


## 1. Introduction.

This program creates a "Bill of Materials" (BOM) by parsing a gschem schematic file and grouping components that have identical attributes (except for reference designator). Only components that have a reference designator (refdes) are included in the BOM.

 This program was built using a literate programming tool I am working on which is called `pweb`. My tool is modeled after the Knuth tool called `cweb`. Non-catastrophic infelicities should be considered bugs. Sections are numbered, hyperlinks to sections are in red and a list of sections is at the end of the document.

2. Shebang, a couple of my favorite pragmas, the usual suspects and the `Gschem` library. The `Gschem` library is used to parse the schematic and place component attributes in a hash.

```
#!/usr/bin/perl
use strict;
use warnings;
use Carp;
use Data::Dumper;
use Tie::IxHash;
use Gschem_3;
```

3. Setup and initialize the global variables.

There are three global variables — `%Opt`, `@Schs` and `%Cfg`.

`%Opt` contains key-value pairs for each command-line option that is retrieved by `(get command-line options)`. After the command-line options are removed the remaining arguments are treated as schematic filenames to be parsed.

`@Schs` contains the names of the schematics to be parsed. If there were no schematic names on the command-line then `@Schs` will contain the names of all schematic files in the current directory. If `@Schs` is empty then the program will exit with an error message.

`%Cfg` contains the configuration file options. Currently the only option is `key_fields`. The `key_fields` define the component attributes that will be used to group components. A default list is initialized but can be overwritten using the configuration file.

```
my %Opt;
<get command-line options 7>;
my @Schs = @ARGV;
@Schs = <*.sch> if $#Schs == -1;
croak "(bom) no schematics to parse" if $#Schs == -1;
my %Cfg;
tie %Cfg, "Tie::IxHash";
%Cfg = (key_fields => [qw(manufacturer manufacturer_part_number value device footprint)]);
<initialize configuration options 8>;
```

4. Main Program.

After each schematic is read identical components are grouped. To group a component a key is built by concatenating the values for each of the key fields defined in `$Cfg{key_fields}`. For each component with an identical key a refdes is added to `%comp_group`. For each unique key an entry is added to `%first_comp` that contains the key and all of the component attributes.

Entries in `%comp_group` have the format:

$\langle \text{component group key} \rangle \rightarrow \langle \text{refdes} \rangle \rightarrow 1$

Entries in `%first_comp` have the format:

$\langle \text{first refdes in component group} \rangle \rightarrow \langle \text{list with component group key and component attributes} \rangle$

```
foreach my $sch (@Schs) {
    my %comp_group;
    my %first_comp;
    tie %comp_group, "Tie::IxHash";
    tie %first_comp, "Tie::IxHash";
    $sch =~ s/\.sch$//;
    my $bom_filename = "$sch.bom";
    my $gschem = Gschem_3 → new();
    $gschem → read_sch(filename ⇒ "$sch.sch");
    ⟨group components with identical attribute values 6⟩;
    ⟨output bom 5⟩;
}
```

## 5. Output the Bill of Materials.

For each component group the common attributes are output followed by a line for each refdes that is in the component group.

The example below has two component groups. The first group is of capacitors that are 3900pF and have an 0805 footprint. The second group contains capacitors that are made by TDK and have a part number of C3216X7R1H105K.

```
[bom]
value=3900pF
footprint=0805
refdes=C1
refdes=C5

manufacturer=TDK
manufacturer_part_number=C3216X7R1H105K
value=1uF
footprint=1206
refdes=C4
refdes=C8
refdes=C40
refdes=C41
refdes=C44
refdes=C45
```

```
⟨output bom 5⟩≡
printf("(bom) creating $bom_filename\n");
open(OUT, ";$bom_filename") || die "Could not open $bom_filename for output";
print OUT "[bom]\n";
foreach my $refdes (keys %first_comp) {
    my ($key, @attrs) = @ { $first_comp{$refdes} };
    while (@attrs) {
        printf(OUT "%s=%s\n", splice @attrs, 0, 2);
    }
}
```

```

    }
    my @refdes = (keys % { $comp_group{$key} });
    foreach (@refdes) {
        print OUT "refdes=$_\n";
    }
    print OUT "\n";
}
close(OUT) || die "Could not close $bom_filename";

```

## 6. Group components.

Components are grouped using a hash-key built with attribute values. The attribute names used to build the hash-key are in the list referenced by `$Cfg{key_fields}`. The hash-key is a concatenation of attribute values separated by colons. If a required attribute does not have a value then an empty string is used in building the key.

For each component a hash-key is built. If the hash-key exists then its refdes is added to the component group otherwise a new group is created.

```

⟨group components with identical attribute values 6⟩≡
foreach my $refdes ($gschem → refdes_list) {
    my $c = $gschem → get_comp_attribs(refdes ⇒ $refdes);
    my @attribs; # names and values
    # Initialize undefined attribute values to an empty string
    # and save attribute names and values in @attribs
    foreach (@ { $Cfg{key_fields} }) {
        $c→{$_} = '' unless defined $c→{$_};
        push @attribs, $_, $c→{$_};
    }
    # Build the key
    my $key = join(':', map { $c→{$_} } @ { $Cfg{key_fields} });
    # If this is the first component with this key then initialize the
    # hash containing the refdes's for this component group. Since
    # the refdes's in this loop are already sorted making this group
    # hash indexed will preserve the sorting.
    # An easy way to retrieve attribute values for a component group
    # is to save the attribute values (including refdes) for the first
    # component in each group. The attribute values and group key for
    # the first component in each group is saved in the %first_comp
    # hash.
    unless (defined $comp_group{$key}) {
        my %h;
        tie %h, "Tie::IxHash";
        $comp_group{$key} = \%h;
        $first_comp{$refdes} = [ $key, @attribs ];
    }
    $comp_group{$key}{$refdes} = 1;
}

```

## 7. There are two command line options for this program — `cfgfile` and `debug`.

The `cfgfile` option is used to specify the name of a file containing configuration options. Currently the only configuration option is the attribute names that will be used to group identical components.

The `debug` option is used to specify the amount of debugging messages to be printed. When `debug` is a positive number debugging messages will be output to `STDOUT`.

```

<get command-line options 7>≡
use Getopt::Long;
GetOptions('cfgfile=s' ⇒ \$Opt{cfgfile},
           'debug=i'   ⇒ \$Opt{debug});
$Opt{cfgfile} = './bom.cfg' unless defined $Opt{cfgfile};
$Opt{debug}   = 1          unless defined $Opt{debug};

```

## 8. Initialize the configuration options.

Lines in the configuration file are in the following format:

```
@<parameter name>@ <parameter value>
```

<parameter value> can be a single value or a list of values with values separated by vertical bars.

```

<initialize configuration options 8>≡
if (-e $Opt{cfgfile}) {
  @ARGV = $Opt{cfgfile};
  while (<>) {
    next unless s/^\s*\@(.*?)\@\s*//; # skip lines without parameter names
    my $param = $1;
    # only accept parameter names that have already been
    # initialized in %Cfg. If the configuration parameter was a
    # scalar then rewrite a scalar value. If the configuration
    # parameter was an array then split the configuration line and
    # create an array.
    next unless defined $Cfg{$param};
    $Cfg{$param} = $_, next unless ref($Cfg{$param});
    $Cfg{$param} = [ split /\s*\|\s*/ ], next if ref($Cfg{$param}) eq 'ARRAY';
  }
}

```

## 9. Style.

Adapted from the Perl Cookbook, First Edition, Recipe 12.4

- Names of functions and local variables are all lowercase.
- The program's persistent variables (either file lexicals or package globals) are capitalized.
- Identifiers with multiple words have each of these separated by an underscore to make it easier to read.
- Constants are all uppercase.
- If the arrow operator (->) is followed by either a method name or a variable containing a method name then there is a space before and after the operator.

## 10. License.

### No-Fee Software License Version 0.2

#### Intent

The intent of this license is to allow for distribution of this software without fee. Usage of this software other than distribution, is unrestricted.

#### License

Permission is granted to make and distribute verbatim copies of this software provided that (1) no fee is charged and (2) the original copyright notice and this license notice are preserved on all copies.

Permission is granted to make and distribute modified versions of this software under the conditions for verbatim copying provided that the entire resulting derived work is distributed under terms of a license identical to this one.

This software is provided by the author "AS IS" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

## 11. Code Sections

- ⟨output bom 5⟩ Used in section 4
- ⟨group components with identical attribute values 6⟩ Used in section 4
- ⟨get command-line options 7⟩ Used in section 3
- ⟨initialize configuration options 8⟩ Used in section 3