2.1 CSL Memory map get methods summary

```
int get_lower_bound();
int get_upper_bound();
numeric_expression memory_map_page_name.get_address_increment()
numeric_expression memory_map_page_name.get_next_address();
int get_data_word_width();
int get_alignment();
enum get_endianess();
enum memory_map_page_name.get_access_rights();
int get_symbol_length();
int get_data_word_width();
string get_sufix();
string get_prefix();
```

2.2 Get methods

```
int get_lower_bound();
```

DESCRIPTION :

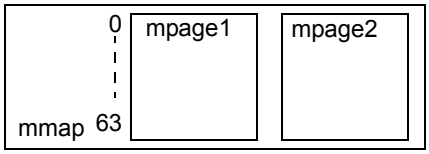
Returns the lower address of a memory range object. Return type can be integer, hex or octal number.

[CSL Memory Map Command Summary]

EXAMPLE :

Create two memory map pages with the same address range. The address range for the second memory map page is set using *get_lower_bound()* and *get_upper_bound()* methods.

FIGURE 2.1 Two memory map pages



CSL CODE

```
csl_memory_map_page mpage1{
  mpage1(){
  add_address_range(0, 63);
 }
};
csl_memory_map_page mpage2{
  mpage1 mpage1;
  mpage2(){

mpage2.add_address_range(mpage1.get_lower_bound(),mpage1.get_upper_bou
nd());
}
};
csl_memory_map mmap{
  mpage1 mpage1;
  mpage2 mpage2;
  mmap(){
  set_data_word_width(32);
}
};
```

VERILOG CODE

```
//AV
```

```
int get_upper_bound();
```

DESCRIPTION :

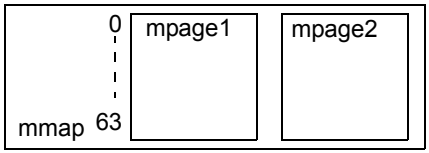
Returns the upper address of a memory range object. Return type can be integer, hex or octal number.

[CSL Memory Map Command Summary]

EXAMPLE :

Create two memory map pages with the same address range. The address range for the second memory map page is set using *get_lower_bound()* and *get_upper_bound()* methods.

FIGURE 2.2 Two memory map pages



CSL CODE

```
csl_memory_map_page mpage1{
  mpage1(){
  add_address_range(0, 63);
  }
};
csl_memory_map_page mpage2{
  mpage1 mpage1;
  mpage2(){
mpage2.add_address_range(mpage1.get_lower_bound(),mpage1.get_upper_bou
nd());
  }
};
csl_memory_map mmap{
  mpage1 mpage1;
  mpage2 mpage2;
  mmap(){
  set_data_word_width(32);
  }
};
```

VERILOG CODE

```
//AV
```

```
int get_data_word_width();
```

DESCRIPTION :

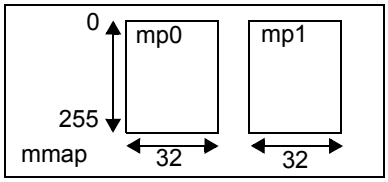
Return the width of the words in memory map page.

[CSL Memory Map Command Summary]

EXAMPLE :

Create a memory map with two memory map pages. The data word width for the second memory map page is set using *get_data_word_method* ().

FIGURE 2.3 A memory map named mem_map with two memory map pages



CSL CODE

```
csl_memory_map_page mpage0{
  mpage0(){
  add_address_range(0,255);
  set_data_word_width(32);
  }
};
csl_memory_map_page mpage1{
  mpage0 mpage0;
  mpage1(){
  add_address_range(0,255);
  set_data_word_width(mpage0.get_data_word_width());
  }
};
csl_memory_map mmap{
  mpage0 mp0;
  mpage1 mp1;
  mmap(){}
};
```

VERILOG CODE

```
//AV
```

```
int get_alignment();
```

DESCRIPTION :

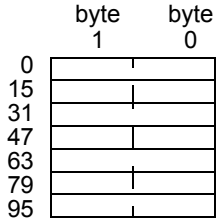
Returns the address alignment setting. The return type is integer.

[CSL Memory Map Command Summary]

EXAMPLE :

Create two memory map pages named *mpage_0* and *mpage_1* and set alignment as byte.

FIGURE 2.4 Two memory maps with byte alignment



CSL CODE

```
csl_memory_map_page mpage_0{
  mpage_0(){
   add_address_range(0,63);
   set_address_increment(2);
   set_alignment(64);
  }
};
csl_memory_map_page mpage_1{
  mpage_0 mpage_0;
  mpage_1(){
  add_address_range(64,512);
  set_address_increment(2);
  set_alignment(mpage_0.get_alignment());
}
};
csl_memory_map mmap{
  mpage_1 mpage_1;
  mmap(){
  set_data_word_width(16);
}
};
```

VERILOG CODE

```
`define <MMN1>_ALIGN 8
`define <MMN2>_ALIGN 8
```

```
enum get_endianess();
```

DESCRIPTION :

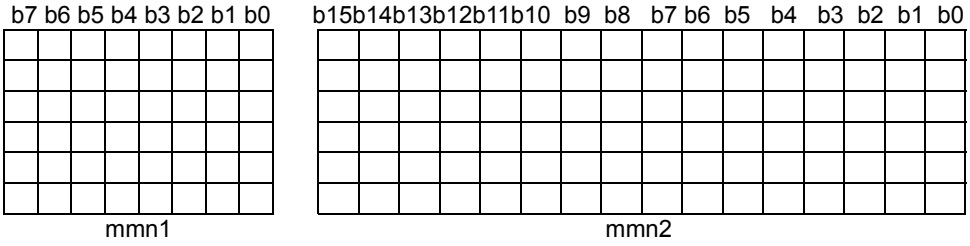
Return the endianess type of a memory map.

[CSL Memory Map Command Summary]

EXAMPLE :

Create two memory map pages and set endianess to big_endian.

FIGURE 2.5 Big Endian



CSL CODE

```
csl_memory_map_page pg1{
  pg1(){
  add_address_range(0, 128);
  set_data_word_width(32);
  set_endianess(big_endian);
  }
};
csl_memory_map_page pg2{
  pg1 pg1;
  pg2(){
  add_address_range(0, 128);
  set_data_word_width(32);
  set_endianess(pg1.get_endianess());
  }
};
csl_memory_map mmn1{
  pg2 pg2;
  mmn1(){}
};
```

VERILOG CODE

```
//AV
`define MMN1_ENDIANESS ENDIAN_BIG
`define MMN1__ALIGN 8
`define MMN2_ENDIANESS ENDIAN_BIG
`define MMN2__ALIGN 16
```

```
int get_symbol_length();
```

DESCRIPTION :

Returns the number which specifies the number of characters in the name.

[CSL Memory Map Command Summary]

EXAMPLE :

Create two memory map pages and set the maximum number of characters for each word to 8

CSL CODE

```
csl_memory_map_page mpage_0 {
  mpage_0(){
  add_address_range(0, 63);
  set_symbol_max_length(8);
  }
};
csl_memory_map_page mpage_1 {
  mpage_0 mpage_0;
  mpage_1(){
  add_address_range(64, 512);
  set_symbol_max_length(mpage_0.get_symbol_length());
  }
};
csl_memory_map mmap{
  mpage_1 mpage_1;
  mmap(){}
};
```

VERILOG CODE

```
`define MEM_MAP1_MAX_LENGTH 8;
`define MEM_MAP2_MAX_LENGTH 8;
```

```
int get_data_word_width();
```

DESCRIPTION :

This command returns the data word width of a memory map or memory map page. Return type can be integer, hex or octal number.

[CSL Memory Map Command Summary]

EXAMPLE :

Sets data word width for a memory map named *mmap*, using the *get_data_word_width()* method.

CSL CODE

```
csl_memory_map_page mpage0{
  mpage0(){
  add_address_range(0,255);
  set_data_word_width(32);
  }
};
csl_memory_map mmap{
  mpage0 mp0;
  mmap(){
  set_data_word_width(mp0.get_data_word_width());
  }
};
```

VERILOG CODE

```
//
```

```
string get_prefix();
```

DESCRIPTION :

Returns the current prefix set to a memory map.

[CSL Memory Map Command Summary]

EXAMPLE :

Sets the prefix “mem” to a memory map *mmap*, using *get_prefix()* method.

CSL CODE

```
csl_memory_map_page mpage_0{
  mpage_0{
  set_prefix("mem");
  }
};
csl_memory_map mmap{
  mpage_0 mpage_0;
  mmap{
  set_prefix(mpage_0.get_prefix());
  }
};
```

VERILOG CODE

```
`define <MMN>_PREFIX name;
```

```
string get_sufix();
```

DESCRIPTION :

Returns the current sufix set to a memory map.

[CSL Memory Map Command Summary]

EXAMPLE :

Sets the sufix “mem” to a memory map *mmap*, using *get_sufix()* method.

CSL CODE

```
csl_memory_map mpage_0{
  mmap_0{
  set_sufix("mem");
  }
};
csl_memory_map mmap{
  mpage_0 mpage_0;
  mmap{
  set_sufix(mpage_0.get_sufix());
  }
};
```

VERILOG CODE

```
`define <MMN>_SUFIX name;
```