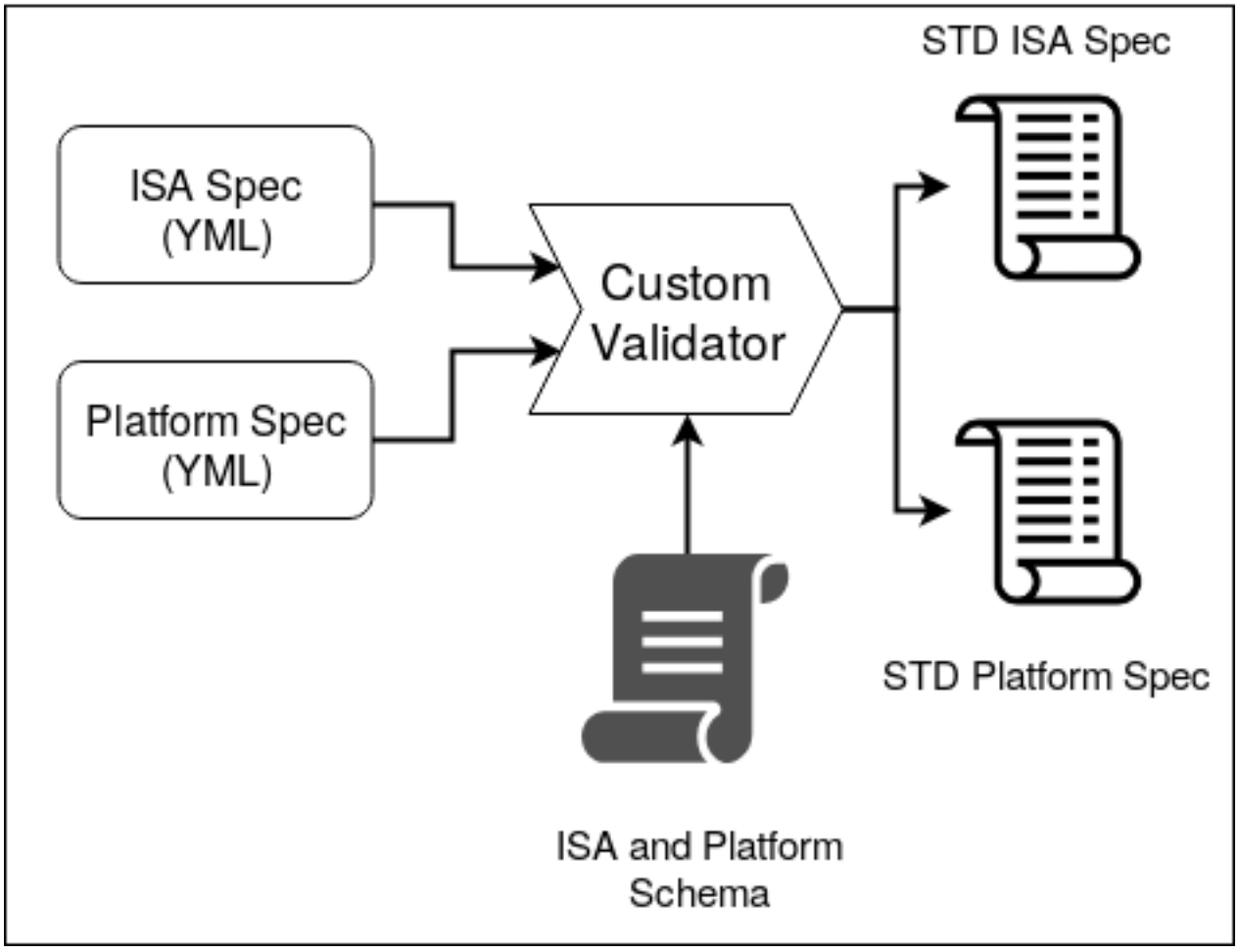
STD ISA Spec
ISA Spec
(YML)
Platform Spec
(YML)
Custom
Validator
STD Platform Spec
ISA and Platform
Schema

```
$ sudo apt-get install python3.6
$ pip3 install --upgrade pip
```

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt-get install python3.6 -y
$ pip3 install --upgrade pip
```

python3pip3

```
$ python3 --version
Python 3.6.9
$ pip3 --version
pip 20.1 from <user-path>.local/lib/python3.6/site-packages/pip (python 3.6)
```

```
$ sudo yum update -y
$ sudo yum install -y python3
$ pip3 install --upgrade pip
```

```
$ sudo yum update -y
$ sudo yum install yum-utils
$ sudo yum install https://centos7.iuscommunity.org/ius-release.rpm
$ sudo yum install python36u
$ pip3 install --upgrade pip
```

```
$ python3 --version
Python 3.6.8
$ pip --version
pip 20.1 from <user-path>.local/lib/python3.6/site-packages/pip (python 3.6)
```

```
$ pyenv install 3.6.0
$ pip3 install --upgrade pip
$ pyenv shell 3.6.0
```

```
$ python --version
Python 3.6.0
$ pip --version
pip 20.1 from <user-path>.local/lib/python3.6/site-packages/pip (python 3.6)
```

```
$ pip3 install riscv_config
```

```
$ pip3 install -U riscv_config
```

```
$ pip3 install riscv_config--1.x.x
```

riscv_config --help

```
riscv_config [-h] [--version] [--isa_spec YAML] [--platform_spec YAML]
                    [--work_dir DIR] [--verbose]

RISC-V Configuration Validator

optional arguments:
  --isa_spec YAML, -ispec YAML
                        The YAML which contains the ISA specs.
  --platform_spec YAML, -pspec YAML
                        The YAML which contains the Platfrorm specs.
  --verbose             debug | info | warning | error
  --version, -v         Print version of RISCV-CONFIG being used
  --work_dir DIR        The name of the work dir to dump the output files to.
  -h, --help            show this help message and exit
```

```
$ git clone https://github.com/riscv/riscv-config.git
$ cd riscv_config
$ pip3 install -r requirements.txt
```

```
python -m riscv_config.main --help
```

```
$ riscv-config -ispec examples/rv32i_isa.yaml -pspec examples/rv32i_platform.yaml
```

```
[INFO]    : Input-ISA file
[INFO]    : Loading input file: /scratch/git-repo/github/riscv-config/examples/rv32i_isa.
↪yaml
[INFO]    : Load Schema /scratch/git-repo/github/riscv-config/riscv_config/schemas/
↪schema_isa.yaml
[INFO]    : Initiating Validation
[INFO]    : No Syntax errors in Input ISA Yaml. :)
[INFO]    : Initiating post processing and reset value checks.
[INFO]    : Dumping out Normalized Checked YAML: /scratch/git-repo/github/riscv-config/
↪riscv_config_work/rv32i_isa_checked.yaml
[INFO]    : Input-Platform file
[INFO]    : Loading input file: /scratch/git-repo/github/riscv-config/examples/rv32i_
↪platform.yaml
[INFO]    : Load Schema /scratch/git-repo/github/riscv-config/riscv_config/schemas/
↪schema_platform.yaml
[INFO]    : Initiating Validation
[INFO]    : No Syntax errors in Input Platform Yaml. :)
[INFO]    : Dumping out Normalized Checked YAML: /scratch/git-repo/github/riscv-config/
↪riscv_config_work/rv32i_platform_checked.yaml
```

```
Vendor: Shakti
Vendor: Incoresemi
```

```
Device: E-Class
Device: C-Class
```

```
ISA: RV32IMA
ISA: RV64IMAFDCZifencei
```

```
User_Spec_Version: "2.2"
User_Spec_Version: "2.3"
```

```
Privilege_Spec_Version: "1.10"
Privilege_Spec_Version: "1.11"
```

```
hw_data_misaligned_support: True
hw_data_misaligned_support: False
```

```
supported_xlen : [32]
supported_xlen : [64, 32]
supported_xlen : [64]
```

```
pmp_granularity : 2
pmp_granularity : 4
```

```
physical_addr_sz : 32
```

```
custom_exceptions:
  - cause_val: 25
    cause_name: mycustom
    priv_mode: M
  - cause_val: 26
    cause_name: mycustom2
    priv_mode: M
```

```
custom_interrupts:
  - cause_val: 25
    cause_name: mycustom
    priv_mode: M
    on_reset_enable: 1
  - cause_val: 26
    cause_name: mycustom2
    priv_mode: S
    on_reset_enable: 0
```

```
pte_ad_hw_update: False
pte_ad_hw_update: True
```

```
mtval_update: 0b1110
```

```
<name>:                                 # name of the csr
  description: <text>                   # textual description of the csr
  address: <hex>                        # address of the csr
  priv_mode: <D/M/H/S/U>                # privilege mode that owns the register
  reset-val: <hex>                      # Reset value of the register. This an␣
↪accumulation
                                        # of the all reset values of the sub-fields
  rv32:                                 # this node and its subsequent fields can exist
                                        # if [M/S/U]XL value can be 1
    accessible: <boolean>               # indicates if the csr is accessible in rv32␣
↪mode or not.
                                        # When False, all fields below will be trimmed␣
↪off
                                        # in the checked yaml. False also indicates␣
↪that
                                        # access-exception should be generated.
    fields:                             # a quick summary of the list of all fields of␣
↪the
                                        # csr including a list of WPRI fields of the␣
↪csr.
      - <field_name1>
      - <field_name2>
      - - [23,30]                       # A list which contains a squashed pair
        - 6                             # (of form [lsb,msb]) of all WPRI bits within␣
↪the
```

```
                                        # csr. Does not exist if there are no WPRI bits

    <field_name1>:                      # name of the field
      description: <text>               # textual description of the csr
      shadow: <csr-name>::<field>       # which this field shadows,'none' indicates␣
↪that
                                        # this field does not shadow anything.
      msb: <integer>                    # msb index of the field. max: 31, min:0
      lsb: <integer>                    # lsb index of the field. max: 31, min:0
      implemented: <boolean>            # indicates if the user has implemented this␣
↪field
                                        # or not. When False, all
                                        # fields below this will be trimmed.
      type:                             # type of field. Can be only one of the␣
↪following
        wlrl: [list of value-descriptors] # field is wlrl and the set of legal values.
        ro_constant: <hex>              # field is readonly and will return the same␣
↪value.
        ro_variable: True               # field is readonly but the value returned␣
↪depends
                                        # on other arch-states
        warl:                           # field is warl type. Refer to WARL section
          dependency_fields: [list]
          legal: [list of warl-string]
          wr_illegal: [list of warl-string]
  rv64:                                 # this node and its subsequent fields can exist
                                        # if [M/S/U]XL value can be 2
    accessible: <boolean>               # indicates if this register exists in rv64␣
↪mode
                                        # or not. Same definition as for rv32 node.
  rv128:                                # this node and its subsequent fields can␣
↪exist if
                                        # [M/S/U]XL value can be 3
    accessible: <boolean>               # indicates if this register exists in rv128␣
↪mode
                                        # or not. Same definition as for rv32 node.
```

```
<name>:                                 # name of the csr
  description: <text>                   # textual description of the csr
  address: <hex>                        # address of the csr
  priv_mode: <D/M/H/S/U>                # privilege mode that owns the register
  reset-val: <hex>                      # Reset value of the register. This an␣
↪accumulation
```

```
                                        # of the all reset values of the sub-fields
  rv32:                                 # this node and its subsequent fields can exist
                                        # if [M/S/U]XL value can be 1
    accessible: <boolean>               # indicates if the csr is accessible in rv32␣
↪mode or not.
                                        # When False, all fields below will be trimmed␣
↪off
                                        # in the checked yaml. False also indicates that
                                        # access-exception should be generated
    fields: []                          # This should be empty always.
    shadow: <csr-name>::<register>      # which this register shadows,'none' indicates␣
↪that
                                        # this register does not shadow anything.
    msb: <int>                          # msb index of the csr. max: 31, min:0
    lsb: <int>                          # lsb index of the csr. max: 31, min:0
    type:                               # type of field. Can be only one of the following
      wlrl: [list of value-descriptors] # field is wlrl and the set of legal values.
      ro_constant: <hex>                # field is readonly and will return the same␣
↪value.
      ro_variable: True                 # field is readonly but the value returned␣
↪depends
                                        # on other arch-states
      warl:                             # field is warl type. Refer to WARL section
        dependency_fields: [list]
        legal: [list of warl-string]
        wr_illegal: [list of warl-string]
  rv64:                                 # this node and its subsequent fields can exist
                                        # if [M/S/U]XL value can be 2
    accessible: <boolean>               # indicates if this register exists in rv64 mode
                                        # or not. Same definition as for rv32 node.
  rv128:                                # this node and its subsequent fields can exist␣
↪if
                                        # [M/S/U]XL value can be 3
    accessible: <boolean>              # indicates if this register exists in rv128 mode
```

```
mtvec:
reset-val: 0x80010000
rv32:
  accessible: true
  base:
    implemented: true
    type:
      warl:
        dependency_fields: [mtvec::mode]
        legal:
          - "mode[1:0] in [0] -> base[29:0] in [0x20000000, 0x20004000]"             #␣
↪can take only 2 fixed values in direct mode.
          - "mode[1:0] in [1] -> base[29:6] in [0x000000:0xF00000] base[5:0] in [0x00]"
↪# 256 byte aligned values only in vectored mode.
        wr_illegal:
          - "mode[1:0] in [0] -> Unchanged"
          - "mode[1:0] in [1] && writeval in [0x2000000:0x4000000] -> 0x2000000"
          - "mode[1:0] in [1] && writeval in [0x4000001:0x3FFFFFFF] -> Unchanged"
  mode:
    implemented: true
    type:
      warl:
        dependency_fields: []
        legal:
          - "mode[1:0] in [0x0:0x1] # Range of 0 to 1 (inclusive)"
        wr_illegal:
          - "Unchanged"
```

```yaml
mtvec:
  description: MXLEN-bit read/write register that holds trap vector configuration.
  address: 773
  priv_mode: M
  reset-val: 0x80010000
  rv32:
    accessible: true
    base:
      implemented: true
      type:
        warl:
          dependency_fields: [mtvec::mode, writeval]
          legal:
          - 'mode[1:0] in [0] -> base[29:0] in [0x20000000, 0x20004000]'               # can take only 2 fixed values in direct mode.
          - 'mode[1:0] in [1] -> base[29:6] in [0x000000:0xF00000] base[5:0] in [0x00]' # 256 byte aligned values only in vectored mode.
          wr_illegal:
          - 'mode[1:0] in [0] -> Unchanged'
          - 'mode[1:0] in [1] && writeval in [0x2000000:0x4000000] -> 0x2000000'
          - 'mode[1:0] in [1] && writeval in [0x4000001:0x3FFFFFFF] -> Unchanged'
      description: Vector base address.
      shadow: none
      msb: 31
      lsb: 2
    mode:
      implemented: true
      type:
        warl:
          dependency_fields: []
          legal:
          - 'mode[1:0] in [0x0:0x1] # Range of 0 to 1 (inclusive)'
          wr_illegal:
          - Unchanged

      description: Vector mode.
      shadow: none
      msb: 1
      lsb: 0
    fields:
    - mode
    - base
  rv64:
    accessible: false
```

```
val
```

```
lower:upper
```

```
# To represent the set {0, 1, 2, 3, 4, 5}
  [0:5]

# To represent the set {5, 10, 31}
  [5, 10, 31]

# To represent the set {2, 3, 4, 5, 10, 11, 12, 13, 50}
  [2:5, 10:13, 50]
```

```
warl:
   dependency_fields: [list of csrs/subfields that legal values depend on]
   legal: [list of strings adhering to the warl-syntax for legal assignments]
   wr_illegal: [list of strings ahdering to the warl-syntax for illegal assignments]
```

::

```
- dependency_fields: [mtvec::mode]
- dependency_fields: [misa::mxl, mepc]
```

writeval
currval

```
dependency_string -> legal_value_string
```

dependency_stringdependency_fieldslegal_value_stringlegal_value_string->
dependency_string ->dependency_fieldsdependency_string ->
dependency_stringlegal_value_string

```
<variable-name>[<hi-index>:<lo-index>] <op> <value-descriptors>
```

variable-name::dependency_stringdependency_fieldslegal_value_stringvariable-name
::
hi-indexlo-indexhi-indexlo-index:lo-indexdependency_stringlegal_value_string
opdependency_stringinnot invalue-descriptorslegal_value_stringbitmaskbitmask

```
csr_name[hi:lo] bitmask [mask, fixedval]
```

maskfixedval
dependency_string&&||

```
(csrA[2:0] in [0, 1]) && (csrB[5:0] in [0:25] || csrB[5:0] in [31,30]) ->
```

dependency_stringdependency_fields
dependency_stringinnot in
legal_value_stringinnot inbitmask
legal_value_string
dependency_fields
dependency_stringlegal_value_string

dependency_stringdependency_stringswr_illegal
dependency_fieldsdependency_strings

```
dependency_string -> update_mode
```

dependency_stringdependency_stringupdate_modeupdate_mode

```
if ( val < base || val > bound)
    return Flip-MSB of field
```

```
# When base of mtvec depends on the mode field.
WARL:
  dependency_fields: [mtvec::mode]
  legal:
    - "mode[1:0] in [0] -> base[29:0] in [0x20000000, 0x20004000]"  # can take only 2␣
↪fixed values when mode==0.
    - "mode[1:0] in [1] -> base[29:6] in [0x000000:0xF00000] base[5:0] in [0x00]" # 256␣
↪byte aligned when mode==1
  wr_illegal:
    - "mode[1:0] in [0] -> unchanged"
    - "mode[1:0] in [1] && writeval in [0x2000000:0x4000000] -> 0x2000000" # predefined␣
↪value if write value is
    - "mode[1:0] in [1] && writeval in [0x4000001:0x3FFFFFFF] -> unchanged"

# When base of mtvec depends on the mode field. Using bitmask instead of range
WARL:
  dependency_fields: [mtvec::mode]
  legal:
    - "mode[1:0] in [0] -> base[29:0] in [0x20000000, 0x20004000]"  # can take only 2␣
↪fixed values when mode==0.
    - "mode[1:0] in [1] -> base[29:0] bitmask [0x3FFFFFC0, 0x00000000]" # 256 byte␣
↪aligned when mode==1
  wr_illegal:
    - "mode[1:0] in [0] -> unchanged" # no illegal for bitmask defined legal strings.
    - Unchanged


# no dependencies. Mode field of mtvec can take only 2 legal values using range-
↪descriptor
WARL:
  dependency_fields:
  legal:
    - "mode[1:0] in [0x0:0x1] # Range of 0 to 1 (inclusive)"
  wr_illegal:
    - "0x00"

# no dependencies. using single-value-descriptors
WARL:
  dependency_fields:
```

```
  legal:
    - "mode[1:0] in [0x0,0x1] # Range of 0 to 1 (inclusive)"
  wr_illegal:
    - "0x00"
```

```
reset:
  label: reset_vector
reset:
  label: 0x80000000
```

```
nmi:
   label: nmi_vector

nmi:
   address: 0x8000000
```

```
mtime:
   implemented: True
   address: 0x458
```

```
mtimecmp:
   implemented: True
   address: 0x458
```

TBD: Provide a concrete use-case for the above.

```
scause_non_standard:
   implemented: True
   value: [16,17,20]
```

TBD: Provide a concrete use-case for the above.

```
zicbo_cache_block_sz :
  implemented: true
  zicbom_sz: 64
  zicboz_sz: 64
```

```
supported_xlen : [32]
supported_xlen : [64, 32]
supported_xlen : [64]
```

```
Debug_Spec_Version: "1.0.0"
Debug_Spec_Version: "0.13.2"
```

```
debug_mode: False
```

```
parking_loop: 0x800
```

```
Z_extensions = [
        "Zicbom", "Zicbop", "Zicboz", "Zicsr", "Zifencei", "Zihintpause",
        "Zam",
        "Zfh",
        "Zfinx", "Zdinx", "Zhinx", "Zhinxmin",
        "Ztso",
        "Zba", "Zbb", "Zbc", "Zbe", "Zbf", "Zbkb", "Zbkc", "Zbkx", "Zbm", "Zbp", "Zbr",
↪"Zbs", "Zbt",
        "Zk", "Zkn", "Zknd", "Zkne", "Zknh", "Zkr", "Zks", "Zksed", "Zksh", "Zkt",
        "Zmmul",
        "Svnapot"
]
isa_regex = \
        re.compile("^RV(32|64|128)[IE][ACDFGHJLMNPQSTUVX]*(("+'|'.join(Z_extensions)+")(_
↪("+'|'.join(Z_extensions)+"))*){,1}$")
```

extension_list
err
err_list

errextension_list

errerr_list

extension_list

```
(...)
if 'Zkn' in extension_list and ( set(['Zbkb', 'Zbkc', 'Zbkx', 'Zkne', 'Zknd', 'Zknh']) &␣
↪set(extension_list)):
  err_list.append( "Zkn is a superset of Zbkb, Zbkc, Zbkx, Zkne, Zknd, Zknh. In presence␣
↪of Zkn the subsets must be ignored in the ISA string.")
  err = True
if 'Zks' in extension_list and ( set(['Zbkb', 'Zbkc', 'Zbkx', 'Zksed', 'Zksh']) &␣
↪set(extension_list) ):
  err_list.append( "Zks is a superset of Zbkb, Zbkc, Zbkx, Zksed, Zksh. In presence of␣
↪Zks the subsets must be ignored in the ISA string.")
  err = True
```

(extension_list, err, err_list)extension_listerr

check_withcheck_with

_check_withcheck_with

```
stval:
   type: dict
   schema:
     description:
       type: string
       default: The stval is a warl register that holds the address of the instruction
         which caused the exception.
     address: {type: integer, default: 0x143, allowed: [0x143]}
     priv_mode: {type: string, default: S, allowed: [S]}
     reset-val:
       type: integer
       default: 0
       check_with: max_length
     rv32:
       type: dict
       check_with: s_check
       schema:
         fields: {type: list, default: []}
         shadow: {type: string, default: , nullable: True}
         msb: {type: integer, default: 31, allowed: [31]}
         lsb: {type: integer, default: 0, allowed: [0]}
         type:
           type: dict
           schema: { warl: *ref_warl }
           default:
             warl:
               dependency_fields: []
               legal:
               - stval[31:0] in [0x00000000:0xFFFFFFFF]
               wr_illegal:
               - unchanged
```

```
        accessible:
          type: boolean
          default: true
          check_with: rv32_check
      default: {accessible: false}
    rv64:
      type: dict
      check_with: s_check
      schema:
        fields: {type: list, default: []}
        shadow: {type: string, default: , nullable: True}
        msb: {type: integer, default: 63, allowed: [63]}
        lsb: {type: integer, default: 0, allowed: [0]}
        type:
          type: dict
          schema: { warl: *ref_warl }
          default:
            warl:
              dependency_fields: []
              legal:
              - stval[63:0] in [0x00000000:0xFFFFFFFFFFFFFFFF]
              wr_illegal:
              - unchanged

        accessible:
          default: true
          check_with: rv64_check
      default: {accessible: false}
```

```
def _check_with_rv32_check(self, field, value):
  global xlen
  if value:
      if not rv32:
          self._error( field, "Register cannot be implemented in rv32 mode due to␣
↪unsupported xlen.")

def _check_with_rv64_check(self, field, value):
  global xlen
  if value:
      if not rv64:
          self._error( field, "Register cannot be implemented in rv64 mode due to␣
↪unsupported xlen.")

def _check_with_max_length(self, field, value):
  '''Function to check whether the given value is less than the maximum value that can␣
↪be stored(2^xlen-1).'''
```

```
  global supported_xlen
  global extensions
  maxv = max(supported_xlen)
  if value > (2**maxv) - 1:
      self._error(field, "Value exceeds max supported length")

def _check_with_s_check(self, field, value):
    s = 18
    check = False
    if 'implemented' in value:
        if value['implemented']:
            check = True
    if 'accessible' in value:
        if value['accessible']:
            check = True

    if rv64 and check:
        mxl = format(extensions, '#066b')
        if (mxl[65 - s:66 - s] != '1'):
            self._error(field, "should not be implemented since S is not present")

    elif rv32 and check:
        mxl = format(extensions, '#034b')
        if (mxl[33 - s:34 - s] != '1'):
            self._error(field, "should not be implemented S is not present")
```

```
schema_yaml['stval']['default_setter'] = sregsetter
```

```
def sregset():
 '''Function to set defaults based on presence of 'S' extension.'''
 global inp_yaml
 temp = {'rv32': {'accessible': False}, 'rv64': {'accessible': False}}
 if 'S' in inp_yaml['ISA']:
   if 32 in inp_yaml['supported_xlen']:
     temp['rv32']['accessible'] = True
   if 64 in inp_yaml['supported_xlen']:
     temp['rv64']['accessible'] = True
 return temp
```

```
parser.add_argument('--debug_spec', '-dspec', type=str, metavar='YAML', default=None,␣
↪help='The YAML which contains the debug csr specs.')
```

```
debug_schema = os.path.join(root, 'schemas/schema_debug.yaml')
```

```
if args.debug_spec is not None:
    if args.isa_spec is None:
     logger.error(' Isa spec missing, Compulsory for debug')
    checker.check_debug_specs(os.path.abspath(args.debug_spec), isa_file, work_dir, True,
↪ args.no_anchors)
```

riscv_config.checker.**add_debug_setters**()

riscv_config.checker.**add_def_setters**()

riscv_config.checker.**add_reset_setters**()

riscv_config.checker.**check_custom_specs**()
    Cerberus

        isa_spec*str*
        logging*bool*
    ValidationError

riscv_config.checker.**check_debug_specs**()
    Cerberus

        debug_spec
        isa_spec*str*
        logging*bool*
    ValidationError

riscv_config.checker.**check_isa_specs**()
    Cerberus

        isa_spec*str*
        logging*bool*

ValidationError

riscv_config.checker.**check_mhpm**()

riscv_config.checker.**check_pmp**()

riscv_config.checker.**check_reset_fill_fields**()

riscv_config.checker.**check_shadows**()

riscv_config.checker.**check_supervisor**()

riscv_config.checker.**delegset**()

riscv_config.checker.**fsset**()

riscv_config.checker.**groupc**()

riscv_config.checker.**hregset**()

riscv_config.checker.**hregseth**()

riscv_config.checker.**hset**()

riscv_config.checker.**nregset**()

riscv_config.checker.**nuset**()

riscv_config.checker.**reset**()

riscv_config.checker.**reset_vsstatus**()

riscv_config.checker.**resetsu**()

riscv_config.checker.**sregset**()

riscv_config.checker.**sregseth**()

riscv_config.checker.**sset**()

riscv_config.checker.**trim**()

foo*dict*

riscv_config.checker.**twset**()

riscv_config.checker.**uregset**()

riscv_config.checker.**uregseth**()

riscv_config.checker.**uset**()

class riscv_config.schemaValidator.**schemaValidator**()

__init__()

_check_with_cannot_be_false_rv32()

_check_with_cannot_be_false_rv64()

_check_with_capture_isa_specifics()

_check_with_max_length()

_check_with_max_length32()

_check_with_mtval_update()

_check_with_s_debug_check()

_check_with_s_exists()

_check_with_u_debug_check()

_check_with_xcause_check()

_check_with_xtveccheck()

class riscv_config.utils.**ColoredFormatter**()

__init__()

str.format(){}string.Template

style

format()

class riscv_config.utils.**SortingHelpFormatter**()

riscv_config.utils.**setup_logging**()

log_level*str*

class riscv_config.warl.**warl_class**()

```
warl:
   dependency_fields: [list]
   legal: [list of warl-string]
   wr_illegal: [list of warl-string]
```

node*dict*
csrname*str*::
f_msb*int*
f_lsb*int*
spec*dict*

__init__()

__weakref__

check_subval_legal()

legalstr*str*
value*int*

\[(.*?)\]\s*(bitmask|in|not in)\s*\[(.*?)\]

bitmask

in

not in

getlegal()

dependency_vals*dict*

islegal()

value*int*
dependency_vals*dict*

check_subval_legal()

check_with

_Z

`--version`

```
cerberus.validator.DocumentError: document is missing
```

r

riscv_config.checker
riscv_config.schemaValidator
riscv_config.utils
riscv_config.warl

Symbols

A

C

D

F

G

H

I

M

N

R

S

T

U

W