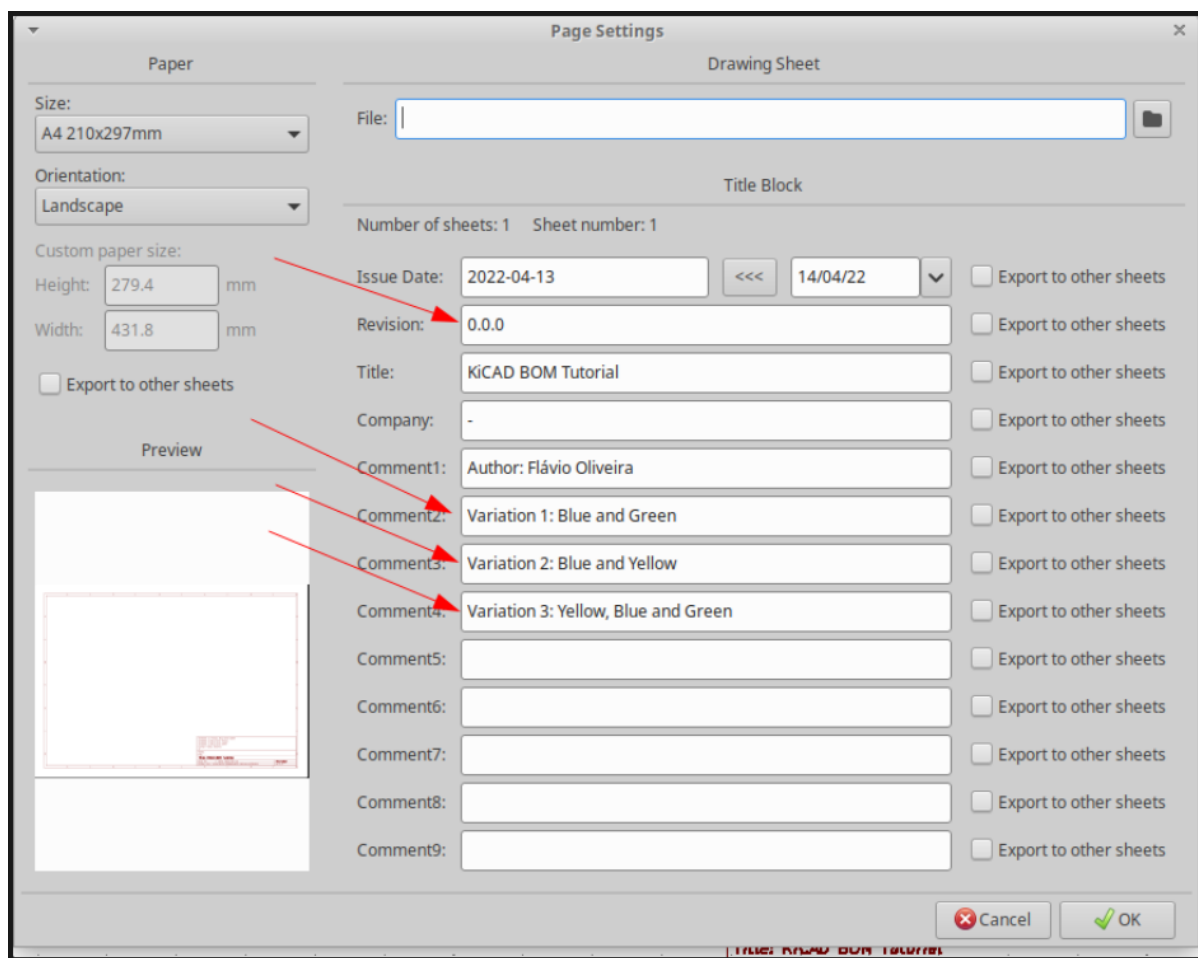
Caso o componente seja montado em todas as variantes, não se torna necessário criar o campo DNP para o mesmo.

Uma sugestão para não ter que desenhar sempre o footprint dos componentes é baixar do site: <https://www.snapeda.com/home/>

Em seguida, personalizar os campos do símbolo, removendo as linhas dos layers que não são utilizados.

## Configuração básica da folha

File→Page Settings



## BOM .csv

## Script tradicional

O BOM é gerado normalmente através de scripts em python que ficam localizados na pasta:

- /usr/share/kicad/plugins/

Para rodar um script que gera o BOM é preciso estar na tela do esquemático e ir no menu “tools→Generate BOM”, selecionar o script, modificar os parâmetros que são enviados ao script pela linha de comando (quando necessário) e clicar em generate.

O script mais utilizado para geração de BOM é o “**bom\_csv\_grouped\_by\_value.py**” onde são agrupados os componentes pelo campo “value”.

Isto é útil quando todos os componentes que estão no esquemático devem ser montados na placa e os campos personalizados não tem relevância.

Porém, isto raramente ocorre na prática. Na maioria das vezes são previstos componentes para fazer variações do mesmo produto ou até mesmo para montar outros produtos com funcionalidades muito diferentes. Neste caso precisaríamos editar manualmente o .csv gerado pelo script e todos sabemos que isto abre uma margem enorme para erros.

O ideal seria que um campo personalizado indicasse ao script (gerador de BOM) qual item deve ser montado, ou não montado, dependendo da variante que se deseja gerar no BOM.

Abaixo está o resultado do BOM gerado pelo script tradicional.

Item	Qty	Reference(s)	Value
1	1	CN1	CON_PIN_HEADER_1X6_2.54_180
2	3	LD1, LD2, LD3	LED_AMARELO_0805_L171L-YGC
3	3	LD4, LD5, LD6	LED_AZUL_0805_L171L-QBC-TR
4	3	LD7, LD8, LD9	LED_VERDE_0805_L171L-GC-TR
5	3	R1, R2, R3	R_0805_1K_1%_1/8W
6	3	T1, T2, T3	T_NPN_SOT23_BC817

## Script modificado

Os scripts geram um .csv mas não contemplam a possibilidade de variações de montagem. Para possibilitar a geração de um BOM para cada variante de montagem é necessário editar o script.

Script python modificado para geração de BOM:

```
#
# Example python script to generate a BOM from a KiCad generic netlist
#
# Example: Sorted and Grouped CSV BOM
#
"""
    @package
    Generate a csv BOM list.
    Components are sorted by ref and grouped by value
    Fields are (if exist)
    Item, Qty, Reference(s), Value, LibPart, Footprint, Datasheet

    Command line:
    python "pathToFile/bom_csv_grouped_by_value.py" "%I" "%O.csv" "variation"
"""

from __future__ import print_function

# Import the KiCad python helper module and the csv formatter
import kicad_netlist_reader
import csv
```

```

import sys

if len(sys.argv) != 4:
    print("Usage ", __file__, "<generic_netlist.xml> <output.csv> <variant>",
          file=sys.stderr)
    sys.exit(1)

variant = sys.argv[3]
print('Variante = ', end='')
print(variant)

dnp_field = 'DNP'+variant
print('DNP = ', end='')
print(dnp_field)

def myEqu(self, other):
    """myEqu is a more advanced equivalence function for components which is
    used by component grouping. Normal operation is to group components based
    on their value and footprint.

    In this example of a custom equivalency operator we compare the
    value, the part name and the footprint.
    """
    result = True
    if self.getValue() != other.getValue():
        result = False
    elif self.getPartName() != other.getPartName():
        result = False
    elif self.getFootprint() != other.getFootprint():
        result = False
    elif dnp_field in self.getFieldNames():
        if dnp_field in other.getFieldNames():
            if self.getField(dnp_field) != other.getField(dnp_field):
                result = False
        else:
            result = False
    else:
        if dnp_field in other.getFieldNames():
            result = False

    return result

# Override the component equivalence operator - it is important to do this
# before loading the netlist, otherwise all components will have the original
# equivalency operator.
kicad_netlist_reader.comp.__eq__ = myEqu

# Generate an instance of a generic netlist, and load the netlist tree from
# the command line option. If the file doesn't exist, execution will stop
net = kicad_netlist_reader.netlist(sys.argv[1])

# Open a file to write to, if the file cannot be opened output to stdout
# instead
try:
    f = open(sys.argv[2], 'w')
except IOError:
    e = "Can't open output file for writing: " + sys.argv[2]
    print(__file__, ":", e, sys.stderr)
    f = sys.stdout

# subset the components to those wanted in the BOM, controlled
# by <configure> block in kicad_netlist_reader.py
components = net.getInterestingComponents()

compfields = net.gatherComponentFieldUnion(components)
partfields = net.gatherLibPartFieldUnion()

```

```

# remove Reference, Value, Datasheet, and Footprint, they will come from
'columns' below
partfields -= set( ['Reference', 'Value', 'Datasheet', 'Footprint'] )

columnset = compfields | partfields      # union

# prepend an initial 'hard coded' list and put the enchillada into list 'columns'
#print(columnset)

columns = ['Item', 'Qty', 'Reference(s)', 'Value', 'LibPart', 'Footprint',
'Datasheet'] + sorted(list(columnset))

# Create a new csv writer object to use as the output formatter
out = csv.writer( f, lineterminator='\n', delimiter=',', quotechar='"',
quoting=csv.QUOTE_ALL )

# override csv.writer's writerow() to support encoding conversion (initial
encoding is utf8):
def writerow( acsvwriter, columns ):
    utf8row = []
    for col in columns:
        utf8row.append( str(col) ) # currently, no change
    acsvwriter.writerow( utf8row )

# Output a set of rows as a header providing general information
writerow( out, ['Source:', net.getSource()] )
writerow( out, ['Date:', net.getDate()] )
writerow( out, ['Component Count:', len(components)] )

# Output all the interesting components individually first:
row = []

writerow( out, ['Item', 'Qty', 'Designator', 'PN', 'Valor', 'Detailed
Description', 'Fabricante', 'Link'] ) # reuse same columns

# Get all of the components in groups of matching parts + values
# (see kicad_netlist_reader.py)
grouped = net.groupComponents(components)

# Output component information organized by group, aka as collated:
item = 0
for group in grouped:
    del row[:]
    refs = ""

    # Add the reference of every component in the group and keep a reference
    # to the component so that the other data can be filled in once per group
    for component in group:
        if len(refs) > 0:
            refs += ", "
        refs += component.getRef()
        c = component

    # Fill in the component groups common data
    # columns = ['Item', 'Qty', 'Reference(s)', 'Value', 'LibPart', 'Footprint',
'Datasheet'] + sorted(list(columnset))

    if net.getGroupField(group, dnp_field) != 'X':
        item += 1
        row.append(item )
        row.append(len(group) ) # quantidade
        row.append(refs )
        row.append(net.getGroupField(group, 'PN'))
        row.append(c.getValue())
        row.append(net.getGroupField(group, 'Detailed Description'))
        row.append(net.getGroupField(group, 'Fabricante'))
        row.append(net.getGroupField(group, 'Link'))

```

```
writerow(out, row)

f.close()
```

No linux o diretório dos plugins é protegido contra escrita, portanto é necessário criar o arquivo utilizando o comando sudo.

```
cd /usr/share/kicad/plugins/
sudo nano bom_csv_grouped_by_value_with_variation.py
```

Ao abrir o editor nano, vamos colar o script modificado, sair e gravar as modificações através do comando ctrl + X, seguido de Y para confirmar a gravação.

## Adicionando o novo script na lista

- Na tela de scripts, clique no botão +
- Selecione o arquivo bom\_csv\_grouped\_by\_value\_with\_variation.py
- Ajuste a linha de comando para ficar de acordo com **"/usr/bin/python3"**  
**"/usr/share/kicad/plugins/bom\_csv\_grouped\_by\_value\_with\_variation.py" "%I"**  
**"%O3.csv" "3"**
  - Perceba que %O é uma alias default para o diretório do projeto e o último parâmetro é o caractere que utilizaremos para identificar a variante.

## Executando o novo script



A linha de comando para gerar os BOM com variante é a seguinte:



```
"/usr/bin/python3" "/usr/share/kicad/plugins/bom_csv_grouped_by_value_with_variation.py"
"%I" "%O1.csv" "1"
```



```
"/usr/bin/python3" "/usr/share/kicad/plugins/bom_csv_grouped_by_value_with_variation.py"
"%I" "%O2.csv" "2"
```

```
"/usr/bin/python3" "/usr/share/kicad/plugins/bom_csv_grouped_by_value_with_variation.py"
"%I" "%O3.csv" "3"
```

Arquivos de saída:

Source:	/home/.....bom_kicad/bom_kicad.kicad_sch			
Date:	Wed 13 Apr 2022 23:22:15 -03			
Variant:	1 			
Item	Qty	Designator	PN	Value
1	1	CN1	PH1-06-UA	Component
2	3	LD4, LD5, LD6	L171L-QBC-TR	LED
3	3	LD7, LD8, LD9	L171L-GC-TR	LED
4	2	R2, R3	RMCF0805FT1K00	Resistor
5	2	T2, T3	BC817-40,215	Transistor
Component Count:	11 			

Source:	/home/...../bom_kicad/bom_kicad.kicad_sch			
Date:	Wed 13 Apr 2022 23:22:25 -03			
Variant:	2 			
Item	Qty	Designator	PN	Value
1	1	CN1	PH1-06-UA	Component
2	3	LD1, LD2, LD3	L171L-YGC	LED
3	3	LD4, LD5, LD6	L171L-QBC-TR	LED
4	2	R1, R2	RMCF0805FT1K00	Resistor
5	2	T1, T2	BC817-40,215	Transistor
Component Count:	11 			

Source:	/home/.....om_kicad/bom_kicad.kicad_sch			
Date:	Wed 13 Apr 2022 23:22:33 -03			
Variant:	3 			
Item	Qty	Designator	PN	Value
1	1	CN1	PH1-06-UA	Component
2	3	LD1, LD2, LD3	L171L-YGC	LED
3	3	LD4, LD5, LD6	L171L-QBC-TR	LED
4	3	LD7, LD8, LD9	L171L-GC-TR	LED
5	3	R1, R2, R3	RMCF0805FT1K00	Resistor
6	3	T1, T2, T3	BC817-40,215	Transistor
Component Count:	16 			



# Perguntas e limitações

1. E quando uma variante de montagem mudar apenas o valor do componente, como por exemplo um resistor, capacitor ou indutor?
  1. É sempre indicado que se adicione os dois componentes, no caso eles ficariam em paralelo.
2. E se existirem restrições de espaço?
  1. Neste caso é necessário usar um “artifício” criando os dois componentes no esquemático, porém na PCB estes componentes serão sobrepostos. Esta prática não é recomendada e me coloco a disposição para que comentem com sugestões para contornar esta limitação. No DRC isto vai ser marcado como um erro grave.
3. Como é feito o versionamento dos outputs? No caso de modificar apenas uma montagem?
  1. Uma alteração em uma montagem implica em uma modificação no esquemático, ou seja, será necessário editar campos de DNP ou trocar o componente. O esquemático deverá ser revisado em qualquer um dos casos. A ideia é que os outputs gerados estejam vinculados a uma versão do esquemático. Sugere-se a criação de todos os outputs novamente, mesmo que fiquem idênticos à versão anterior para assegurar “sincronismo” entre o arquivo gerado e o esquemático que o gerou. Um script pode auxiliar na automatização desta tarefa, mas isto é assunto para outro artigo.

## Interactive BOM

O interactive BOM é um plugin que pode servir para o departamento de assistência técnica / manutenção e também pode ser muito útil para o montador da placa.

O arquivo gerado pelo plugin é um html interativo onde é possível identificar a localização de componentes, consultar PN, Fabricante, se é ou não montado e como ele se conecta com os demais componentes na placa.

Para gerar o arquivo é necessário antes exportar o netlist do esquemático.

## Gerando netlist

O netlist é gerado no esquemático a partir do menu File→Export→Netlist.

## Baixando o Plugin

<https://github.com/openscopeproject/InteractiveHtmlBom>

No linux (ubuntu), colocar a pasta do plugin inteiro no diretório:

`/usr/share/kicad/scripting/`

```
sudo su
cp -r InteractiveHtmlBom/ /usr/share/kicad/scripting
```

## Gerando o HTML

O HTML é gerado a partir da tela da PCB no menu “Tools→External Plugins→Generate Interactive HTML BOM”

O “**Name format**” é o nome que o arquivo terá, é recomendado adicionar ao final do nome o número da variante que ele está relacionado.

Importante marcar estes dois check boxes pois a informação das trilhas da PCB pode ser muito útil para o departamento de assistência técnica / manutenção.

General    Html defaults    Fields

Bom destination

Directory

Name format  ?

Additional pcb data

☒ Include tracks/zones    ☒ Include nets

Component sort order

C  
R  
L  
D  
U  
Y  
X  
F  
SW

Component blacklist

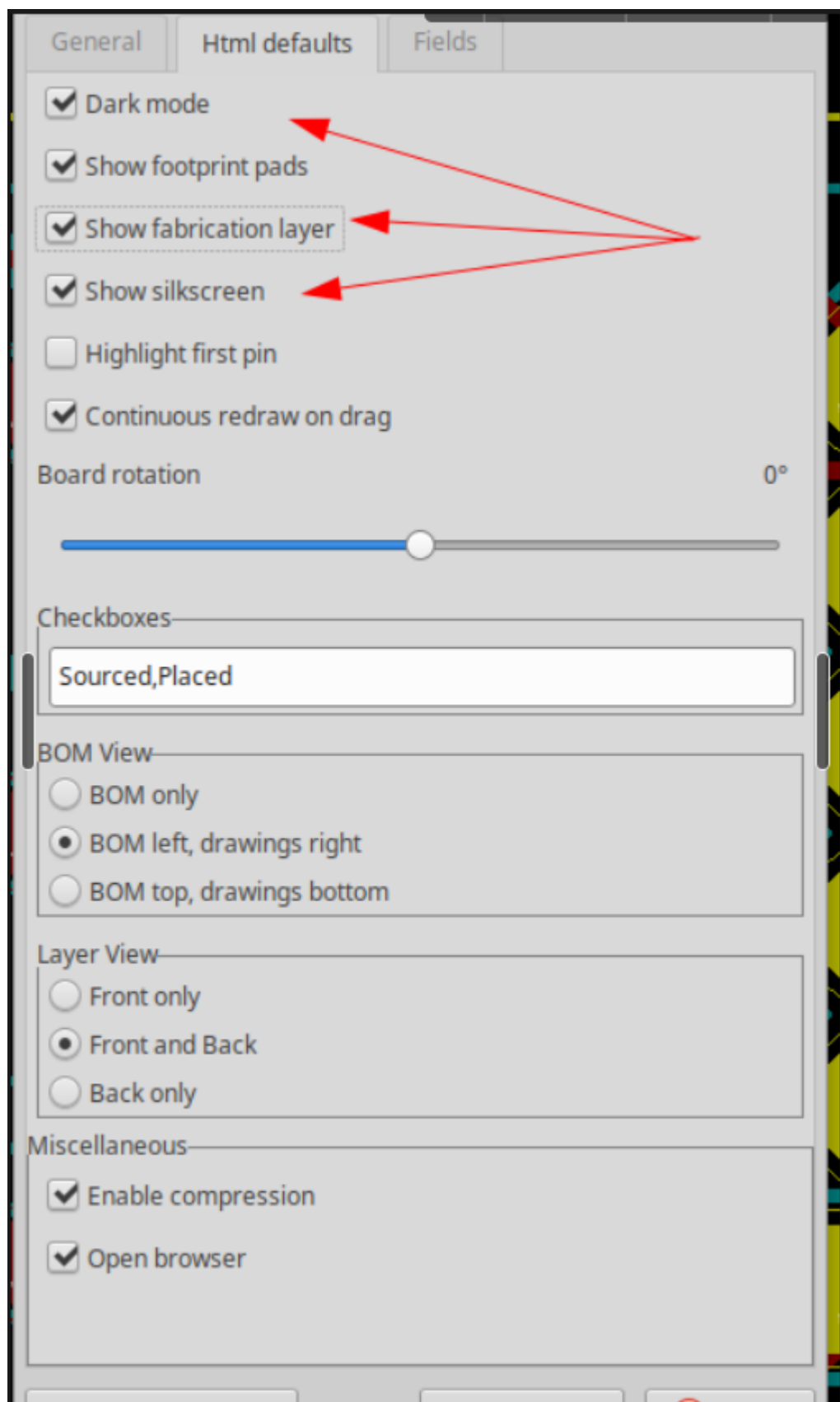
Globs are supported, e.g. MH\*

☒ Blacklist virtual components

☐ Blacklist components with empty value

Save current settings    Generate BOM    Cancel

Algumas opções podem ser adicionadas no wizard, isto vai de acordo com as preferências de cada usuário.



Adicionar os campos PN, Fabricante e Descrição podem ajudar bastante o montador e a pessoa responsável pela configuração da pick and place.

No campo DNP seleccione o campo referente a variante que será montada.

InteractiveHtmlBom v2.4.1

General    Html defaults    **Fields**

Extra data file

Fields

Show	Group	Name
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Value
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Footprint
<input type="checkbox"/>	<input type="checkbox"/>	DNP1
<input type="checkbox"/>	<input type="checkbox"/>	DNP2
<input type="checkbox"/>	<input type="checkbox"/>	Datasheet
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Description
<input type="checkbox"/>	<input type="checkbox"/>	Detailed Description
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Fabricante
<input type="checkbox"/>	<input type="checkbox"/>	Link
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	PN

☐ Normalize field name case

Board variant

Board variant field name

<none>

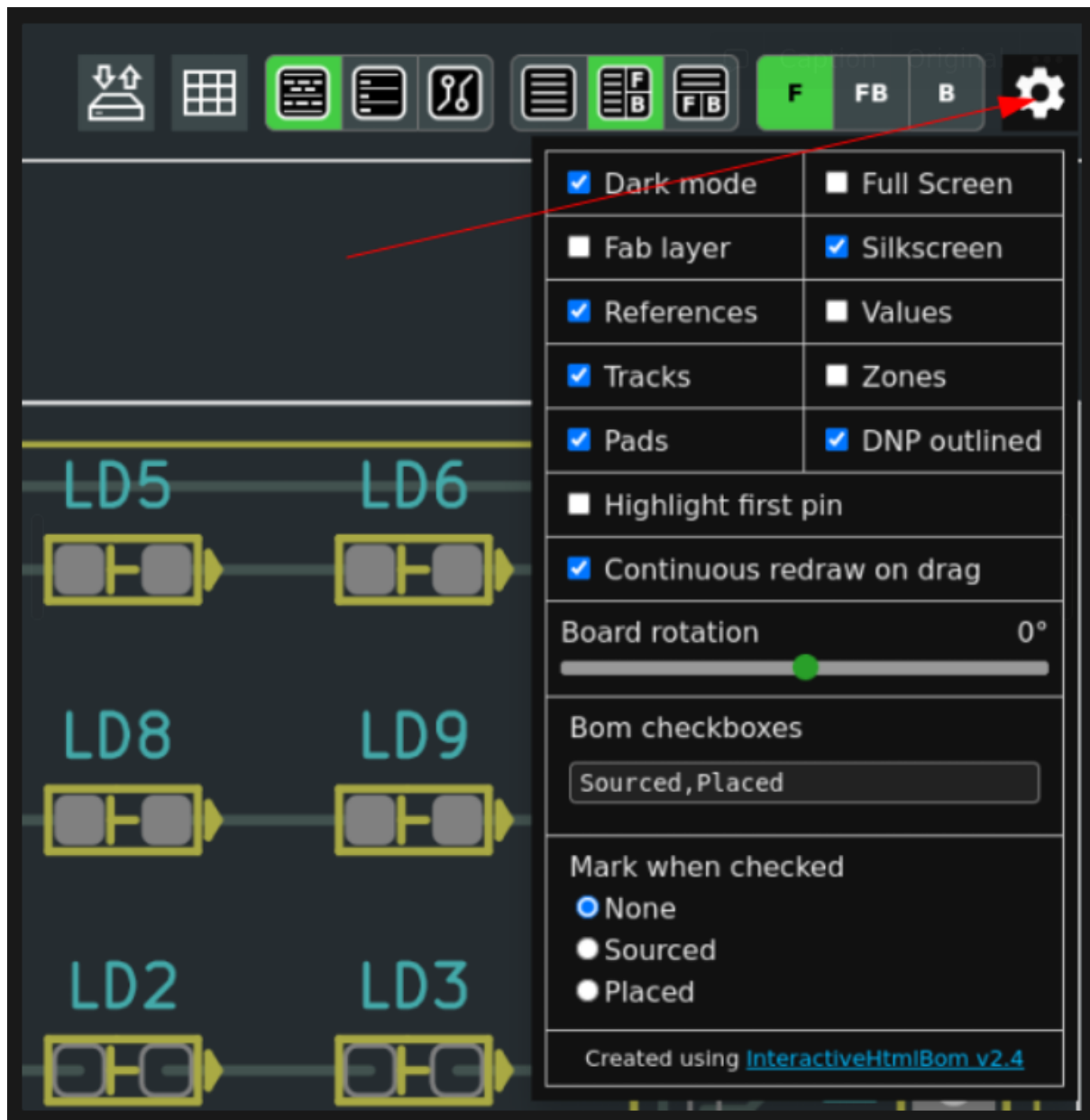
Whitelist    Blacklist

DNP field name

Components with this field not empty will be ignored

DNP1

No html gerado pelo plugin é possível habilitar e desabilitar alguns layers da placa, entre muitas outras opções de visualização.



## Repositório do projeto

[https://github.com/flaviohpo/bom\\_generation\\_kicad](https://github.com/flaviohpo/bom_generation_kicad)