

---

# **py-ssz Documentation**

***Release 0.5.0***

**The Ethereum Foundation**

**Apr 01, 2024**



# CONTENTS

<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	ssz package . . . . .	3
1.1.1	Subpackages . . . . .	3
1.1.2	Submodules . . . . .	12
1.1.3	ssz.abc module . . . . .	12
1.1.4	ssz.codec module . . . . .	13
1.1.5	ssz.constants module . . . . .	13
1.1.6	ssz.exceptions module . . . . .	13
1.1.7	ssz.hash module . . . . .	13
1.1.8	ssz.hash_tree module . . . . .	13
1.1.9	ssz.hashable_container module . . . . .	15
1.1.10	ssz.hashable_list module . . . . .	16
1.1.11	ssz.hashable_structure module . . . . .	17
1.1.12	ssz.hashable_vector module . . . . .	18
1.1.13	ssz.tree_hash module . . . . .	19
1.1.14	ssz.typing module . . . . .	19
1.1.15	ssz.utils module . . . . .	19
1.1.16	Module contents . . . . .	19
1.2	Release Notes . . . . .	19
1.2.1	py-ssz v0.5.0 (2024-04-01) . . . . .	19
1.2.2	py-ssz v0.4.0 (2023-12-07) . . . . .	20
1.2.3	py-ssz v0.3.1 (2023-06-08) . . . . .	20
1.2.4	py-ssz v0.3.0 (2022-08-19) . . . . .	20
1.2.5	v0.2.4 . . . . .	21
1.2.6	v0.1.0-alpha.8 . . . . .	21
1.2.7	v0.1.0-alpha.7 . . . . .	21
1.2.8	v0.1.0-alpha.6 . . . . .	21
1.2.9	v0.1.0-alpha.5 . . . . .	21
1.2.10	v0.1.0-alpha.4 . . . . .	22
1.2.11	v0.1.0-alpha.3 . . . . .	22
1.2.12	v0.1.0-alpha.2 . . . . .	22
1.2.13	v0.1.0-alpha.1 . . . . .	22
1.2.14	v0.1.0-alpha.0 . . . . .	22
<b>2</b>	<b>Indices and tables</b>	<b>23</b>
	<b>Python Module Index</b>	<b>25</b>
	<b>Index</b>	<b>27</b>



Python implementation of the Simple Serialization encoding and decoding



## CONTENTS

## 1.1 ssz package

### 1.1.1 Subpackages

**ssz.cache package**

**Submodules**

**ssz.cache.cache module**

**class** `ssz.cache.cache.SSZCache(cache_size: int = 1024)`

Bases: `MutableMapping`

**property** `cache_size: int`

**clear()** → None. Remove all items from D.

**ssz.cache.utils module**

`ssz.cache.utils.get_base_key(sedes: BaseSedes[Any, Any], value: Any)` → bytes

`ssz.cache.utils.get_key(sedes, value: Any)` → str

`ssz.cache.utils.get_merkle_leaves_with_cache(value: Any, element_sedes: BaseSedes[Any, Any], cache: CacheObj)` → `Iterable[Hash32]`

Generate the merkle leaves for every element in *value*, from the cache.

NOTE: cache will be mutated when any new merkle leaves are generated.

`ssz.cache.utils.get_merkle_leaves_without_cache(value: Any, element_sedes: BaseSedes[Any, Any])` → `Iterable[Hash32]`

## Module contents

### ssz.sedes package

#### Submodules

#### ssz.sedes.base module

```
class ssz.sedes.base.BaseBitfieldCompositeSedes(*args, **kws)
    Bases: BaseSedes[TSerializable, TDeserialized]

class ssz.sedes.base.BaseProperCompositeSedes(*args, **kws)
    Bases: BaseSedes[TSerializable, TDeserialized]

    abstract property chunk_count: int | None

    abstract property element_size_in_tree: int

    abstract get_element_sedes(index: int) → BaseSedes

    abstract property is_packing: bool

    abstract serialize_element_for_tree(index: int, element: TSerializable) → bytes

class ssz.sedes.base.BaseSedes(*args, **kws)
    Bases: ABC, Generic[TSerializable, TDeserialized]

    abstract deserialize(data: bytes) → TDeserialized

    abstract get_fixed_size() → int

    abstract get_hash_tree_root(value: TSerializable) → Hash32

    abstract get_hash_tree_root_and_leaves(value: TSerializable, cache: CacheObj) → Tuple[Hash32, CacheObj]

    abstract get_key(value: Any) → str

    abstract get_sedes_id() → str

    abstract property is_fixed_sized: bool

    abstract serialize(value: TSerializable) → bytes
```

#### ssz.sedes.basic module

```
class ssz.sedes.basic.BasicSedes(size: int)
    Bases: BaseSedes[TSerializable, TDeserialized]

    get_fixed_size()

    get_hash_tree_root(value: TSerializable) → Hash32

    get_hash_tree_root_and_leaves(value: TSerializable, cache: CacheObj) → Tuple[Hash32, CacheObj]

    get_key(value: Any) → str
```



```

    is_fixed_sized = True

class ssz.sedes.basic.BitfieldCompositeSedes(*args, **kwargs)
    Bases: BaseBitfieldCompositeSedes[TSerializable, TDeserialized]
    get_key(value: Any) → str

class ssz.sedes.basic.HomogeneousProperCompositeSedes(*args, **kwargs)
    Bases: ProperCompositeSedes[TSerializable, TDeserialized]
    property chunk_count: int
    get_sedes_id() → str
    property is_packing: bool

class ssz.sedes.basic.ProperCompositeSedes(*args, **kwargs)
    Bases: BaseProperCompositeSedes[TSerializable, TDeserialized]
    deserialize(data: bytes) → TDeserialized
    property element_size_in_tree: int
    get_key(value: Any) → str
    serialize(value: TSerializable) → bytes
    serialize_element_for_tree(index: int, element: TSerializable) → bytes

```

### ssz.sedes.bitlist module

```

class ssz.sedes.bitlist.Bitlist(max_bit_count: int)
    Bases: BitfieldCompositeSedes[Union[bytes, bytearray], bytes]
    property chunk_count: int
    deserialize(data: bytes) → Tuple[bool, ...]
    get_fixed_size()
    get_hash_tree_root(value: Sequence[bool]) → bytes
    get_hash_tree_root_and_leaves(value: Sequence[bool], cache: CacheObj) → Tuple[Hash32,
                                                                                     CacheObj]
    get_sedes_id() → str
    is_fixed_sized = False
    serialize(value: Sequence[bool]) → bytes

ssz.sedes.bitlist.get_bitlist_len(x: int) → int

```

### ssz.sedes.bitvector module

```
class ssz.sedes.bitvector.Bitvector(bit_count: int)
    Bases: BitfieldCompositeSedes[Union[bytes, bytearray], bytes]
    property chunk_count: int
    deserialize(data: bytes) → bytes
    get_fixed_size()
    get_hash_tree_root(value: Sequence[bool]) → bytes
    get_hash_tree_root_and_leaves(value: Sequence[bool], cache: CacheObj) → Tuple[Hash32,
                                                                                      CacheObj]
    get_sedes_id() → str
    is_fixed_sized = True
    serialize(value: Sequence[bool]) → bytes
```

### ssz.sedes.boolean module

```
class ssz.sedes.boolean.Bit
    Bases: Boolean
class ssz.sedes.boolean.Boolean
    Bases: BasicSedes[bool, bool]
    deserialize(data: bytes) → bool
    get_sedes_id() → str
    serialize(value: bool) → bytes
```

### ssz.sedes.byte module

```
class ssz.sedes.byte.Byte
    Bases: BasicSedes[bytes, bytes]
    deserialize(data: bytes) → bytes
    get_sedes_id() → str
    serialize(value: bytes) → bytes
    size = 1
```

## ssz.sedes.byte\_list module

**class** `ssz.sedes.byte_list.ByteList(max_length: int)`

Bases: `List[Union[bytes, bytearray], bytes]`

Equivalent to `List(byte, size)` but more convenient & efficient.

When encoding a series of bytes, `List(byte, ...)` requires an awkward input shaped like: `(b'A', b'B', b'C')`. `ByteList` accepts a simple `bytes` object like `b'ABC'` for encoding.

**deserialize**(data: bytes) → bytes

**get\_hash\_tree\_root**(value: bytes) → bytes

**get\_sedes\_id**() → str

**serialize**(value: bytes | bytearray) → bytes

**serialize\_element\_for\_tree**(index: int, byte\_value: int) → bytes

## ssz.sedes.byte\_vector module

**class** `ssz.sedes.byte_vector.ByteVector(size: int)`

Bases: `Vector[Union[bytes, bytearray], bytes]`

Equivalent to `Vector(byte, size)` but more efficient.

**deserialize**(data: bytes) → bytes

**get\_hash\_tree\_root**(value: bytes) → bytes

**get\_hash\_tree\_root\_and\_leaves**(value: bytes, cache: CacheObj) → `Tuple[Hash32, CacheObj]`

**get\_sedes\_id**() → str

**serialize**(value: bytes | bytearray) → bytes

**serialize\_element\_for\_tree**(index: int, byte\_value: int) → bytes

## ssz.sedes.container module

**class** `ssz.sedes.container.Container(field_sedes: Sequence[BaseSedes[Any, Any]])`

Bases: `ProperCompositeSedes[Sequence[Any], Tuple[Any, ...]]`

**property** `chunk_count`: int

**deserialize\_fixed\_size\_parts**(stream: IO[bytes]) → `Iterable[Tuple[Tuple[Any], Tuple[int, BaseSedes[Any, Any]]]]`

**deserialize\_variable\_size\_parts**(offset\_pairs: `Tuple[Tuple[int, BaseSedes[Any, Any]], ...]`, stream: IO[bytes]) → `Iterable[Any]`

**get\_element\_sedes**(index: int) → `BaseSedes`

**get\_fixed\_size**()

```
get_hash_tree_root(value: Tuple[Any, ...]) → bytes
get_hash_tree_root_and_leaves(value: Tuple[Any, ...], cache: CacheObj) → Tuple[Hash32, CacheObj]
get_sedes_id() → str
property is_fixed_sized
property is_packing: bool
serialize(value) → bytes
```

### ssz.sedes.list module

```
class ssz.sedes.list.List(element_sedes: BaseSedes[Any, Any], max_length: int)
    Bases: HomogeneousProperCompositeSedes[Sequence[TSerializable], Tuple[TDSerialized, ...]]
    get_element_sedes(index) → BaseSedes[TSerializable, TDSerialized]
    get_fixed_size()
    get_hash_tree_root(value: Iterable[TSerializable]) → bytes
    get_hash_tree_root_and_leaves(value: TSerializable, cache: CacheObj) → Tuple[Hash32, CacheObj]
    is_fixed_sized = False
```

### ssz.sedes.serializable module

```
class ssz.sedes.serializable.BaseSerializable(*args, cache=None, **kwargs)
    Bases: Sequence
    as_dict()
    cache = None
    copy(*args, **kwargs)
    get_key() → bytes
    classmethod get_sedes_id() → str
    property hash_tree_root
    reset_cache()

class ssz.sedes.serializable.Meta(has_fields, fields, container_sedes, field_names, field_attrs)
    Bases: tuple
    container_sedes: Container | None
        Alias for field number 2
    field_attrs: Tuple[str, ...] | None
        Alias for field number 4
    field_names: Tuple[str, ...] | None
        Alias for field number 3
```

**fields:** `Tuple[Tuple[str, BaseSedes], ...] | None`

Alias for field number 1

**has\_fields:** `bool`

Alias for field number 0

**class** `ssz.sedes.serializable.MetaSerializable(name, bases, namespace)`

Bases: `ABCMeta`

**deserialize**(data: `bytes`) → `TSerializable`

**get\_fixed\_size**()

**get\_hash\_tree\_root**(value: `TSerializable`, cache: `bool = True`) → `bytes`

**property** `is_fixed_sized`

**serialize**(value: `TSerializable`) → `bytes`

**class** `ssz.sedes.serializable.Serializable(*args, cache=None, **kwargs)`

Bases: `BaseSerializable`

The base class for serializable objects.

`ssz.sedes.serializable.make_immutable(value)`

`ssz.sedes.serializable.merge_args_to_kwargs(args, kwargs, arg_names)`

`ssz.sedes.serializable.merge_kwargs_to_args(args, kwargs, arg_names)`

`ssz.sedes.serializable.validate_args_and_kwargs(args, kwargs, arg_names)`

## ssz.sedes.signed\_serializable module

**class** `ssz.sedes.signed_serializable.MetaSignedSerializable(name, bases, namespace)`

Bases: `MetaSerializable`

**class** `ssz.sedes.signed_serializable.SignedMeta(has_fields, fields, container_sedes, signed_container_sedes, field_names, field_attrs)`

Bases: `tuple`

**container\_sedes:** `Container | None`

Alias for field number 2

**field\_attrs:** `Tuple[str, ...] | None`

Alias for field number 5

**field\_names:** `Tuple[str, ...] | None`

Alias for field number 4

**fields:** `Tuple[Tuple[str, BaseSedes]] | None`

Alias for field number 1

**has\_fields:** `bool`

Alias for field number 0

**signed\_container\_sedes:** `Container | None`

Alias for field number 3

```
class ssz.sedes.signed_serializable.SignedSerializable(*args, cache=None, **kwargs)
    Bases: BaseSerializable
    property signing_root
```

### ssz.sedes.uint module

```
class ssz.sedes.uint.UInt(num_bits: int)
    Bases: BasicSedes[int, int]
    deserialize(data: bytes) → int
    get_sedes_id() → str
    serialize(value: int) → bytes
```

### ssz.sedes.vector module

```
class ssz.sedes.vector.Vector(element_sedes: BaseSedes[Any, Any], length: int)
    Bases: HomogeneousProperCompositeSedes[Sequence[TSerializable], Tuple[TDeserialized, ...]]
    get_element_sedes(index) → BaseSedes[TSerializable, TDeserialized]
    get_fixed_size() → int
    get_hash_tree_root(value: Sequence[Any]) → bytes
    get_hash_tree_root_and_leaves(value: Sequence[Any], cache: CacheObj) → Tuple[Hash32,
                                                                                   CacheObj]
    property is_fixed_sized: bool
    property length: int
```

### Module contents

```
ssz.sedes.infer_sedes(value)
    Try to find a sedes objects suitable for a given Python object.
```

### ssz.tools package

#### Submodules

#### ssz.tools.codec module

```
class ssz.tools.codec.DefaultCodec
    Bases: object
    static decode_bool(value, sedes) → bool
    static decode_bytes(value, sedes) → bytes
```

```
static decode_integer(value, sedes) → int  
static encode_bool(value: bool, sedes)  
static encode_bytes(value: bytes, sedes)  
static encode_integer(value: int, sedes)
```

## ssz.tools.dump module

```
ssz.tools.dump.dump(value, sedes=None, codec=<class 'ssz.tools.codec.DefaultCodec'>)  
ssz.tools.dump.dump_bits(value, sedes, codec)  
ssz.tools.dump.dump_boolean(value, sedes, codec)  
ssz.tools.dump.dump_bytes(value, sedes, codec)  
ssz.tools.dump.dump_container(value, sedes, codec)  
ssz.tools.dump.dump_hashable_container(value, codec)  
ssz.tools.dump.dump_hashable_sequence(value, codec)  
ssz.tools.dump.dump_integer(value, sedes, codec)  
ssz.tools.dump.dump_list(value, sedes, codec)  
ssz.tools.dump.dump_serializable(value, codec)  
ssz.tools.dump.dump_vector(value, sedes, codec)  
ssz.tools.dump.to_formatted_dict(value, sedes=None, codec=<class 'ssz.tools.codec.DefaultCodec'>)
```

## ssz.tools.parse module

```
ssz.tools.parse.from_formatted_dict(value, sedes, codec=<class 'ssz.tools.codec.DefaultCodec'>)  
ssz.tools.parse.parse(value, sedes, codec=<class 'ssz.tools.codec.DefaultCodec'>)  
ssz.tools.parse.parse_bits(value, sedes, codec)  
ssz.tools.parse.parse_boolean(value, sedes, codec)  
ssz.tools.parse.parse_bytes(value, sedes, codec)  
ssz.tools.parse.parse_container(value, sedes, codec)  
ssz.tools.parse.parse_hashable(value, hashable_cls, codec)  
ssz.tools.parse.parse_integer(value, sedes, codec)  
ssz.tools.parse.parse_list(value, sedes, codec)  
ssz.tools.parse.parse_serializable(value, serializable_cls, codec)  
ssz.tools.parse.parse_vector(value, sedes, codec)
```

## Module contents

### 1.1.2 Submodules

#### 1.1.3 ssz.abc module

```
class ssz.abc.HashableStructureAPI(*args, **kwargs)
    Bases: ABC, Generic[TElement]
    abstract property chunks: PVector[Hash32]
    abstract property elements: PVector[TElement]
    abstract evolver() → HashableStructureEvolverAPI[TStructure, TElement]
    abstract classmethod from_iterable_and_sedes(iterable: Iterable[TElement], sedes:
                                                BaseProperCompositeSedes, max_length: int |
                                                None)

    abstract property hash_tree: HashTree
    abstract property hash_tree_root: Hash32
    abstract mset(*args: int | TElement) → TStructure
    abstract property raw_root: Hash32
    abstract set(index: int, value: TElement) → TStructure
    abstract transform(*transformations)

class ssz.abc.HashableStructureEvolverAPI(hashable_structure: TStructure)
    Bases: ABC, Generic[TStructure, TElement]
    abstract is_dirty() → bool
    abstract persistent() → TStructure
    abstract set(index: int, element: TElement) → None

class ssz.abc.ResizableHashableStructureAPI(*args, **kwargs)
    Bases: HashableStructureAPI[TElement]
    abstract append(value: TElement) → TStructure
    abstract evolver() → ResizableHashableStructureEvolverAPI[TStructure, TElement]
    abstract extend(values: Iterable[TElement]) → TStructure

class ssz.abc.ResizableHashableStructureEvolverAPI(hashable_structure: TStructure)
    Bases: HashableStructureEvolverAPI[TStructure, TElement]
    abstract append(element: TElement) → None
    abstract extend(iterable: Iterable[TElement]) → None
```



### 1.1.4 ssz.codec module

`ssz.codec.decode(ssz, sedes)`

Decode a SSZ encoded object.

`ssz.codec.encode(value, sedes=None)`

Encode object in SSZ format. *sedes* needs to be explicitly mentioned for encode/decode of integers(as of now). *sedes* parameter could be given as a string or as the actual sedes object itself.

### 1.1.5 ssz.constants module

### 1.1.6 ssz.exceptions module

**exception** `ssz.exceptions.DeserializationError`

Bases: *SSZException*

Exception raised if deserialization fails.

**exception** `ssz.exceptions.SSZException`

Bases: *Exception*

Base class for exceptions raised by this package.

**exception** `ssz.exceptions.SerializationError`

Bases: *SSZException*

Exception raised if serialization fails.

### 1.1.7 ssz.hash module

`ssz.hash.hash_eth2(data: bytes) → Hash32`

Return SHA-256 hashed result. Note: it's a placeholder and we aim to migrate to a S[T/N]ARK-friendly hash function in a future Ethereum 2.0 deployment phase.

### 1.1.8 ssz.hash\_tree module

**class** `ssz.hash_tree.HashTree(raw_hash_tree: PVector[PVector[Hash32]], chunk_count: int | None = None)`

Bases: *PVector[Hash32]*

**append**(value: Hash32) → *HashTree*

**property chunks:** *PVector[Hash32]*

**classmethod compute**(chunks: *Iterable[Hash32]*, chunk\_count: int | None = None) → *HashTree*

**count**(value) → integer -- return number of occurrences of value

**delete**(index: int, stop: int | None = None) → *HashTree*

**evolver**()

**extend**(value: *Iterable[Hash32]*) → *HashTree*

**index**(value[, start[, stop ]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

**mset**(\*args: int | Hash32) → HashTree

**remove**(value: Hash32) → HashTree

**property root:** Hash32

**set**(index: int, value: Hash32) → HashTree

**transform**(\*transformations)

**class** ssz.hash\_tree.HashTreeEvolver(hash\_tree: HashTree)

Bases: object

**append**(value: Hash32) → None

**delete**(index, stop=None)

**extend**(values: Iterable[Hash32]) → None

**is\_dirty**() → bool

**persistent**() → HashTree

**remove**(value)

**set**(index: Integral, value: Hash32) → None

ssz.hash\_tree.append\_chunk\_to\_tree(hash\_tree: PVector[PVector[Hash32]], chunk: Hash32) → PVector[PVector[Hash32]]

ssz.hash\_tree.compute\_hash\_tree(chunks: Iterable[Hash32], chunk\_count: int | None = None) → PVector[PVector[Hash32]]

ssz.hash\_tree.generate\_chunk\_tree\_padding(unpadded\_chunk\_tree: PVector[Hash32], chunk\_count: int | None) → Generator[Hash32, None, None]

ssz.hash\_tree.generate\_hash\_tree\_layers(chunks: PVector[Hash32]) → Generator[PVector[Hash32], None, None]

ssz.hash\_tree.get\_num\_layers(num\_chunks: int, chunk\_count: int | None) → int

ssz.hash\_tree.hash\_layer(child\_layer: PVector[Hash32], layer\_index: int) → PVector[Hash32]

ssz.hash\_tree.pad\_hash\_tree(unpadded\_chunk\_tree: PVector[PVector[Hash32]], chunk\_count: int | None = None) → PVector[PVector[Hash32]]

ssz.hash\_tree.recompute\_hash\_in\_tree(hash\_tree: PVector[PVector[Hash32]], layer\_index: int, hash\_index: int) → PVector[PVector[Hash32]]

ssz.hash\_tree.set\_chunk\_in\_tree(hash\_tree: PVector[PVector[Hash32]], index: int, chunk: Hash32) → PVector[PVector[Hash32]]

ssz.hash\_tree.validate\_chunk\_count(chunk\_count: int | None) → None

ssz.hash\_tree.validate\_raw\_hash\_tree(raw\_hash\_tree: PVector[PVector[Hash32]], chunk\_count: int | None = None) → None

### 1.1.9 ssz.hashable\_container module

```

class ssz.hashable_container.FieldDescriptor(name: str)
    Bases: object
    Descriptor translating from __getattr__ to __getitem__ calls for a given attribute.

ssz.hashable_container.GenericMetaHashableContainer
    alias of MetaHashableContainer

ssz.hashable_container.GenericMetaSignedHashableContainer
    alias of MetaSignedHashableContainer

class ssz.hashable_container.HashableContainer(*args, **kwargs)
    Bases: BaseHashableStructure[TElement]
    Base class for hashable containers.

    classmethod create(**field_kwargs: Dict[str, Any])

    evolver() → HashableContainerEvolver[TStructure, TElement]

    property hash_tree_root: Hash32

    normalize_item_index(index: str | int) → int

class ssz.hashable_container.HashableContainerEvolver(hashable_structure: TStructure)
    Bases: HashableStructureEvolver[TStructure, TElement]
    Base class for evolvers for hashable containers.

    Subclasses (created dynamically by MetaHashableContainer when creating the corresponding HashableContainer) should add settable field descriptors for all fields.

class ssz.hashable_container.Meta(fields, field_names, field_names_to_element_indices, container_sedes,
                                   evolver_class)

    Bases: tuple

    container_sedes: Container
        Alias for field number 3

    evolver_class: Type[HashableContainerEvolver]
        Alias for field number 4

    field_names: Tuple[str, ...]
        Alias for field number 1

    field_names_to_element_indices: Dict[str, int]
        Alias for field number 2

    fields: Tuple[Tuple[str, BaseSedes], ...]
        Alias for field number 0

    classmethod from_fields(fields: Tuple[Tuple[str, BaseSedes], ...], container: Container, name: str) →
        Meta

```

```
class ssz.hashable_container.MetaHashableContainer(name: str, bases: Tuple[Type, ...], namespace: Dict[str, Any])
```

Bases: [ABCMeta](#)

Metaclass which creates HashableContainers.

**deserialize**(data)

**get\_fixed\_size**()

**get\_hash\_tree\_root**(value)

**get\_hash\_tree\_root\_and\_leaves**(value, cache)

**get\_key**(value)

**get\_sedes\_id**()

**property is\_fixed\_sized**

**serialize**(value)

```
class ssz.hashable_container.MetaSignedHashableContainer(name, bases, namespace)
```

Bases: [MetaHashableContainer](#)

```
class ssz.hashable_container.SettableFieldDescriptor(name: str)
```

Bases: [FieldDescriptor](#)

Settable variant of FieldDescriptor for evolver.

```
class ssz.hashable_container.SignedHashableContainer(*args, **kwargs)
```

Bases: [HashableContainer](#)[TElement]

**property signing\_root**: Hash32

```
ssz.hashable_container.get_field_sedes_from_fields(fields: Sequence[Tuple[str, BaseSedes]]) → Generator[BaseSedes, None, None]
```

```
ssz.hashable_container.get_meta_from_bases(bases: Tuple[Type, ...]) → Meta | None
```

Return the meta object defined by one of the given base classes.

Returns None if no base defines a meta object. Raises a TypeError if more than one do.

```
ssz.hashable_container.hashablify_field_kwargs(field_kwargs: Dict[str, Any], fields: Sequence[Tuple[str, BaseSedes]]) → Generator[Tuple[str, Any], None, None]
```

```
ssz.hashable_container.hashablify_value(value: Any, sedes: BaseSedes) → Any
```

### 1.1.10 ssz.hashable\_list module

```
class ssz.hashable_list.HashableList(elements: PVector[TElement], hash_tree: HashTree, sedes: BaseProperCompositeSedes, max_length: int | None = None)
```

Bases: [BaseResizableHashableStructure](#)[TElement], [Sequence](#)[TElement]

**classmethod from\_iterable**(iterable: Iterable[TElement], sedes: List[TElement, TElement])

**property hash\_tree\_root**: Hash32

### 1.1.11 ssz.hashable\_structure module

```

class ssz.hashable_structure.BaseHashableStructure(elements: PVector[TElement], hash_tree:
                                                    HashTree, sedes: BaseProperCompositeSedes,
                                                    max_length: int | None = None)

    Bases: HashableStructureAPI[TElement]
    property chunks: PVector[Hash32]
    property elements: PVector[TElement]
    evolver() → HashableStructureEvolverAPI[TStructure, TElement]
    classmethod from_iterable_and_sedes(iterable: Iterable[TElement], sedes:
                                        BaseProperCompositeSedes, max_length: int | None = None)

    property hash_tree: HashTree
    property max_length: int | None
    mset(*args: int | TElement) → TStructure
    property raw_root: Hash32
    property sedes: BaseProperCompositeSedes
    set(index: int, value: TElement) → TStructure
    transform(*transformations)

class ssz.hashable_structure.BaseResizableHashableStructure(elements: PVector[TElement],
                                                            hash_tree: HashTree, sedes:
                                                            BaseProperCompositeSedes,
                                                            max_length: int | None = None)

    Bases: BaseHashableStructure, ResizableHashableStructureAPI[TElement]
    append(value: TElement) → TResizableStructure
    evolver() → ResizableHashableStructureEvolverAPI[TResizableStructure, TElement]
    extend(values: Iterable[TElement]) → TResizableStructure

class ssz.hashable_structure.HashableStructureEvolver(hashable_structure: TStructure)
    Bases: HashableStructureEvolverAPI[TStructure, TElement]
    is_dirty() → bool
    persistent() → TStructure
    set(index: int, element: TElement) → None

class ssz.hashable_structure.ResizableHashableStructureEvolver(hashable_structure: TStructure)
    Bases: HashableStructureEvolver, ResizableHashableStructureEvolverAPI[TStructure,
    TElement]
    append(element: TElement) → None
    extend(elements: Iterable[TElement]) → None

```

```
ssz.hashable_structure.get_appended_chunks(*, appended_elements: Sequence[bytes], element_size: int,
                                           num_padding_elements: int) → Generator[Hash32, None,
                                           None]
```

Get the sequence of appended chunks.

```
ssz.hashable_structure.get_num_padding_elements(*, num_original_elements: int, num_original_chunks:
                                              int, element_size: int) → int
```

Compute the number of elements that would still fit in the empty space of the last chunk.

```
ssz.hashable_structure.get_updated_chunks(*, updated_elements: Dict[int, bytes], appended_elements:
                                         Sequence[bytes], original_chunks: Sequence[Hash32],
                                         element_size: int, num_original_elements: int,
                                         num_padding_elements: int) → Generator[Tuple[int,
                                         Hash32], None, None]
```

For an element changeset, compute the updates that have to be applied to the existing chunks.

The changeset is given as a dictionary of element indices to updated elements and a sequence of appended elements. Note that appended elements that do not affect existing chunks are ignored.

The pre-existing state is given by the sequence of original chunks and the number of elements represented by these chunks.

The return value is a dictionary mapping chunk indices to chunks.

```
ssz.hashable_structure.update_element_in_chunk(original_chunk: Hash32, index: int, element: bytes) →
Hash32
```

Replace part of a chunk with a given element.

The chunk is interpreted as a concatenated sequence of equally sized elements. This function replaces the element given by its index in the chunk with the given data.

If the length of the element is zero or not a divisor of the chunk size, a *ValueError* is raised. If the index is out of range, an *IndexError* is raised.

```
>>> from ssz.hashable_structure import update_element_in_chunk
>>> update_element_in_chunk(b"aabbcc", 1, b"xx")
b'aaxxcc'
```

```
ssz.hashable_structure.update_elements_in_chunk(original_chunk: Hash32, updated_elements: Dict[int,
                                                  bytes]) → Hash32
```

Update multiple elements in a chunk.

The set of updates is given by a dictionary mapping indices to elements. The items of the dictionary will be passed one by one to *update\_element\_in\_chunk*.

### 1.1.12 ssz.hashable\_vector module

```
class ssz.hashable_vector.HashableVector(elements: PVector[TElement], hash_tree: HashTree, sedes:
                                         BaseProperCompositeSedes, max_length: int | None = None)
```

Bases: *BaseHashableStructure*[TElement], *Sequence*[TElement]

```
classmethod from_iterable(iterable: Iterable[TElement], sedes: Vector[TElement, TElement])
```

```
property hash_tree_root: Hash32
```

### 1.1.13 ssz.tree\_hash module

`ssz.tree_hash.get_hash_tree_root(value: Any, sedes: BaseSedes | None = None) → Hash32`

### 1.1.14 ssz.typing module

### 1.1.15 ssz.utils module

`ssz.utils.decode_offset(data: bytes) → int`

`ssz.utils.encode_offset(offset: int) → bytes`

`ssz.utils.get_duplicates(values)`

`ssz.utils.get_items_per_chunk(item_size: int) → int`

`ssz.utils.get_next_power_of_two(value: int) → int`

`ssz.utils.get_serialized_bytearray(value: Sequence[bool], bit_count: int, extra_byte: bool) → bytearray`

`ssz.utils.is_immutable_field_value(value: Any) → bool`

`ssz.utils.merkleize(chunks: Sequence[Hash32], limit: int | None = None) → Hash32`

`ssz.utils.merkleize_with_cache(chunks: Sequence[Hash32], cache: CacheObj, limit: int | None = None) → Tuple[Hash32, CacheObj]`

`ssz.utils.mix_in_length(root: Hash32, length: int) → Hash32`

`ssz.utils.pack(serialized_values: Sequence[bytes]) → Tuple[Hash32, ...]`

`ssz.utils.pack_bits(values: Sequence[bool]) → Tuple[Hash32]`

`ssz.utils.pack_bytes(byte_string: bytes) → Tuple[bytes, ...]`

`ssz.utils.pad_zeros(value: bytes) → bytes`

`ssz.utils.read_exact(num_bytes: int, stream: IO[bytes]) → bytes`

`ssz.utils.s_decode_offset(stream: IO[bytes]) → int`

`ssz.utils.to_chunks(packed_data: bytes) → Tuple[bytes, ...]`

### 1.1.16 Module contents

## 1.2 Release Notes

### 1.2.1 py-ssz v0.5.0 (2024-04-01)

#### Internal Changes - for py-ssz Contributors

- Add python 3.12 support, add all doc type tests and night runs to CI, add blocklint to linting, turned off yaml tests in CI (#136)

## 1.2.2 py-ssz v0.4.0 (2023-12-07)

### Breaking Changes

- Drop support for python 3.7 (#134)

### Internal Changes - for py-ssz Contributors

- Add `build.os` config to `readthedocs.yml` (#133)
- Merge changes from the template, including use `pre-commit` for linting and change the name of the master branch to `main` (#134)

## 1.2.3 py-ssz v0.3.1 (2023-06-08)

### Internal Changes - for py-ssz Contributors

- pull in most project template updates (#128)
- bump `eth-utils` requirement version to `>=2` (#132)

## 1.2.4 py-ssz v0.3.0 (2022-08-19)

### Breaking changes

- Dropping official support for Python 3.6 (although it still worked as of the last test run). (#125)

### Features

- Add a `ByteList` sedes that is more convenient and more performant. With `ByteList`, the caller can decode a `bytes` object, rather than passing in a list of single-byte elements. (#118)

### Bugfixes

- Reject empty bytes at the end of a bitlist as invalid (#109)
- Reject vectors and bitvectors of length 0 as invalid, as defined in the spec. (#111)
- Enforce that vector types must have a maximum length of 1 or more, and lists may have a 0 max length (#116)

### Improved Documentation

- Sort release notes with most recent on top (#124)



## Internal Changes - for py-ssz Contributors

- Upgrade black to a stable version, and pass newest style checks (#120)
- Use the latest project template, which gives many developer-focused benefits: in making release notes, releasing new versions, etc. (#121)
- Miscellaneous changes (#124):
  - Run black autoformat, as part of `make lint-roll`
  - Added some tests to check length validation of `ByteList` and `ByteVector`
  - When generating website docs from docstrings, skip tests

### 1.2.5 v0.2.4

Released 2020-03-24

- Update *pyrsistent* dependency.

### 1.2.6 v0.1.0-alpha.8

Released 2018-05-05

- Less strict class relationship requirement for equality of serializables - #71

### 1.2.7 v0.1.0-alpha.7

Released 2018-05-02

- Fix equality of serializable objects - #64
- Add helpers to convert objects to and from a human readable representation - #66
- Cache hash tree root of serializable objects - #68

### 1.2.8 v0.1.0-alpha.6

Released 2018-04-23

No changes

### 1.2.9 v0.1.0-alpha.5

Released 2018-04-23

- Slight change in serializable inheritance rules - #57
- Add root property to serializable - #57
- Add SignedSerializable base class with signing-root property - #57

### 1.2.10 v0.1.0-alpha.4

Released 2018-04-09

- Fix bug in serializable class - [#56](#)

### 1.2.11 v0.1.0-alpha.3

Released 2018-04-04

- Implement spec version 0.5.0

### 1.2.12 v0.1.0-alpha.2

Released 2018-02-05

- Add zero padding to tree hash - [#35](#)

### 1.2.13 v0.1.0-alpha.1

Released 2018-02-05

- Implements January pre-release spec

### 1.2.14 v0.1.0-alpha.0

- Launched repository, claimed names for pip, RTD, github, etc

## INDICES AND TABLES

- `genindex`
- `modindex`



## PYTHON MODULE INDEX

### S

- ssz, 19
- ssz.abc, 12
- ssz.cache, 4
- ssz.cache.cache, 3
- ssz.cache.utils, 3
- ssz.codec, 13
- ssz.constants, 13
- ssz.exceptions, 13
- ssz.hash, 13
- ssz.hash\_tree, 13
- ssz.hashable\_container, 15
- ssz.hashable\_list, 16
- ssz.hashable\_structure, 17
- ssz.hashable\_vector, 18
- ssz.sedes, 10
- ssz.sedes.base, 4
- ssz.sedes.basic, 4
- ssz.sedes.bitlist, 5
- ssz.sedes.bitvector, 6
- ssz.sedes.boolean, 6
- ssz.sedes.byte, 6
- ssz.sedes.byte\_list, 7
- ssz.sedes.byte\_vector, 7
- ssz.sedes.container, 7
- ssz.sedes.list, 8
- ssz.sedes.serializable, 8
- ssz.sedes.signed\_serializable, 9
- ssz.sedes.uint, 10
- ssz.sedes.vector, 10
- ssz.tools, 12
- ssz.tools.codec, 10
- ssz.tools.dump, 11
- ssz.tools.parse, 11
- ssz.tree\_hash, 19
- ssz.typing, 19
- ssz.utils, 19



## INDEX

### A

`append()` (*ssz.abc.ResizableHashableStructureAPI* method), 12  
`append()` (*ssz.abc.ResizableHashableStructureEvolverAPI* method), 12  
`append()` (*ssz.hash\_tree.HashTree* method), 13  
`append()` (*ssz.hash\_tree.HashTreeEvolver* method), 14  
`append()` (*ssz.hashable\_structure.BaseResizableHashableStructure* method), 17  
`append()` (*ssz.hashable\_structure.ResizableHashableStructureEvolver* method), 17  
`append_chunk_to_tree()` (in module *ssz.hash\_tree*), 14  
`as_dict()` (*ssz.sedes.serializable.BaseSerializable* method), 8

### B

`BaseBitfieldCompositeSedes` (class in *ssz.sedes.base*), 4  
`BaseHashableStructure` (class in *ssz.hashable\_structure*), 17  
`BaseProperCompositeSedes` (class in *ssz.sedes.base*), 4  
`BaseResizableHashableStructure` (class in *ssz.hashable\_structure*), 17  
`BaseSedes` (class in *ssz.sedes.base*), 4  
`BaseSerializable` (class in *ssz.sedes.serializable*), 8  
`BasicSedes` (class in *ssz.sedes.basic*), 4  
`Bit` (class in *ssz.sedes.boolean*), 6  
`BitfieldCompositeSedes` (class in *ssz.sedes.basic*), 5  
`Bitlist` (class in *ssz.sedes.bitlist*), 5  
`Bitvector` (class in *ssz.sedes.bitvector*), 6  
`Boolean` (class in *ssz.sedes.boolean*), 6  
`Byte` (class in *ssz.sedes.byte*), 6  
`ByteList` (class in *ssz.sedes.byte\_list*), 7  
`ByteVector` (class in *ssz.sedes.byte\_vector*), 7

### C

`cache` (*ssz.sedes.serializable.BaseSerializable* attribute), 8  
`cache_size` (*ssz.cache.cache.SSZCache* property), 3

`chunk_count` (*ssz.sedes.base.BaseProperCompositeSedes* property), 4  
`chunk_count` (*ssz.sedes.basic.HomogeneousProperCompositeSedes* property), 5  
`chunk_count` (*ssz.sedes.bitlist.Bitlist* property), 5  
`chunk_count` (*ssz.sedes.bitvector.Bitvector* property), 6  
`chunk_count` (*ssz.sedes.container.Container* property), 7  
`chunks` (*ssz.abc.HashableStructureAPI* property), 12  
`chunks` (*ssz.hash\_tree.HashTree* property), 13  
`chunks` (*ssz.hashable\_structure.BaseHashableStructure* property), 17  
`clear()` (*ssz.cache.cache.SSZCache* method), 3  
`compute()` (*ssz.hash\_tree.HashTree* class method), 13  
`compute_hash_tree()` (in module *ssz.hash\_tree*), 14  
`Container` (class in *ssz.sedes.container*), 7  
`container_sedes` (*ssz.hashable\_container.Meta* attribute), 15  
`container_sedes` (*ssz.sedes.serializable.Meta* attribute), 8  
`container_sedes` (*ssz.sedes.signed\_serializable.SignedMeta* attribute), 9  
`copy()` (*ssz.sedes.serializable.BaseSerializable* method), 8  
`count()` (*ssz.hash\_tree.HashTree* method), 13  
`create()` (*ssz.hashable\_container.HashableContainer* class method), 15

### D

`decode()` (in module *ssz.codec*), 13  
`decode_bool()` (*ssz.tools.codec.DefaultCodec* static method), 10  
`decode_bytes()` (*ssz.tools.codec.DefaultCodec* static method), 10  
`decode_integer()` (*ssz.tools.codec.DefaultCodec* static method), 10  
`decode_offset()` (in module *ssz.utils*), 19  
`DefaultCodec` (class in *ssz.tools.codec*), 10  
`delete()` (*ssz.hash\_tree.HashTree* method), 13  
`delete()` (*ssz.hash\_tree.HashTreeEvolver* method), 14  
`DeserializationError`, 13

- `deserialize()` (`ssz.hashable_container.MetaHashableContainer` method), 16
  - `deserialize()` (`ssz.sedes.base.BaseSedes` method), 4
  - `deserialize()` (`ssz.sedes.basic.ProperCompositeSedes` method), 5
  - `deserialize()` (`ssz.sedes.bitlist.Bitlist` method), 5
  - `deserialize()` (`ssz.sedes.bitvector.Bitvector` method), 6
  - `deserialize()` (`ssz.sedes.boolean.Boolean` method), 6
  - `deserialize()` (`ssz.sedes.byte.Byte` method), 6
  - `deserialize()` (`ssz.sedes.byte_list.ByteList` method), 7
  - `deserialize()` (`ssz.sedes.byte_vector.ByteVector` method), 7
  - `deserialize()` (`ssz.sedes.serializable.MetaSerializable` method), 9
  - `deserialize()` (`ssz.sedes.uint.UInt` method), 10
  - `deserialize_fixed_size_parts()` (`ssz.sedes.container.Container` method), 7
  - `deserialize_variable_size_parts()` (`ssz.sedes.container.Container` method), 7
  - `dump()` (in module `ssz.tools.dump`), 11
  - `dump_bits()` (in module `ssz.tools.dump`), 11
  - `dump_boolean()` (in module `ssz.tools.dump`), 11
  - `dump_bytes()` (in module `ssz.tools.dump`), 11
  - `dump_container()` (in module `ssz.tools.dump`), 11
  - `dump_hashable_container()` (in module `ssz.tools.dump`), 11
  - `dump_hashable_sequence()` (in module `ssz.tools.dump`), 11
  - `dump_integer()` (in module `ssz.tools.dump`), 11
  - `dump_list()` (in module `ssz.tools.dump`), 11
  - `dump_serializable()` (in module `ssz.tools.dump`), 11
  - `dump_vector()` (in module `ssz.tools.dump`), 11
- ## E
- `element_size_in_tree` (`ssz.sedes.base.BaseProperCompositeSedes` property), 4
  - `element_size_in_tree` (`ssz.sedes.basic.ProperCompositeSedes` property), 5
  - `elements` (`ssz.abc.HashableStructureAPI` property), 12
  - `elements` (`ssz.hashable_structure.BaseHashableStructure` property), 17
  - `encode()` (in module `ssz.codec`), 13
  - `encode_bool()` (`ssz.tools.codec.DefaultCodec` static method), 11
  - `encode_bytes()` (`ssz.tools.codec.DefaultCodec` static method), 11
  - `encode_integer()` (`ssz.tools.codec.DefaultCodec` static method), 11
  - `encode_offset()` (in module `ssz.utils`), 19
  - `evolver()` (`ssz.abc.HashableStructureAPI` method), 12
  - `evolver()` (`ssz.abc.ResizableHashableStructureAPI` method), 12
  - `evolver()` (`ssz.hash_tree.HashTree` method), 13
  - `evolver()` (`ssz.hashable_container.HashableContainer` method), 15
  - `evolver()` (`ssz.hashable_structure.BaseHashableStructure` method), 17
  - `evolver()` (`ssz.hashable_structure.BaseResizableHashableStructure` method), 17
  - `evolver_class` (`ssz.hashable_container.Meta` attribute), 15
  - `extend()` (`ssz.abc.ResizableHashableStructureAPI` method), 12
  - `extend()` (`ssz.abc.ResizableHashableStructureEvolverAPI` method), 12
  - `extend()` (`ssz.hash_tree.HashTree` method), 13
  - `extend()` (`ssz.hash_tree.HashTreeEvolver` method), 14
  - `extend()` (`ssz.hashable_structure.BaseResizableHashableStructure` method), 17
  - `extend()` (`ssz.hashable_structure.ResizableHashableStructureEvolver` method), 17
- ## F
- `field_attrs` (`ssz.sedes.serializable.Meta` attribute), 8
  - `field_attrs` (`ssz.sedes.signed_serializable.SignedMeta` attribute), 9
  - `field_names` (`ssz.hashable_container.Meta` attribute), 15
  - `field_names` (`ssz.sedes.serializable.Meta` attribute), 8
  - `field_names` (`ssz.sedes.signed_serializable.SignedMeta` attribute), 9
  - `field_names_to_element_indices` (`ssz.hashable_container.Meta` attribute), 15
  - `FieldDescriptor` (class in `ssz.hashable_container`), 15
  - `fields` (`ssz.hashable_container.Meta` attribute), 15
  - `fields` (`ssz.sedes.serializable.Meta` attribute), 8
  - `fields` (`ssz.sedes.signed_serializable.SignedMeta` attribute), 9
  - `from_fields()` (`ssz.hashable_container.Meta` class method), 15
  - `from_formatted_dict()` (in module `ssz.tools.parse`), 11
  - `from_iterable()` (`ssz.hashable_list.HashableList` class method), 16
  - `from_iterable()` (`ssz.hashable_vector.HashableVector` class method), 18
  - `from_iterable_and_sedes()` (`ssz.abc.HashableStructureAPI` class method), 12
  - `from_iterable_and_sedes()` (`ssz.hashable_structure.BaseHashableStructure` class method), 17



## G

- `generate_chunk_tree_padding()` (in module `ssz.hash_tree`), 14
- `generate_hash_tree_layers()` (in module `ssz.hash_tree`), 14
- `GenericMetaHashableContainer` (in module `ssz.hashable_container`), 15
- `GenericMetaSignedHashableContainer` (in module `ssz.hashable_container`), 15
- `get_appended_chunks()` (in module `ssz.hashable_structure`), 17
- `get_base_key()` (in module `ssz.cache.utils`), 3
- `get_bitlist_len()` (in module `ssz.sedes.bitlist`), 5
- `get_duplicates()` (in module `ssz.utils`), 19
- `get_element_sedes()` (`ssz.sedes.base.BaseProperCompositeSedes` method), 4
- `get_element_sedes()` (`ssz.sedes.container.Container` method), 7
- `get_element_sedes()` (`ssz.sedes.list.List` method), 8
- `get_element_sedes()` (`ssz.sedes.vector.Vector` method), 10
- `get_field_sedes_from_fields()` (in module `ssz.hashable_container`), 16
- `get_fixed_size()` (`ssz.hashable_container.MetaHashableContainer` method), 16
- `get_fixed_size()` (`ssz.sedes.base.BaseSedes` method), 4
- `get_fixed_size()` (`ssz.sedes.basic.BasicSedes` method), 4
- `get_fixed_size()` (`ssz.sedes.bitlist.Bitlist` method), 5
- `get_fixed_size()` (`ssz.sedes.bitvector.Bitvector` method), 6
- `get_fixed_size()` (`ssz.sedes.container.Container` method), 7
- `get_fixed_size()` (`ssz.sedes.list.List` method), 8
- `get_fixed_size()` (`ssz.sedes.serializable.MetaSerializable` method), 9
- `get_fixed_size()` (`ssz.sedes.vector.Vector` method), 10
- `get_hash_tree_root()` (in module `ssz.tree_hash`), 19
- `get_hash_tree_root()` (`ssz.hashable_container.MetaHashableContainer` method), 16
- `get_hash_tree_root()` (`ssz.sedes.base.BaseSedes` method), 4
- `get_hash_tree_root()` (`ssz.sedes.basic.BasicSedes` method), 4
- `get_hash_tree_root()` (`ssz.sedes.bitlist.Bitlist` method), 5
- `get_hash_tree_root()` (`ssz.sedes.bitvector.Bitvector` method), 6
- `get_hash_tree_root()` (`ssz.sedes.byte_list.ByteList` method), 7
- `get_hash_tree_root()` (`ssz.sedes.byte_vector.ByteVector` method), 7
- `get_hash_tree_root()` (`ssz.sedes.container.Container` method), 7
- `get_hash_tree_root()` (`ssz.sedes.list.List` method), 8
- `get_hash_tree_root()` (`ssz.sedes.serializable.MetaSerializable` method), 9
- `get_hash_tree_root()` (`ssz.sedes.vector.Vector` method), 10
- `get_hash_tree_root_and_leaves()` (`ssz.hashable_container.MetaHashableContainer` method), 16
- `get_hash_tree_root_and_leaves()` (`ssz.sedes.base.BaseSedes` method), 4
- `get_hash_tree_root_and_leaves()` (`ssz.sedes.basic.BasicSedes` method), 4
- `get_hash_tree_root_and_leaves()` (`ssz.sedes.bitlist.Bitlist` method), 5
- `get_hash_tree_root_and_leaves()` (`ssz.sedes.bitvector.Bitvector` method), 6
- `get_hash_tree_root_and_leaves()` (`ssz.sedes.byte_vector.ByteVector` method), 7
- `get_hash_tree_root_and_leaves()` (`ssz.sedes.container.Container` method), 8
- `get_hash_tree_root_and_leaves()` (`ssz.sedes.list.List` method), 8
- `get_hash_tree_root_and_leaves()` (`ssz.sedes.vector.Vector` method), 10
- `get_items_per_chunk()` (in module `ssz.utils`), 19
- `get_key()` (in module `ssz.cache.utils`), 3
- `get_key()` (`ssz.hashable_container.MetaHashableContainer` method), 16
- `get_key()` (`ssz.sedes.base.BaseSedes` method), 4
- `get_key()` (`ssz.sedes.basic.BasicSedes` method), 4
- `get_key()` (`ssz.sedes.basic.BitfieldCompositeSedes` method), 5
- `get_key()` (`ssz.sedes.basic.ProperCompositeSedes` method), 5
- `get_key()` (`ssz.sedes.serializable.BaseSerializable` method), 8
- `get_merkle_leaves_with_cache()` (in module `ssz.cache.utils`), 3
- `get_merkle_leaves_without_cache()` (in module `ssz.cache.utils`), 3
- `get_meta_from_bases()` (in module `ssz.hashable_container`), 16
- `get_next_power_of_two()` (in module `ssz.utils`), 19
- `get_num_layers()` (in module `ssz.hash_tree`), 14
- `get_num_padding_elements()` (in module `ssz.hashable_structure`), 18
- `get_sedes_id()` (`ssz.hashable_container.MetaHashableContainer` method), 18

method), 16  
 get\_sedes\_id() (ssz.sedes.base.BaseSedes method), 4  
 get\_sedes\_id() (ssz.sedes.basic.HomogeneousProperCompositeSedes method), 5  
 get\_sedes\_id() (ssz.sedes.bitlist.Bitlist method), 5  
 get\_sedes\_id() (ssz.sedes.bitvector.Bitvector method), 6  
 get\_sedes\_id() (ssz.sedes.boolean.Boolean method), 6  
 get\_sedes\_id() (ssz.sedes.byte.Byte method), 6  
 get\_sedes\_id() (ssz.sedes.byte\_list.ByteList method), 7  
 get\_sedes\_id() (ssz.sedes.byte\_vector.ByteVector method), 7  
 get\_sedes\_id() (ssz.sedes.container.Container method), 8  
 get\_sedes\_id() (ssz.sedes.serializable.BaseSerializable class method), 8  
 get\_sedes\_id() (ssz.sedes.uint.UInt method), 10  
 get\_serialized\_bytearray() (in module ssz.utils), 19  
 get\_updated\_chunks() (in module ssz.hashable\_structure), 18

## H

has\_fields (ssz.sedes.serializable.Meta attribute), 9  
 has\_fields (ssz.sedes.signed\_serializable.SignedMeta attribute), 9  
 hash\_eth2() (in module ssz.hash), 13  
 hash\_layer() (in module ssz.hash\_tree), 14  
 hash\_tree (ssz.abc.HashableStructureAPI property), 12  
 hash\_tree (ssz.hashable\_structure.BaseHashableStructure property), 17  
 hash\_tree\_root (ssz.abc.HashableStructureAPI property), 12  
 hash\_tree\_root (ssz.hashable\_container.HashableContainer property), 15  
 hash\_tree\_root (ssz.hashable\_list.HashableList property), 16  
 hash\_tree\_root (ssz.hashable\_vector.HashableVector property), 18  
 hash\_tree\_root (ssz.sedes.serializable.BaseSerializable property), 8  
 HashableContainer (class in ssz.hashable\_container), 15  
 HashableContainerEvolver (class in ssz.hashable\_container), 15  
 HashableList (class in ssz.hashable\_list), 16  
 HashableStructureAPI (class in ssz.abc), 12  
 HashableStructureEvolver (class in ssz.hashable\_structure), 17  
 HashableStructureEvolverAPI (class in ssz.abc), 12  
 HashableVector (class in ssz.hashable\_vector), 18  
 hashablify\_field\_kwargs() (in module ssz.hashable\_container), 16  
 hashablify\_value() (in module ssz.hashable\_container), 16  
 HashTree (class in ssz.hash\_tree), 13  
 HashTreeEvolver (class in ssz.hash\_tree), 14  
 HomogeneousProperCompositeSedes (class in ssz.sedes.basic), 5

## I

index() (ssz.hash\_tree.HashTree method), 13  
 infer\_sedes() (in module ssz.sedes), 10  
 is\_dirty() (ssz.abc.HashableStructureEvolverAPI method), 12  
 is\_dirty() (ssz.hash\_tree.HashTreeEvolver method), 14  
 is\_dirty() (ssz.hashable\_structure.HashableStructureEvolver method), 17  
 is\_fixed\_sized (ssz.hashable\_container.MetaHashableContainer property), 16  
 is\_fixed\_sized (ssz.sedes.base.BaseSedes property), 4  
 is\_fixed\_sized (ssz.sedes.basic.BasicSedes attribute), 4  
 is\_fixed\_sized (ssz.sedes.bitlist.Bitlist attribute), 5  
 is\_fixed\_sized (ssz.sedes.bitvector.Bitvector attribute), 6  
 is\_fixed\_sized (ssz.sedes.container.Container property), 8  
 is\_fixed\_sized (ssz.sedes.list.List attribute), 8  
 is\_fixed\_sized (ssz.sedes.serializable.MetaSerializable property), 9  
 is\_fixed\_sized (ssz.sedes.vector.Vector property), 10  
 is\_immutable\_field\_value() (in module ssz.utils), 19  
 is\_packing (ssz.sedes.base.BaseProperCompositeSedes property), 4  
 is\_packing (ssz.sedes.basic.HomogeneousProperCompositeSedes property), 5  
 is\_packing (ssz.sedes.container.Container property), 8

## L

length (ssz.sedes.vector.Vector property), 10  
 List (class in ssz.sedes.list), 8

## M

make\_immutable() (in module ssz.sedes.serializable), 9  
 max\_length (ssz.hashable\_structure.BaseHashableStructure property), 17  
 merge\_args\_to\_kwargs() (in module ssz.sedes.serializable), 9  
 merge\_kwargs\_to\_args() (in module ssz.sedes.serializable), 9  
 merkleize() (in module ssz.utils), 19  
 merkleize\_with\_cache() (in module ssz.utils), 19  
 Meta (class in ssz.hashable\_container), 15  
 Meta (class in ssz.sedes.serializable), 8

MetaHashableContainer (class in *ssz.hashable\_container*), 15  
 MetaSerializable (class in *ssz.sedes.serializable*), 9  
 MetaSignedHashableContainer (class in *ssz.hashable\_container*), 16  
 MetaSignedSerializable (class in *ssz.sedes.signed\_serializable*), 9  
 mix\_in\_length() (in module *ssz.utils*), 19  
 module  
   *ssz*, 19  
   *ssz.abc*, 12  
   *ssz.cache*, 4  
   *ssz.cache.cache*, 3  
   *ssz.cache.utils*, 3  
   *ssz.codec*, 13  
   *ssz.constants*, 13  
   *ssz.exceptions*, 13  
   *ssz.hash*, 13  
   *ssz.hash\_tree*, 13  
   *ssz.hashable\_container*, 15  
   *ssz.hashable\_list*, 16  
   *ssz.hashable\_structure*, 17  
   *ssz.hashable\_vector*, 18  
   *ssz.sedes*, 10  
   *ssz.sedes.base*, 4  
   *ssz.sedes.basic*, 4  
   *ssz.sedes.bitlist*, 5  
   *ssz.sedes.bitvector*, 6  
   *ssz.sedes.boolean*, 6  
   *ssz.sedes.byte*, 6  
   *ssz.sedes.byte\_list*, 7  
   *ssz.sedes.byte\_vector*, 7  
   *ssz.sedes.container*, 7  
   *ssz.sedes.list*, 8  
   *ssz.sedes.serializable*, 8  
   *ssz.sedes.signed\_serializable*, 9  
   *ssz.sedes.uint*, 10  
   *ssz.sedes.vector*, 10  
   *ssz.tools*, 12  
   *ssz.tools.codec*, 10  
   *ssz.tools.dump*, 11  
   *ssz.tools.parse*, 11  
   *ssz.tree\_hash*, 19  
   *ssz.typing*, 19  
   *ssz.utils*, 19  
 mset() (*ssz.abc.HashableStructureAPI* method), 12  
 mset() (*ssz.hash\_tree.HashTree* method), 14  
 mset() (*ssz.hashable\_structure.BaseHashableStructure* method), 17

## N

normalize\_item\_index()  
   (*ssz.hashable\_container.HashableContainer* method), 15

## P

pack() (in module *ssz.utils*), 19  
 pack\_bits() (in module *ssz.utils*), 19  
 pack\_bytes() (in module *ssz.utils*), 19  
 pad\_hash\_tree() (in module *ssz.hash\_tree*), 14  
 pad\_zeros() (in module *ssz.utils*), 19  
 parse() (in module *ssz.tools.parse*), 11  
 parse\_bits() (in module *ssz.tools.parse*), 11  
 parse\_boolean() (in module *ssz.tools.parse*), 11  
 parse\_bytes() (in module *ssz.tools.parse*), 11  
 parse\_container() (in module *ssz.tools.parse*), 11  
 parse\_hashable() (in module *ssz.tools.parse*), 11  
 parse\_integer() (in module *ssz.tools.parse*), 11  
 parse\_list() (in module *ssz.tools.parse*), 11  
 parse\_serializable() (in module *ssz.tools.parse*), 11  
 parse\_vector() (in module *ssz.tools.parse*), 11  
 persistent() (*ssz.abc.HashableStructureEvolverAPI* method), 12  
 persistent() (*ssz.hash\_tree.HashTreeEvolver* method), 14  
 persistent() (*ssz.hashable\_structure.HashableStructureEvolver* method), 17  
 ProperCompositeSedes (class in *ssz.sedes.basic*), 5

## R

raw\_root (*ssz.abc.HashableStructureAPI* property), 12  
 raw\_root (*ssz.hashable\_structure.BaseHashableStructure* property), 17  
 read\_exact() (in module *ssz.utils*), 19  
 recompute\_hash\_in\_tree() (in module *ssz.hash\_tree*), 14  
 remove() (*ssz.hash\_tree.HashTree* method), 14  
 remove() (*ssz.hash\_tree.HashTreeEvolver* method), 14  
 reset\_cache() (*ssz.sedes.serializable.BaseSerializable* method), 8  
 ResizableHashableStructureAPI (class in *ssz.abc*), 12  
 ResizableHashableStructureEvolver (class in *ssz.hashable\_structure*), 17  
 ResizableHashableStructureEvolverAPI (class in *ssz.abc*), 12  
 root (*ssz.hash\_tree.HashTree* property), 14

## S

s\_decode\_offset() (in module *ssz.utils*), 19  
 sedes (*ssz.hashable\_structure.BaseHashableStructure* property), 17  
 Serializable (class in *ssz.sedes.serializable*), 9  
 SerializationError, 13  
 serialize() (*ssz.hashable\_container.MetaHashableContainer* method), 16  
 serialize() (*ssz.sedes.base.BaseSedes* method), 4  
 serialize() (*ssz.sedes.basic.ProperCompositeSedes* method), 5

`serialize()` (*ssz.sedes.bitlist.Bitlist method*), 5  
`serialize()` (*ssz.sedes.bitvector.Bitvector method*), 6  
`serialize()` (*ssz.sedes.boolean.Boolean method*), 6  
`serialize()` (*ssz.sedes.byte.Byte method*), 6  
`serialize()` (*ssz.sedes.byte\_list.ByteList method*), 7  
`serialize()` (*ssz.sedes.byte\_vector.ByteVector method*), 7  
`serialize()` (*ssz.sedes.container.Container method*), 8  
`serialize()` (*ssz.sedes.serializable.MetaSerializable method*), 9  
`serialize()` (*ssz.sedes.uint.UInt method*), 10  
`serialize_element_for_tree()` (*ssz.sedes.base.BaseProperCompositeSedes method*), 4  
`serialize_element_for_tree()` (*ssz.sedes.basic.ProperCompositeSedes method*), 5  
`serialize_element_for_tree()` (*ssz.sedes.byte\_list.ByteList method*), 7  
`serialize_element_for_tree()` (*ssz.sedes.byte\_vector.ByteVector method*), 7  
`set()` (*ssz.abc.HashableStructureAPI method*), 12  
`set()` (*ssz.abc.HashableStructureEvolverAPI method*), 12  
`set()` (*ssz.hash\_tree.HashTree method*), 14  
`set()` (*ssz.hash\_tree.HashTreeEvolver method*), 14  
`set()` (*ssz.hashable\_structure.BaseHashableStructure method*), 17  
`set()` (*ssz.hashable\_structure.HashableStructureEvolver method*), 17  
`set_chunk_in_tree()` (in module *ssz.hash\_tree*), 14  
`SettableFieldDescriptor` (class in *ssz.hashable\_container*), 16  
`signed_container_sedes` (*ssz.sedes.signed\_serializable.SignedMeta attribute*), 9  
`SignedHashableContainer` (class in *ssz.hashable\_container*), 16  
`SignedMeta` (class in *ssz.sedes.signed\_serializable*), 9  
`SignedSerializable` (class in *ssz.sedes.signed\_serializable*), 9  
`signing_root` (*ssz.hashable\_container.SignedHashableContainer property*), 16  
`signing_root` (*ssz.sedes.signed\_serializable.SignedSerializable property*), 10  
`size` (*ssz.sedes.byte.Byte attribute*), 6  
`ssz` module, 19  
`ssz.abc` module, 12  
`ssz.cache` module, 4  
`ssz.cache.cache` module, 3  
`ssz.cache.utils` module, 3  
`ssz.codec` module, 13  
`ssz.constants` module, 13  
`ssz.exceptions` module, 13  
`ssz.hash` module, 13  
`ssz.hash_tree` module, 13  
`ssz.hashable_container` module, 15  
`ssz.hashable_list` module, 16  
`ssz.hashable_structure` module, 17  
`ssz.hashable_vector` module, 18  
`ssz.sedes` module, 10  
`ssz.sedes.base` module, 4  
`ssz.sedes.basic` module, 4  
`ssz.sedes.bitlist` module, 5  
`ssz.sedes.bitvector` module, 6  
`ssz.sedes.boolean` module, 6  
`ssz.sedes.byte` module, 6  
`ssz.sedes.byte_list` module, 7  
`ssz.sedes.byte_vector` module, 7  
`ssz.sedes.container` module, 7  
`ssz.sedes.list` module, 8  
`ssz.sedes.serializable` module, 8  
`ssz.sedes.signed_serializable` module, 9  
`ssz.sedes.uint` module, 10  
`ssz.sedes.vector` module, 10  
`ssz.tools` module, 12  
`ssz.tools.codec`

module, 10  
 ssz.tools.dump  
     module, 11  
 ssz.tools.parse  
     module, 11  
 ssz.tree\_hash  
     module, 19  
 ssz.typing  
     module, 19  
 ssz.utils  
     module, 19  
 SSZCache (*class in ssz.cache.cache*), 3  
 SSZException, 13

## T

to\_chunks() (*in module ssz.utils*), 19  
 to\_formatted\_dict() (*in module ssz.tools.dump*), 11  
 transform() (*ssz.abc.HashableStructureAPI method*),  
     12  
 transform() (*ssz.hash\_tree.HashTree method*), 14  
 transform() (*ssz.hashable\_structure.BaseHashableStructure  
     method*), 17

## U

UInt (*class in ssz.sedes.uint*), 10  
 update\_element\_in\_chunk() (*in module  
     ssz.hashable\_structure*), 18  
 update\_elements\_in\_chunk() (*in module  
     ssz.hashable\_structure*), 18

## V

validate\_args\_and\_kwargs() (*in module  
     ssz.sedes.serializable*), 9  
 validate\_chunk\_count() (*in module ssz.hash\_tree*),  
     14  
 validate\_raw\_hash\_tree() (*in module  
     ssz.hash\_tree*), 14  
 Vector (*class in ssz.sedes.vector*), 10