How to use Executable Consensus Pyspec

Hsiao-Wei Wang
Ethereum Foundation - Consensus R&D
1. It’s a collection of Ethereum core **consensus specifications**
   Define the consensus protocol that running by the consensus layer (CL) clients

2. It’s **executable** and **verifiable**
   It can be built into a Python program that can be executed

3. It’s **test vector generator**
   It can generate the test vectors for CL clients to run with and test against the consensus rules.
Adding New Feature Patch

1. Implement new features in Pyspec markdown files

2. Release new Pyspec with test vector suite

3. CL clients implement and test against test vectors

GitHub: @ethereum/consensus-spec-tests
Python is very readable to developers

Credits: r/ProgrammerHumor u/Cant_Grow_a_Stasch “Python == pseudocode”
How to read it?

/specs/

- altair
- bellatrix
- capella
- custody_game
- das
- eip4844
- phase0
- sharding

Mainnet protocol upgrades
WIP features
How to read it?

/specs/

- altair
- bellatrix
- phase0
  - beacon-chain.md
  - deposit-contract.md
  - fork-choice.md
  - p2p-interface.md
  - validator.md
  - weak-subjectivity.md
Type and Values Definitions

**Notation**

Code snippets appearing in **this style** are to be interpreted as Python 3 code.

**Custom types**

We define the following Python custom types for type hinting and readability:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSZ equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot</td>
<td>uint64</td>
<td>a slot number</td>
</tr>
<tr>
<td>Epoch</td>
<td>uint64</td>
<td>an epoch number</td>
</tr>
<tr>
<td>CommitteeIndex</td>
<td>uint64</td>
<td>a committee index at a slot</td>
</tr>
<tr>
<td>ValidatorIndex</td>
<td>uint64</td>
<td>a validator registry index</td>
</tr>
</tbody>
</table>

**Preset**

Note: The below configuration is bundled as a preset; a bundle of configuration variables which are expected to differ between different modes of operation, e.g. testing, but not generally between different networks. Additional preset configurations can be found in the `configs` directory.

**Misc**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_COMMITTEES_PER_SLOT</td>
<td>uint64(2**6) (= 64)</td>
</tr>
<tr>
<td>TARGET_COMMITTEE_SIZE</td>
<td>uint64(2**7) (= 128)</td>
</tr>
<tr>
<td>MAX_VALIDATORS_PER_COMMITTEE</td>
<td>uint64(2**11) (= 2,048)</td>
</tr>
<tr>
<td>SHUFFLE_ROUND_COUNT</td>
<td>uint64(100)</td>
</tr>
</tbody>
</table>

**Configuration**

Note: The default mainnet configuration values are included here for illustrative purposes. Defaults for this more dynamic type of configuration are available with the presets in the `configs` directory. Testnets and other types of chain instances may use a different configuration.

**Genesis settings**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN_GENESIS_ACTIVE_VALIDATOR_COUNT</td>
<td>uint64(2**14) (= 16,384)</td>
</tr>
<tr>
<td>MIN_GENESIS_TIME</td>
<td>uint64(1686524000) (Dec 1, 2020, 12pm UTC)</td>
</tr>
</tbody>
</table>
SSZ Containers

Note: we also use SSZ **hash tree root** as the digests of consensus objects.

See:

State transition function

The post-state corresponding to a pre-state `state` and a signed block `signed_block` is defined as `state_transition(state, signed_block)`. State transitions that trigger an unhandled exception (e.g. a failed `assert` or an out-of-range list access) are considered invalid. State transitions that cause a `uint64` overflow or underflow are also considered invalid.

```python
def state_transition(state: BeaconState, signed_block: SignedBeaconBlock, validate_result: bool=True) -> None:
    block = signed_block.message
    # Process slots (including those with no blocks) since block
    process_slots(state, block.slot)
    # Verify signature
    if validate_result:
        assert verify_block_signature(state, signed_block)
    # Process block
    process_block(state, block)
    # Verify state root
    if validate_result:
        assert block.state_root == hash_tree_root(state)
```

post_state = state_transition(pre_state, block)
Useful resources to understand CL

- Vitalik Buterin’s Annotated spec: https://github.com/ethereum/annotated-spec
- Ben Edgington’s Upgrading Ethereum Book: https://eth2book.info
def _build_spec(preset_name: str, fork: str,  
    source_files: Sequence[Path], preset_files: Sequence[Path], config_file: Path) -> str:
    preset = load_preset(preset_files)
    config = load_config(config_file)
    all_specs = [get_spec(spec, preset, config) for spec in source_files]

    spec_object = all_specs[0]
    for value in all_specs[1:]:
        spec_object = combine_spec_objects(spec_object, value)

    class_objects = {**spec_object.ssz_objects, **spec_object.dataclasses}

    # Ensure it's ordered after multiple forks
    new_objects = {}
    while OrderedDict(new_objects) != OrderedDict(class_objects):
        new_objects = copy.deepcopy(class_objects)
        dependency_order_class_objects(class_objects, spec_object.custom_types)

    return objects_to_spec(preset_name, spec_object, spec_builders[fork], class_objects)
Extend the previous hard forks

Phase 0 beacon chain spec

[SpecBuilder]
transform markdown files to .py files

bellatrix/beacon-chain.md
altair/beacon-chain.md
phase0/beacon-chain.md
Extend the previous hard forks

Altair beacon chain spec

[SpecBuilder]
transform markdown files to .py files

bellatrix/beacon-chain.md
altair/beacon-chain.md
phase0/beacon-chain.md
Extend the previous hard forks

Bellatrix beacon chain spec

[SpecBuilder] transform markdown files to .py files

- bellatrix/beacon-chain.md
- altair/beacon-chain.md
- phase0/beacon-chain.md
How to use Pyspec?
Installation (Python3.8+)

1. Install from PyPI:
   
   ```
   pip install eth2spec
   ```

2. Install from source with venv:
   
   a. Download the source code:
      
      ```
      git clone https://github.com/ethereum/consensus-specs.git
      ```
   
   b. Install with Makefile commands
      
      ```
      cd consensus-specs
      make install_test && make pyspec
      ```
Run your first pyspec program!

```python
>> from eth2spec.bellatrix import mainnet as spec

>> hello = b"Hello World"

>> body = spec.BeaconBlockBody(
     graffiti=hello + b'\0' * (32 - len(hello))
)

>> block = spec.BeaconBlock(body=body)

>> print(block.body.graffiti.decode("utf-8"))

Hello World
```
Write your first pyspec test case!

```python
@with_all_phases
@spec_state_test
def test_empty_block_transition(spec, state):
    pre_slot = state.slot
    pre_eth1_votes = len(state.eth1_data_votes)
    pre_mix = spec.get_randao_mix(state, spec.get_current_epoch(state))

    yield 'pre', state

    block = build_empty_block_for_next_slot(spec, state)
    signed_block = state_transition_and_sign_block(spec, state, block)

    yield 'blocks', [signed_block]
    yield 'post', state

    assert len(state.eth1_data_votes) == pre_eth1_votes + 1
    assert spec.get_block_root_at_slot(state, pre_slot) == signed_block.message.parent_root
    assert spec.get_randao_mix(state, spec.get_current_epoch(state)) != pre_mix
```
@with_all_phases
@spec_state_test
def test_empty_block_transition(spec, state):
    pre_slot = state.slot
    pre_eth1_votes = len(state.eth1_data_votes)
    pre_mix = spec.get_randao_mix(state, spec.get_current_epoch(state))

    yield 'pre', state

    block = build_empty_block_for_next_slot(spec, state)
    signed_block = state_transition_and_sign_block(spec, state, block)

    yield 'blocks', [signed_block]
    yield 'post', state

    assert len(state.eth1_data_votes) == pre_eth1_votes + 1
    assert spec.get_block_root_at_slot(state, pre_slot) == signed_block.message.parent_root
    assert spec.get_randao_mix(state, spec.get_current_epoch(state)) != pre_mix
Write your first pyspec test case!

```python
@with_all_phases
@spec_state_test
def test_empty_block_transition(spec, state):
    pre_slot = state.slot
    pre_eth1_votes = len(state.eth1_data_votes)
    pre_mix = spec.get_randao_mix(state, spec.get_current_epoch(state))

    yield 'pre', state

    block = build_empty_block_for_next_slot(spec, state)
    signed_block = state_transition_and_sign_block(spec, state, block)

    yield 'blocks', [signed_block]
    yield 'post', state

    assert len(state.eth1_data_votes) == pre_eth1_votes + 1
    assert spec.get_block_root_at_slot(state, pre_slot) == signed_block.message.parent_root
    assert spec.get_randao_mix(state, spec.get_current_epoch(state)) != pre_mix
```
Pyspec as the **test vector** generator

```python
@with_all_phases
@spec_state_test
def test_empty_block_transition(spec, state):
    pre_slot = state.slot
    pre_eth1_votes = len(state.eth1_data_votes)
    pre_mix = spec.get_randao_mix(state, spec.get_current_epoch(state))

    yield 'pre', state

    block = build_empty_block_for_next_slot(spec, state)
    signed_block = state_transition_and_sign_block(spec, state, block)

    yield 'blocks', [signed_block]
    yield 'post', state

    assert len(state.eth1_data_votes) == pre_eth1_votes + 1
    assert spec.get_block_root_at_slot(state, pre_slot) == signed_block.message.parent_root
    assert spec.get_randao_mix(state, spec.get_current_epoch(state)) != pre_mix
```
Pyspec as the test vector generator

```python
@with_all_phases
@spec_state_test
def test_empty_block_transition(spec, state):
    pre_slot = state.slot
    pre_eth1_votes = len(state.eth1_data_votes)
    pre_mix = spec.get_randao_mix(state, spec.get_current_epoch(state))

    yield 'pre', state
    block = build_empty_block_for_next_slot(spec, state)
    signed_block = state_transition_and_sign_block(spec, state, block)

    yield 'blocks', [signed_block]
    yield 'post', state

    assert len(state.eth1_data_votes) == pre_eth1_votes + 1
    assert spec.get_block_root_at_slot(state, pre_slot) == signed_block.message.parent_root
    assert spec.get_randao_mix(state, spec.get_current_epoch(state)) != pre_mix

    post_state = state_transition(pre_state, block)
```
Documents

- Pyspec:
  https://github.com/ethereum/consensus-specs/blob/dev/tests/README.md
- Test formats:
  https://github.com/ethereum/consensus-specs/blob/dev/tests/formats/README.md
How to contribute to pyspec?

- Level 1: 👀 Look through the specifications files to learn about the specifications logic and help review it.
- Level 2: 🐍 Help refactor the codebase.
- Level 3: 😈 Try to hack some new edge test cases!
- Level 4: 💰 Submit to bug bounty (https://ethereum.org/en/bug-bounty/)
Job Description
Thank you!

Hsiao-Wei Wang
Researcher, Ethereum Foundation
hww @ ethereum.org
@icebearhww

Onigiri images: furiirakun.com