CHAPTER 1 CSL Pattern generator common features

TABLE 1.1 Chapter Outline

1.1 CSL Pattern generator Commands Summary

1.1.1 Pattern generator

The pattern generator is a functionality implemented at preprocessor level and it's used to expand user code into multiple lines of code according to a specification syntax.
Example:

```
iob  iob[[0---2]] (.in(x\1\));
```

expands into:

```
iob  iob0 (.in(x0));
iob  iob1 (.in(x1));
iob  iob2 (.in(x2));
```

The pattern generator can modify one line at a time or an entire area of code.

NOTE:this may need an additional syntax to mark the area to be expanded

Example:

```
csl_register reg[[2---4]]{
 reg\1\(){
   set_type(register);
   set_width(\1\);
 }
};
```

expands into:

```
csl_register reg2{
 reg2(){
   set_type(register);
   set_width(2);
 }
};

csl_register reg3{
 reg3(){
```

```
   set_type(register);
   set_width(3);
 }
};

csl_register reg4{
 reg4(){
   set_type(register);
   set_width(4);
 }
};
```

Syntax:
The following are the specifiers for the pattern generator:

```
[[ pattern_specifier ]]
```

DESCRIPTION :

this generates a set of values according to the type of the pattern specifier. The type of the pattern specifier can be:

- range_pattern: This can be a numeric range (e.g. 0---9 generates 0 to 9) or a character range (e.g. a---z generates a to z)
- list_pattern: this can be a selection (e.g. a,m,z will generate a, m and z)

The pattern specifier creates a backreference that can be called later in the code. Thus, the first pattern specifier creates backreference 1, the second creates backreference 2 and so on. the backreferences can be later called using a specific syntax detailed below:

backreference_number
Syntax:

```
\backreference_number\
```

Example:

```
\1\
```

this will insert generated code according to pattern specifier linked to backreference 1 as in the examples:

- with range pattern

```
iob  iob[[0---2]] (.in(x\1\));
```

which expands into:

```
iob  iob0 (.in(x0));
iob  iob1 (.in(x1));
iob  iob2 (.in(x2));
```

- with list_pattern

```
abc_[[a,c,e,g]] = b_\1\;
```

generates

```
abc_a = b_a;
abc_c = b_c;
abc_e = b_e;
abc_g = b_g;
```

backreference_number with increment amount
Syntax:

*backreference_number***i***increment_amount*\

Example:

\1**i**4\

this will insert generated code according to pattern specifier linked to backreference 1 incremented at each iteration with a value of 4

```
abc[[0---3]] = xyz\1i4\;
```

generates

```
abc0 = xyz0;
abc1 = xyz4;
abc2 = xyz8;
abc3 = xyz12;
```

Note: this does not work for ASCII character ranges in pattern specifier

reversed backreference_number
Syntax:

*back_reference_number***r***\

Example:

\1**r**\

counts backwards from the end of the range or list to the beginning of the range or list

- with range pattern

```
abc[[0---3]] = xyz\1r\;
```

generates

```
abc0 = xyz3;
abc1 = xyz2;
abc2 = xyz1;
abc3 = xyz0;
```

- with list pattern

```
abc_[[a,c,e,g]] = b_\1r\;
```

generates

```
abc_a = b_g;
abc_c = b_e
abc_e = b_c;
abc_g = b_a;
```

ignore sequence

Syntax:

(?#*ignore_sequence***)**

Everything between the (?# and) is the ignore sequence. The pattern generator will not affect the ignore sequence, leaving it as it is.

Example:

```
iob  iob[[(a---d($#_user)) , (e---g($#_driver))]][[0---
9]](.in(x\1\\2\));
```

1.2 CSL Pattern generator Commands