Craft Grammar

Canonical Group Ltd

May 02, 2024

GETTING STARTED

CHAPTER

ONE

CRAFT_GRAMMAR PACKAGE

1.1 Submodules

1.1.1 craft_grammar.create module

Utilities to create grammar models.

craft_grammar.create.**create_grammar_model**(*model_class: type[BaseModel]*) → str

Create the code for a grammar-aware class compatible with `model_class`.

Parameters

model_class – A pydantic.BaseModel subclass.

1.1.2 craft_grammar.errors module

Errors for Craft Grammar.

exception craft_grammar.errors.**CraftGrammarError**

Bases: `Exception`

Base class error for craft-grammar.

exception craft_grammar.errors.**GrammarSyntaxError**(*message: str*)

Bases: *CraftGrammarError*

Error raised on grammar syntax errors.

exception craft_grammar.errors.**OnStatementSyntaxError**(*on_statement: str*, ***, *message: str | None = None*)

Bases: *GrammarSyntaxError*

Error raised on on statement syntax errors.

exception craft_grammar.errors.**ToStatementSyntaxError**(*to_statement: str*, ***, *message: str | None = None*)

Bases: *GrammarSyntaxError*

Error raised on to statement syntax errors.

exception craft_grammar.errors.**UnsatisfiedStatementError**(*statement: str*)

Bases: *CraftGrammarError*

Error raised when a statement cannot be satisfied.

1.1.3 craft_grammar.models module

Pydantic models for grammar.

class craft_grammar.models.**Grammar**

Bases: Generic[T]

Grammar aware type.

Allows to use Grammar[T] to define a grammar-aware type.

Grammar[int] Grammar[list[str]] Grammar[dict[str, int]]

class craft_grammar.models.**GrammarMetaClass**

Bases: type

Grammar type metaclass.

Allows to use GrammarType[T] to define a grammar-aware type.

1.2 Module contents

Enhance part definitions with advanced grammar.

class craft_grammar.**CompoundStatement**(*, *statements: list[Statement]*, *body: Sequence[str | dict[str, Any]]*, *processor: BaseProcessor*, *call_stack: list[Statement] | None = None*)

Bases: *Statement*

Multiple statements that need to be treated as a group.

check() → bool

Check if a statement main body should be processed.

Returns

True if main body should be processed, False if elses should be processed.

class craft_grammar.**GrammarProcessor**(*, *checker: Callable[[Any], bool]*, *arch: str*, *target_arch: str*, *transformer: Callable[[list[Statement], str, str], str] | None = None*)

Bases: BaseProcessor

The GrammarProcessor extracts desired primitives from grammar.

process(*, *grammar: Sequence[str | dict[str, Any]]*, *call_stack: list[Statement] | None = None*) → list[Any]

Process grammar and extract desired primitives.

Parameters

- **grammar** – Unprocessed grammar.
- **call_stack** – Call stack of statements leading to now.

Returns

Primitives selected

class craft_grammar.**OnStatement**(*, *on_statement: str*, *body: Sequence[str | dict[str, Any]]*, *processor: BaseProcessor*, *call_stack: list[Statement] | None = None*)

Bases: *Statement*

Process an 'on' statement in the grammar.

check() → bool

Check if a statement main body should be processed.

Returns
True if main body should be processed, False if elses should be processed.

class craft_grammar.**Statement**(*, *body: Sequence[str | dict[str, Any]]*, *processor: BaseProcessor*, *call_stack: list[Statement] | None*, *check_primitives: bool = False*)

Bases: object

Base class for all grammar statements.

add_else(*else_body: Sequence[str | dict[str, Any]] | None*) → None

Add an 'else' clause to the statement.

Parameters
else_body (*list*) – The body of an 'else' clause.

The 'else' clauses will be processed in the order they are added.

abstract check() → bool

Check if a statement main body should be processed.

Returns
True if main body should be processed, False if elses should be processed.

process() → list[str]

Process this statement.

Returns
Primitives as determined by evaluating the statement or its else clauses.

class craft_grammar.**ToStatement**(*, *to_statement: str*, *body: Sequence[str | dict[str, Any]]*, *processor: BaseProcessor*, *call_stack: list[Statement] | None = None*)

Bases: *Statement*

Process a 'to' statement in the grammar.

check() → bool

Check if a statement main body should be processed.

Returns
True if main body should be processed, False if elses should be processed.

class craft_grammar.**TryStatement**(*, *body: Sequence[str | dict[str, Any]]*, *processor: BaseProcessor*, *call_stack: list[Statement] | None = None*)

Bases: *Statement*

Process a 'try' statement in the grammar.

For example: >>> from snapcraft_legacy import ProjectOptions >>> from ._processor import GrammarProcessor >>> def checker(primitive): … return 'invalid' not in primitive >>> options = ProjectOptions() >>> processor = GrammarProcessor(None, options, checker) >>> clause = TryStatement(body=['invalid'], processor=processor) >>> clause.add_else(['valid']) >>> clause.process() {'valid'}

check() → bool

Check if a statement main body should be processed.

Returns
True if main body should be processed, False if elses should be processed.

craft_grammar.**create_grammar_model**(*model_class: type[BaseModel]*) → str

Create the code for a grammar-aware class compatible with `model_class`.

Parameters

model_class – A pydantic.BaseModel subclass.

CHAPTER

TWO

CHANGELOG

2.1 1.2.0 (2024-04-05)

- Add more grammar types

2.2 1.1.2 (2023-11-30)

- Include type information

2.3 1.1.1 (2022-02-28)

- Fix models for grammar validation

2.4 1.1.0 (2022-02-24)

- Introduce grammar aware Pydantic Models that deprecate the use of try

2.5 1.0.0 (2022-02-16)

- Initial import from Snapcraft
- Initial packaging
- Code updated to follow latest development practices

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX