

# dabmodel

---

## User Manual

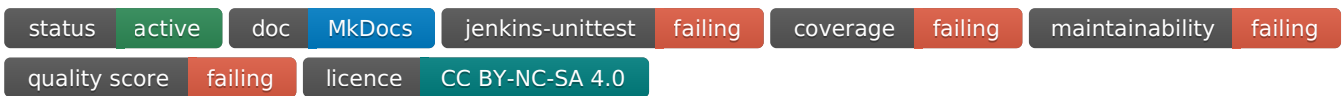
chacha

CC BY-NC-SA 4.0

## Table of contents

---

1. Python project template	3
1.1 Features	3
2. Usage	4
2.1 Pulvinar dolor	4
2.2 Condimentum faucibus	4
2.3 Aliquam lacinia	4
3. Reference	5
3.1 Dabmodel	5



# 1. Python project template

---

A nice template to start blank python projects.

This template automate a lot of handy things and allow CI/CD automatic releases generation.

It is also collectings data to feed Jenkins build.

Checkout [Latest Documentation](#).

## 1.1 Features

---

### 1.1.1 Generic pipeline skeleton:

```
- Prepare
- GetCode
- BuildPackage
- Install
- CheckCode
- PlotMetrics
- RunUnitTests
- GenDOC
- PostRelease
```

### 1.1.2 CI/CD Environment

```
- Jenkins
- Gitea (with patch for dynamic Readme variables: https://chacha.ddns.net/gitea/chacha/GiteaMarkupVariable)
- Docker
- MkDocsWeb
```

### 1.1.3 CI/CD Helper libs

```
- VirtualEnv
- Changelog generation based on commits
- copier
- pylint + pylint_json2html
- mypy
- unittest + xmlrunner + junitparser + junit2htmlreport
- mkdocs
```

### 1.1.4 Python project

```
- Full .toml implementation
- .whl automatic generation
- dynamic versionning using git repository
- embedded unit-test
```

## 2. Usage

---

### 2.1 Pulvinar dolor

---

Donec dapibus est fermentum justo volutpat condimentum. Integer quis nunc neque. Donec dictum vehicula justo, in facilisis ex tincidunt in. Vivamus sollicitudin sem dui, id mollis orci facilisis ut. Proin sed pulvinar dolor. Donec volutpat commodo urna imperdiet pulvinar. Fusce eget aliquam risus. Vivamus viverra luctus ex, in finibus mi. Nullam elementum dapibus mollis. Ut suscipit volutpat ex, quis feugiat lacus consectetur eu.

### 2.2 Condimentum faucibus

---

Quisque auctor egestas sem, luctus suscipit ex maximus vitae. Duis facilisis augue et condimentum faucibus. Donec cursus, enim a sagittis egestas, lectus lorem eleifend libero, at tincidunt leo magna at libero. Nunc eros velit, suscipit luctus tempor vel, finibus et est. Curabitur efficitur pretium pulvinar. Donec urna lectus, vulputate quis turpis sed, placerat congue urna. Phasellus aliquet fermentum quam, non auctor elit porta nec. Morbi eu ligula at nisl ultricies condimentum vitae id ante.

### 2.3 Aliquam lacinia

---

In volutpat lorem ex, et fringilla nibh faucibus quis. Mauris et arcu elementum, auctor dui vitae, egestas arcu. Duis sit amet aliquam quam. Phasellus a odio turpis. Etiam tristique mi eu enim varius, eget facilisis est vestibulum. Aliquam lacinia nec purus sed luctus. Cras at laoreet erat.

## 3. Reference

---

### 3.1 Dabmodel

---

#### 3.1.1 metadata

---

Metadata module module

##### Attributes

`__Name__` module-attribute

```
__Name__ = metadata['Name']
```

`__Summary__` module-attribute

```
__Summary__ = metadata['Summary']
```

`__version__` module-attribute

```
__version__ = version('dabmodel')
```

`dist` module-attribute

```
dist = distribution('dabmodel')
```

## 3.1.2 Appliance

---

### Classes

Appliance

Bases: [IAppliance](#)

BaseFeature class Base class for Appliance. An appliance is a server configuration / image that is built using appliance's code and Fields.

Functions

validate\_schema

```
validate_schema()
```

validate\_schema\_class classmethod

```
validate_schema_class()
```

### 3.1.3 Base element

#### Classes

BaseElement

Attributes

`__lam_class_mutable__` class-attribute instance-attribute

```
__lam_class_mutable__ = False
```

`__lam_initialized__` class-attribute instance-attribute

```
__lam_initialized__ = False
```

`__lam_object_mutable__` class-attribute instance-attribute

```
__lam_object_mutable__ = False
```

`__lam_options__` class-attribute instance-attribute

```
__lam_options__ = {}
```

```
def get_model_spec(self, name: str, module: str, memo: dict[str, Any]) -> ElementView: # memo[self.name] = {} init_fieldvalues = {} init_fielddtypes = {} for k, v in self.lam_schema.items(): if isinstance(v, LAMField_Element): memo[k] = {} init_fieldvalues[k] = v.value.get_model_spec(memo[k]) # clone = v.clone_unfrozen().value elif isinstance(v, LAMField): clone = deepcopy(v.value) init_fieldvalues[k] = clone else: pass init_fielddtypes[k] = v.annotations return ElementView(init_fieldvalues, init_fielddtypes)
```

```
@classmethod def get_model_spec_cls(cls, memo: dict[str, Any]) -> ElementView: # memo[self.name] = {} init_fieldvalues = {} init_fielddtypes = {} for k, v in cls.lam_schema.items(): if isinstance(v, LAMField_Element): memo[k] = {} init_fieldvalues[k] = v.value.get_model_spec(k, memo[k]) # clone = v.clone_unfrozen().value elif isinstance(v, LAMField): clone = deepcopy(v.value) init_fieldvalues[k] = clone else: pass init_fielddtypes[k] = v.annotations return ElementViewCls(init_fieldvalues, init_fielddtypes, cls.name, cls.modules)
```

`__lam_schema__` class-attribute instance-attribute

```
__lam_schema__ = {}
```

`frozen` property

```
frozen: bool
```

`frozen_cls` classmethod property

```
frozen_cls: bool
```

`mutable_obj` classmethod property

```
mutable_obj: bool
```

Functions

`__setattr__`

```
__setattr__(key: str, value: Any) -> None
```

`clone_as_mutable_variant`

```
clone_as_mutable_variant(*, deep: bool = True, _memo: Dict[int, Self] | None = None) -> Self
```

`freeze`

```
freeze(force: bool = False) -> None
```

`freeze_class` classmethod

```
freeze_class(force: bool = False) -> None
```

`validate_schema`

```
validate_schema() -> None
```

`validate_schema_class` classmethod

```
validate_schema_class() -> None
```

## Functions

## 3.1.4 Defines

### Attributes

ALLOWED\_ANNOTATIONS module-attribute

```
ALLOWED_ANNOTATIONS: dict[str, Any] = {'Union': Union, 'Optional': Optional, 'List': List, 'Dict': Dict, 'Tuple': Tuple, 'Set': Set, 'FrozenSet': FrozenSet, 'Annotated': Annotated, 'int': int, 'str': str, 'float': float, 'bool': bool, 'complex': complex, 'bytes': bytes, 'None': type(None), 'list': list, 'dict': dict, 'set': set, 'frozenset': frozenset, 'tuple': tuple}
```

ALLOWED\_HELPERS\_DEFAULT module-attribute

```
ALLOWED_HELPERS_DEFAULT: dict[str, object] = {'math': ALLOWED_HELPERS_MATH, 'print': print, 'abs': abs, 'round': round, 'min': min, 'max': max, 'sum': sum, 'len': len, 'sorted': sorted, 'tuple': tuple, 'list': list, 'dict': dict, 'set': set, 'int': int, 'float': float, 'str': str, 'bool': bool, 'bytes': bytes, 'complex': complex, 'range': range}
```

ALLOWED\_HELPERS\_MATH module-attribute

```
ALLOWED_HELPERS_MATH = SimpleNamespace(sqrt=sqrt, floor=floor, ceil=ceil, trunc=trunc, fabs=fabs, copysign=copysign, hypot=hypot, exp=exp, log=log, log10=log10, sin=sin, cos=cos, tan=tan, atan2=atan2, radians=radians, degrees=degrees)
```

ALLOWED\_MODEL\_FIELDS\_TYPES module-attribute

```
ALLOWED_MODEL_FIELDS_TYPES: tuple[type[Any], ...] = (str, int, float, complex, bool, bytes)
```

JSONPrimitive module-attribute

```
JSONPrimitive = Union[str, int, float, bool, None]
```

JSONType module-attribute

```
JSONType = Union[JSONPrimitive, List[Any], Dict[str, Any]]
```

## 3.1.5 Element

---

### Classes

Element

Bases: [IElement](#)

Element class Base class to apply metaclass and set common Fields.

### 3.1.6 Exception

---

#### Classes

BrokenInheritance

Bases: [DABModelException](#)

BrokenInheritance Exception class inheritance chain is broken

ClosureForbidden

Bases: [FunctionForbidden](#)

ClosureForbidden Exception class

DABModelException

Bases: Exception

DABModelException Exception class Base Exception for DABModelException class

ExternalCodeForbidden

Bases: [FunctionForbidden](#)

ExternalCodeForbidden Exception class

FeatureAlreadyBound

Bases: [DABModelException](#)

FeatureAlreadyBound Exception class Feature can only be bind once

FeatureBoundToIncompatibleAppliance

Bases: [DABModelException](#)

FeatureBoundToWrongAppliance Exception class Feature have to be bound to correct appliance

FeatureBoundToNonAppliance

Bases: [DABModelException](#)

FeatureBoundToNonAppliance Exception class Feature can only be bind to Appliance class

FeatureNotBound

Bases: [DABModelException](#)

FeatureNotBound Exception class a Feature must be bound to the appliance (or parent)

FeatureWrongBound

Bases: [DABModelException](#)

FeatureWrongBound Exception class Feature can be bind to one and only one Appliance

FunctionForbidden

Bases: [DABModelException](#)

FunctionForbidden Exception class

ImportForbidden

Bases: [DABModelException](#)

ImportForbidden Exception class Imports are forbidden

IncompletelyAnnotatedField

Bases: [InvalidFieldAnnotation](#)

IncompletelyAnnotatedField Exception class The field annotation is incomplete

InvalidFeatureInheritance

Bases: [DABModelException](#)

InvalidFeatureInheritance Exception class Features of same name in child appliance need to be from same type

InvalidFieldAnnotation

Bases: [AttributeError](#), [DABModelException](#)

InvalidFieldAnnotation Exception class The field annotation is invalid

InvalidFieldName

Bases: [AttributeError](#), [DABModelException](#)

InvalidFieldName Exception class The Field name is invalid

InvalidFieldValue

Bases: [AttributeError](#), [DABModelException](#)

InvalidFieldValue Exception class The Field value is invalid

InvalidInitializerType

Bases: [DABModelException](#)

InvalidInitializerType Exception class The initializer is not a valid type

MultipleInheritanceForbidden

Bases: [DABModelException](#)

MultipleInheritanceForbidden Exception class Multiple inheritance is forbidden when using dabmodel

NonExistingField

Bases: [AttributeError](#), [DABModelException](#)

NonExistingField Exception class The given Field is non existing

NotAnnotatedField

Bases: [InvalidFieldAnnotation](#)

NotAnnotatedField Exception class The Field is not Annotated

ReadOnlyField

Bases: [AttributeError](#), [DABModelException](#)

ReadOnlyField Exception class The used Field is ReadOnly

ReadOnlyFieldAnnotation

Bases: [AttributeError](#), [DABModelException](#)

ReadOnlyFieldAnnotation Exception class Field annotation cannot be modified

ReservedFieldName

Bases: [AttributeError](#), [DABModelException](#)

ReservedFieldName Exception class Base Exception for DABModelException class

SchemaViolation

Bases: [AttributeError](#), [DABModelException](#)

SchemaViolation Exception class The Element Schema is not respected

UnsupportedFieldType

Bases: [InvalidFieldAnnotation](#)

UnsupportedFieldType Exception class The field type is unsupported

WrongUsage

Bases: [DABModelException](#), [RuntimeError](#)

## 3.1.7 Feature

---

### Classes

Feature

Bases: [IFeature](#)

Feature class Base class for Appliance's Features. Features are optional traits of an appliance.

Attributes

**Enabled** class-attribute instance-attribute

```
Enabled: bool = False
```

**\_\_lam\_bound\_appliance\_\_** class-attribute instance-attribute

```
__lam_bound_appliance__ = None
```

Functions

**bind\_appliance** classmethod

```
bind_appliance(appliance_cls)
```

**check\_appliance\_bound** classmethod

```
check_appliance_bound()
```

**check\_appliance\_compatibility** classmethod

```
check_appliance_compatibility(appliance_cls)
```

## 3.1.8 Interfaces

---

### Attributes

`TV_Freezable` module-attribute

```
TV_Freezable = TypeVar('TV_Freezable')
```

### Classes

`FreezableElement`

Bases: `Protocol`, `Generic[TV_Freezable]`

Attributes

`frozen` property

```
frozen: bool
```

`frozen_cls` classmethod property

```
frozen_cls: bool
```

`mutable_obj` classmethod property

```
mutable_obj: bool
```

Functions

`clone_as_mutable_variant`

```
clone_as_mutable_variant(*, deep: bool = True, _memo: Dict[int, TV_Freezable] | None = None) -> TV_Freezable
```

`freeze`

```
freeze(force: bool = False) -> None
```

`freeze_class` classmethod

```
freeze_class(force: bool = False) -> None
```

`validate_schema`

```
validate_schema() -> None
```

`validate_schema_class` classmethod

```
validate_schema_class() -> None
```

## 3.1.9 Tools

---

library's internal tools

### Attributes

### Classes

LAMJSONEncoder

Bases: `JSONEncoder`

allows to JSON encode non supported data type

Functions

default

```
default(o)
```

### Functions

LAMdeepfreeze

```
LAMdeepfreeze(value)
```

recursive freeze helper function

is\_data\_attribute

```
is_data_attribute(name: str, value: any) -> bool
```

## 3.1.10 Lam field

---

### Constraint

#### ATTRIBUTES

`T_Field` module-attribute

```
T_Field = TypeVar('T_Field')
```

#### CLASSES

Constraint

```
Constraint()
```

Bases: `Generic[T_Field]`

Constraint class Base class for Field's constraints

#### Functions

check

```
check(value: T_Field) -> bool
```

Check if a Constraint is completed

## Lam field

### ATTRIBUTES

TV\_LABField module-attribute

```
TV_LABField = TypeVar('TV_LABField')
```

### CLASSES

LAMField

```
LAMField(name: str, val: Optional[TV_LABField], a: Any, i: LAMFieldInfo)
```

Bases: `Generic[TV_LABField]`

This class describe a Field in Schema

### Attributes

`__annotations` instance-attribute

```
__annotations: Any = LAMdeepfreeze(a)
```

`__frozen` instance-attribute

```
__frozen: bool = False
```

`__frozen_value_set` instance-attribute

```
__frozen_value_set: True = False
```

`__info` instance-attribute

```
__info: LAMFieldInfo = deepcopy(i)
```

`__name` instance-attribute

```
__name: str = name
```

`__source` instance-attribute

```
__source: Optional[type] = None
```

`annotations` property

```
annotations: Any
```

Returns Field's annotation

`constraints` property

```
constraints: list[Constraint]
```

Returns Field's constraint

`default_value` property

```
default_value: Any
```

Returns Field's default value (frozen)

`doc` property

```
doc: str
```

Returns Field's documentation

**frozen\_value** property

```
frozen_value: Any
```

**info** property

```
info: LAMFieldInfo
```

Returns Field's info

**name** property

```
name: str
```

**raw\_value** property

```
raw_value: Optional[TV_LABField]
```

Returns Field's value

**value** property

```
value: Any
```

Returns Field's value (frozen)

Functions

**add\_constraint**

```
add_constraint(cons: Constraint) -> None
```

Adds constraint to the Field

**add\_source**

```
add_source(src: type) -> None
```

Adds source Appliance to the Field

**clone\_unfrozen**

```
clone_unfrozen() -> Self
```

**freeze**

```
freeze()
```

**is\_frozen**

```
is_frozen() -> bool
```

**update\_value**

```
update_value(val: Optional[TV_LABField] = None) -> None
```

Updates Field's value

**validate**

```
validate(val: Optional[TV_LABField])
```

**validate\_self**

```
validate_self()
```

LAMFieldFactory

Functions

`create_field` staticmethod

```
create_field(name: str, val: Optional[TV_LABField], anno: Any, info: LAMFieldInfo) -> LAMField
```

`LAMField_Element`

```
LAMField_Element(name: str, val: Optional[TV_LABField], a: Any, i: LAMFieldInfo)
```

Bases: `LAMField[FreezableElement]`

Attributes

`default_value` property

```
default_value: Any
```

Functions

`update_value`

```
update_value(val: Optional[FreezableElement] = None) -> None
```

`validate`

```
validate(val: Optional[FreezableElement])
```

FUNCTIONS

**Lam field info**

## CLASSES

## LAMFieldInfo

```
LAMFieldInfo(*, doc: str = '', constraints: Optional[List[Constraint]] = None)
```

This Class allows to describe a Field in Appliance class

## Attributes

`__constraints` instance-attribute

```
__constraints: list[Constraint]
```

`__doc` instance-attribute

```
__doc: str = doc
```

`constraints` property

```
constraints: list[Constraint[Any]]
```

Returns Field's constraints

`doc` property

```
doc: str
```

Returns Field's documentation

## Functions

`add_constraint`

```
add_constraint(constraint: Constraint)
```

## 3.1.11 Meta

---

### Appliance

CLASSES

FUNCTIONS

## Element

ATTRIBUTES

`T_BE` module-attribute

```
T_BE = TypeVar('T_BE', bound='BaseElement')
```

`T_Meta` module-attribute

```
T_Meta = TypeVar('T_Meta', bound='_MetaElement')
```

CLASSES

ClassMutable

Bases: [ElementOptions](#)

ElementOptions

IAppliance

Bases: [IBaseElement](#)

IBaseElement

Bases: [BaseElement](#)

Functions

`clone_as_mutable_variant`

```
clone_as_mutable_variant(*, deep: bool = True, _memo: Dict[int, BaseElement] | None = None) -> BaseElement
```

IElement

Bases: [IBaseElement](#)

IFeature

Bases: [IBaseElement](#)

IMutableVariant

ModelSpecView

```
ModelSpecView(values: dict[str, Any], types_map: dict[str, type], name: str, module: str)
```

ModelSpecView class A class that will act as fake BaseElement proxy to allow setting values

Attributes

`__module__` property writable

```
__module__: str
```

returns proxified module's name

`__name__` property

```
__name__: str
```

returns proxified class' name

`__slots__` class-attribute instance-attribute

```
__slots__ = ('_vals', '_types', '_touched', '_name', '_module')
```

Functions

`__getattr__`

```
__getattr__(name: str) -> Any
```

internal proxy getattr

```
__setattr__
```

```
__setattr__(name: str, value: Any)
```

internal proxy setattr

export

```
export() -> dict
```

exports all proxified values

ObjectMutable

Bases: [ElementOptions](#)

FUNCTIONS

get\_mutable\_variant

```
get_mutable_variant(base: Type[BaseElement]) -> Type[BaseElement]
```

Return a subclass of `base` that behaves the same except that instances are created object-mutable (because the class was defined with `options=(ObjectMutable,)`).

**Feature**