Craft Store

Release 1.2.0

Canonical Ltd.

Apr 05, 2024

GETTING STARTED

CHAPTER

ONE

CRAFT_GRAMMAR PACKAGE

1.1 Submodules

1.1.1 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*)
Bases: *craft_grammar.errors.CraftGrammarError*

Error raised on grammar syntax errors.

Parameters **message** (str) –

exception craft_grammar.errors.**OnStatementSyntaxError**(*on_statement*, *, *message=None*)
Bases: *craft_grammar.errors.GrammarSyntaxError*

Error raised on on statement syntax errors.

Parameters

- **on_statement** (str) –
- **message** (Optional[str]) –

exception craft_grammar.errors.**ToStatementSyntaxError**(*to_statement*, *, *message=None*)
Bases: *craft_grammar.errors.GrammarSyntaxError*

Error raised on to statement syntax errors.

Parameters

- **to_statement** (str) –
- **message** (Optional[str]) –

exception craft_grammar.errors.**UnsatisfiedStatementError**(*statement*)
Bases: *craft_grammar.errors.CraftGrammarError*

Error raised when a statement cannot be satisfied.

Parameters **statement** (str) –

1.1.2 craft_grammar.models module

Pydantic models for grammar.

class craft_grammar.models.**GrammarBool**
Bases: craft_grammar.models._GrammarBase

Grammar-enabled bool field.

classmethod validate(*entry*)
Ensure the given entry is valid type or grammar.

class craft_grammar.models.**GrammarDict**
Bases: craft_grammar.models._GrammarBase

Grammar-enabled dictionary field.

classmethod validate(*entry*)
Ensure the given entry is valid type or grammar.

class craft_grammar.models.**GrammarDictList**
Bases: craft_grammar.models._GrammarBase

Grammar-enabled list of dictionary field.

classmethod validate(*entry*)
Ensure the given entry is valid type or grammar.

class craft_grammar.models.**GrammarFloat**
Bases: craft_grammar.models._GrammarBase

Grammar-enabled float field.

classmethod validate(*entry*)
Ensure the given entry is valid type or grammar.

class craft_grammar.models.**GrammarInt**
Bases: craft_grammar.models._GrammarBase

Grammar-enabled integer field.

classmethod validate(*entry*)
Ensure the given entry is valid type or grammar.

class craft_grammar.models.**GrammarSingleEntryDictList**
Bases: craft_grammar.models._GrammarBase

Grammar-enabled list of dictionaries field.

classmethod validate(*entry*)
Ensure the given entry is valid type or grammar.

class craft_grammar.models.**GrammarStr**
Bases: craft_grammar.models._GrammarBase

Grammar-enabled string field.

classmethod validate(*entry*)
Ensure the given entry is valid type or grammar.

class craft_grammar.models.**GrammarStrList**
Bases: craft_grammar.models._GrammarBase

Grammar-enabled list of strings field.

classmethod validate(*entry*)
Ensure the given entry is valid type or grammar.

1.2 Module contents

Enhance part definitions with advanced grammar.

class craft_grammar.**CompoundStatement**(*, *statements*, *body*, *processor*, *call_stack=None*)
Bases: *craft_grammar._statement.Statement*

Multiple statements that need to be treated as a group.

Parameters

- **statements** (List[*Statement*]) –
- **body** (Sequence[Union[str, Dict[str, Any]]]) –
- **processor** (GrammarProcessor) –
- **call_stack** (Optional[List[ForwardRef]]) –

check()
Check if a statement main body should be processed.

Return type bool

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

class craft_grammar.**GrammarProcessor**(*, *checker*, *arch*, *target_arch*, *transformer=None*)
Bases: object

The GrammarProcessor extracts desired primitives from grammar.

Parameters

- **checker** (Callable[[Any], bool]) –
- **arch** (str) –
- **target_arch** (str) –
- **transformer** (Optional[Callable[[List[*Statement*], str, str], str]]) –

process(*, *grammar*, *call_stack=None*)
Process grammar and extract desired primitives.

Parameters

- **grammar** (Sequence[Union[str, Dict[str, Any]]]) – Unprocessed grammar.
- **call_stack** (Optional[List[*Statement*]]) – Call stack of statements leading to now.

Return type List[Any]

Returns Primitives selected

class craft_grammar.**OnStatement**(*, *on_statement*, *body*, *processor*, *call_stack=None*)
Bases: *craft_grammar._statement.Statement*

Process an 'on' statement in the grammar.

Parameters

- **on_statement** (str) –

- **body** (Sequence[Union[str, Dict[str, Any]]]) –
- **processor** (GrammarProcessor) –
- **call_stack** (Optional[List[ForwardRef]]) –

check()
Check if a statement main body should be processed.

Return type bool

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

class craft_grammar.**Statement**(*, *body*, *processor*, *call_stack*, *check_primitives=False*)
Bases: object

Base class for all grammar statements.

Parameters

- **body** (Sequence[Union[str, Dict[str, Any]]]) –
- **processor** (GrammarProcessor) –
- **call_stack** (Optional[List[ForwardRef]]) –
- **check_primitives** (bool) –

add_else(*else_body*)
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.

Parameters **else_body** (Optional[Sequence[Union[str, Dict[str, Any]]]]) –

Return type None

abstract check()
Check if a statement main body should be processed.

Return type bool

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

process()
Process this statement.

Return type List[str]

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

class craft_grammar.**ToStatement**(*, *to_statement*, *body*, *processor*, *call_stack=None*)
Bases: *craft_grammar._statement.Statement*

Process a 'to' statement in the grammar.

Parameters

- **to_statement** (str) –
- **body** (Sequence[Union[str, Dict[str, Any]]]) –
- **processor** (GrammarProcessor) –
- **call_stack** (Optional[List[ForwardRef]]) –

check()
Check if a statement main body should be processed.

Return type bool

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

class craft_grammar.**TryStatement**(*, *body*, *processor*, *call_stack=None*)
Bases: *craft_grammar._statement.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'}

Parameters

- **body** (Sequence[Union[str, Dict[str, Any]]]) –
- **processor** (GrammarProcessor) –
- **call_stack** (Optional[List[ForwardRef]]) –

check()
Check if a statement main body should be processed.

Return type bool

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

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