pcs_argpass

Release 1.15

Ing. Rainer Pietsch

Apr 04, 2023

CONTENTS

Document version: 1.15.339.230404114747

Note: This project is under active development.

CHAPTER

ONE

SOURCE

You can get the complete source-code from GitHub repository

CHAPTER

TWO

DETAILED DOCUMENTATION

The detailed documentation is on Read the Docs

CHAPTER

THREE

TABLE OF CONTENTS

3.1 Usage

3.1.1 Installation

To use pcs_argpass, first install it using pip:

```
(.venv) $ pip install pcs_argpass
```

3.1.2 Use in your program

This module handles the most often used command-line parameter types:

- boolean switch
- integer
- float
- file
- dir
- path
- text
- counter

additionally this module handles the generation and display of help-messages and licence informations. Another functionallity is the export of parameters and the import of settings.

normally imported as

```
from pcs_argpass.Param import Param, Translation_de_DE
```

This class can be used to create recursive sub-parameter classes. All children inherit the settings of their parents.

Check out *Examples*

3.2 Param reference

Deals with command-line parameters

Detailed docs at <https://argpass.readthedocs.io/en/latest/index.html>

Source at <https://github.com/rpietsch1953/Argpass>

class Param.**Param**(*, *Def: dict = {}*, *Args: Optional[list] = None*, *Chk=None*, *Desc: str = ''*, *AddPar: str = ''*, *AllParams: bool = True*, *UserPars: Optional[dict] = None*, *UserModes: Optional[dict] = None*, *ErrorOnUnknown: bool = True*, *HelpType: int = 0*, *Children: dict = {}*, *translation: dict = {}*, *Version: str = ''*, *License: Optional[Union[str, tuple, list, dict]] = None*, *ShowPrefixOnHelp: bool = True*, *ShowConfigName: bool = False*, *_Child=False*)

Main class and also the result-dictionary. A member of this class acts like a dictionary. There are some special cases if you use nested childs. (Check out the *Usage* section for further information)

AddChild(*Prefix: str*, *Def: dict = {}*, *Description: str = ''*, *Children: dict = {}*, *Version: str = ''*, *License: list = ['']*, *AddPar: str = ''*) → None

Add a child to a instance

Parameters

- **Def** (*dict, optional*) – Definition for this instance (look at *SetDef()* for details), defaults to { }
- **Description** (*str, optional*) – Description of this instance, defaults to ''
- **Children** (*dict, optional*) – Dictionary of children to the new instance, defaults to { }
- **Version** (*str, optional*) – A version string, defaults to ''
- **AddPar** (*str, optional*) – Additional parameter string of this instance, defaults to ''

Raises

self.DeclarationError – If a parameter is invalid

property Child: Dict[str, object]

Child dict

Returns

return all the children of this instance

Return type

Dict[str, *Param*]

exception DeclarationError

this exception is raised if there is an declaration error within the parameters of the class.

Note: This error messages are NEVER translated since they are not user initiated errors.

property Definition: dict

Returns s copy of the definition

Returns
a definition dictionary

Return type
dict

property FreeShortCommandLineParameter: str

Helper for programmers. Can be used before "Process". Should not be used in production environment

Returns
A formatted, sorted list of all unused (free to use) short options broken to 68 characters a line.

Return type
str

property FullPrefix: str

Returns the full qualified prefix of this instance e.g.: global.alpha.gamma if alpha is a child of global and gamma (this instance) is a child of alpha

Returns
full qualified prefix of this instance

Return type
str

GetCmdPar(*Entry: str*, *dotted: bool = False*, *parents: bool = False*) → str

Return the commandline-options for one entry

Parameters

- **Entry** (*str*) – The entry we are looking for
- **dotted** (*bool, optional*) – show prefix for long params, defaults to False
- **parents** (*bool, optional*) – show also options from parents, defaults to False

Returns
the command-line options for this entry. E.g. "-h, –help"

Return type
str

property GetExportDict

Return the dictionary for exporting all parameters

Returns:
dict: The complete parameter dictionary

GetRemainder() → list

Return list of additionel arguments on command-line

Returns:
list: List of additional arguments within runtime-arguments

exception GetoptError(*msg*, *opt=''*)

Own error-class for option-errors

__init__(*msg*, *opt=''*)

`IsInherited`(*key: str*) → bool

Check if key is from parent

Parameters
key (*str*) – Key to search for

Returns
True if key is inherited from parent

Return type
bool

`IsOwnKey`(*key: str*) → bool

Check if the key is from the own optionset

Parameters
key (*str*) – Key to search for

Returns
True if key is in the own optionset

Return type
bool

`IsValidIp`(*Ip: str*) → Optional[str]

Test if this Ip (or DNS-name) is a valid IP address (either IPV4 or IPV6)

Parameters
Ip (*str*) – The IP-adress or DNS-name

Returns
The (purified) IP or None if this address is not a valid IP address

Return type
Union[str, None]

`IsValidIp4`(*Ip: str*) → Optional[str]

Test if this Ip (or DNS-name) is a valid IPV4 address

Parameters
Ip (*str*) – The IP-adress or DNS-name

Returns
The (purified) IP or None if this address is not a valid IPV4 address

Return type
Union[str, None]

`IsValidIp6`(*Ip: str*) → Optional[str]

Test if this Ip (or DNS-name) is a valid IPV6 address

Parameters
Ip (*str*) – The IP-adress or DNS-name

Returns
The (purified) IP or None if this address is not a valid IPV6 address

Return type
Union[str, None]

`IsValidLocalIp`(*Ip: str*) → Optional[str]

Test if this IPV4 or IPV6 (or DNS-name) is on this computer

Parameters
Ip (*str*) – The IP-adress or DNS-name

Returns
The (purified) IP or None if this address is not on this computer

Return type
Union[str, None]

IsValidLocalIp4(*Ip: str*) → Optional[str]

Test if this Ip (or DNS-name) is on this computer

Parameters
Ip (*str*) – The IP-adress or DNS-name

Returns
The (purified) IP or None if this address is not on this computer

Return type
Union[str, None]

IsValidLocalIp6(*Ip: str*) → Optional[str]

Test if this IV6 (or DNS-name) is on this computer

Parameters
Ip (*str*) – The IPV6-adress or DNS-name

Returns
The (purified) IP or None if this address is not on this computer

Return type
Union[str, None]

property LongOptsList: list

Return copied list of long options

Returns:
list: List of long options

MyProgName() → str

Return the program-name

Returns
Name of the executeable

Return type
str

MyProgPath() → str

Return the program-path

Returns
Path of the directory where executeable resides

Return type
str

MyPwd() → str

Return the directory at invocation of "Process"

Returns
Current directory at the time "Process" was called

Return type
str

property OverviewCommandLineParameter: str

Helper for programmers. can be used before "Process". Should not be used in production environment. Give information about used long and short options and all unused (=availlable) short options.

Returns
A formatted string giving all information about command-line parameters, broken to 68 characters a line.

Return type
str

property OwnIpAddresses: list

Return a list of all IP addresses on this computer at this time (either IPV4 or IPV6)

Returns
List of IP-addresses

Return type
list[str]

property ParDict: dict

Return copied dict with references options -> parameter-names

Returns:
dict: {option: name, ...}

exception ParamError

This exception is raised if there is an error within the runtime-parameters. This is only raised within the *Param.Process()*-function.

Note: This errors are translated with the 'translation' dictionary. There is the initial state of this dict using 'en_US' texts.

This module also provide a 'Translation_de_DE' entry giving a german translation of the error-messages, and a 'Translation_en_US' table only vor completeness.

If you can provide translations to other languages send me this declarations to <r.pietsch@pcs-at.com> and I will add them to this module.

ParamStr(*indent: int = 4*, *header: bool = True*, *allvalues: bool = True*, *dotted: bool = False*, *cmdpar: bool = True*, *parentopts: bool = False*, *recursive: bool = True*) → str

Returns a string with formatted output of the processed parameters.

Parameters

- **indent** (*int, optional*) – Number of leading spaces for children. Defaults to 4. this value is multiplied with the generation. So grandchildren have two times this number of leading spaces and children only one time this number of spaces.
- **header** (*bool, optional*) – If True a header with the name of the object are added, defaults to True
- **allvalues** (*bool, optional*) – Outputs all avallable options for this child, included the inherited options. Defaults to True
- **dotted** (*bool, optional*) – If True the names of the parameters are prefixed with the names of their parents, defaults to False

- **cmdpar** (*bool, optional*) – If True the commandline-options ere included in the output, defaults to True
- **parentopts** (*bool, optional*) – If True and cmdpar is also True the commandline-options of the parents are anso included in the output, defaults to False
- **recursive** (*bool, optional*) – If True all descendants are include in the output, else only the own parameters are included, defaults to True

Returns
The formated string of the processed parameters

Return type
str

Examples:

```
Assuming:
    the topmost level includes
        "NoDaemon", "Quiet", "StdErr", and "Verbose"
    child "alpha" includes
        "Count", "Text" and "Verbose"
    grandchild "alpha -> gamma" includes
        "Xy"
    child "beta" includes
        "Verbose"

The largest format is like:


------------------------------------------------------------
global
------------------------------------------------------------
global                      -> NoDaemon (-d, --[global.]nodaemon)              :␣
↪False
global                      -> Quiet    (-q, --[global.]quiet)                 :␣
↪False
global                      -> StdErr   (-s, --[global.]console)               :␣
↪False
global                      -> Verbose  (-v, --[global.]verbose)               :␣
↪2
    ------------------------------------------------------------
    global.alpha
    ------------------------------------------------------------
    global.alpha            -> Count    (-c, --[alpha.]count, --[alpha.]Count) :␣
↪7
    global.alpha            -> NoDaemon (-d, --[global.]nodaemon)              :␣
↪False
    global.alpha            -> Quiet    (-q, --[global.]quiet)                 :␣
↪False
    global.alpha            -> StdErr   (-s, --[global.]console)               :␣
↪False
    global.alpha            -> Text     (-t, --[alpha.]text, --[alpha.]Text)   :
↪''
    global.alpha            -> Verbose  (-v, --[alpha.]verbose)                :␣
↪2
```

(continues on next page)

(continued from previous page)

```
        ------------------------------------------------------------
        global.alpha.gamma
        ------------------------------------------------------------
        global.alpha.gamma -> Count    (-c, --[alpha.]count, --[alpha.]Count) :␣
↪7
        global.alpha.gamma -> NoDaemon (-d, --[global.]nodaemon)              :␣
↪False
        global.alpha.gamma -> Quiet    (-q, --[global.]quiet)                 :␣
↪False
        global.alpha.gamma -> StdErr   (-s, --[global.]console)               :␣
↪False
        global.alpha.gamma -> Text     (-t, --[alpha.]text, --[alpha.]Text)   :
↪''
        global.alpha.gamma -> Verbose  (-v, --[alpha.]verbose)                :␣
↪2
        global.alpha.gamma -> Xy       (-b, --[gamma.]bbbb)                   :␣
↪False
    ------------------------------------------------------------
    global.beta
    ------------------------------------------------------------
    global.beta            -> NoDaemon (-d, --[global.]nodaemon)              :␣
↪False
    global.beta            -> Quiet    (-q, --[global.]quiet)                 :␣
↪False
    global.beta            -> StdErr   (-s, --[global.]console)               :␣
↪False
    global.beta            -> Verbose  (-v, --[beta.]verbose)                 :␣
↪5

The shortest format is like (recursive = True):

global -> NoDaemon  : False
global -> Quiet     : False
global -> StdErr    : False
global -> Verbose   : 2
alpha  -> Count     : 7
alpha  -> Text      : ''
alpha  -> Verbose   : 2
gamma  -> Xy        : False
beta   -> Verbose   : 5

The shortest format is like (recursive = False):

global -> NoDaemon  : False
global -> Quiet     : False
global -> StdErr    : False
global -> Verbose   : 2
```

property Parents: str

Returns the full qualified parents of this instance e.g.: global.alpha if alpha is a child of global and gamma (this instance) is a child of alpha

Returns

full qualified parents of this instance

Return type
str

property PartPrefix: str

Returns the full qualified prefix without global of this instance e.g.: alpha.gamma if alpha is a child of global and gamma (this instance) is a child of alpha

Returns
full qualified prefix of this instance without root prefix

Return type
str

property Prefix: str

Return the prefix of this class

Returns:
str: the prefix value

Process() → bool

Process the runtime-arguments. After this call the values of the class are all set.

Note: You can not access the values bevore you call this function. The results are undefined.

Raises

- **RuntimeError** – if an internal error occures. Should never occure!
- *ParamError* – if an error occures within a parameter

Returns
True if a terminal function is requested. e.g this are "Help", all "License" and all "Export" options

Return type
bool

SetAddPar(*AddPar: str = ''*) → None

Description of additional parameters for usage-function. printed in first line after "OPTIONS" normally used if there are non-option parameters on command line. (e.g. file … file)

Parameters
AddPar (*str, optional*) – The text or additional parameters. Defaults to "".

Raises
TypeError – if AddPar is not a string

SetAllParams(*AllParams: bool = True*) → None

Set the flag for All Params

Parameters
AllParams (*bool, optional*) – If True, all params are initialized, at least with None. If False params with no default and no setting on the commandline are not defined within the dictionary, defaults to True

SetArgs(*Args: Optional[Union[list, tuple]] = None*) → None

Set the argument list to process

Parameters

Args (*Optional[Union[list, tuple]], optional*) – Runtime Arguments, if None: use sys.argv as the arguments , defaults to None

Raises

TypeError – if Args is not a list or tuple

SetChk(*Chk=None*)

Set the check-function. Not implementet now

Parameters

Chk (*callable*) – The user check function

Raises

TypeError – if function is not of the proper type

SetDef(*Def: dict = {}*) → None

Set the definition for processing arguments

Parameters

Def (*dict, optional*) – A definition-dict. Defaults to {}.

Raises

TypeError – on error within the definition

Describes the definition for arg-parsing.

Def-dict: a dictionary of dictionaries

```
{ 'Name1': {.. declaration ..},
...
'Name2': {.. declaration ..}, }
```

"NameN" is the key with which, at runtime, you get the values within the resulting dictionary.

The individual definitions look like:

```
{'s': 'a',
'l': 'longval',
'o': True,
'v': "LuLu",
'm': 't',
'd': 'Description',
'L': 'Low',
'U': 'Up',
'r': False },
```

where:

```
m : Mode ->
        't' = Text,
        'b' = Bool,
        'p' = Path must not exist but is valid on this operating system,
        'f' = Existing File,
        'd' = Existing Dir,
        'i' = Integer,
```

(continues on next page)

(continued from previous page)

```
        'F' = Float,
        'C' = Counter (start default as 0 and increment each time found)
                Special case: short option takes no argument,
                long option NEEDS argument
                short option increments the value,
                long option adds the argument to the value
The following are processed BEVOR all others:
        'H' = Show help and exit
        '§' = Display short license and exit
        'L' = Display long license and exit
        'x' = Import config-file as json (file must exist like "f")
                    can be given more than once!
        '<' = MultiImport config-file as json
The following are processed AFTER all others:
        'X' = Export config as json to stdout und exit
        '>' = MultiExport config as json to stdout und exit
r : Required -> True or False, False is default
s : Short Option(s) -> string or list or tuple of strings
l : Long Option(s) -> string or list or tuple of strings
o : option needs argument -> True oder False, False is default
v : Default value -> if not given:
        "" for Text,
        False for Bool,
        None for Path, File and Dir,
        0 for Int und Counter,
        0. for Float
        [] for multi-items
L : Lower Limit, value >= L if present
U : Upper Limit, value <= U if present
d : Description for helptext
M : If True: multiples of this option are accepted.
    the resulting value is a list.
```

The entries “m” and (“s” or “l”) must be present, all others are optional.

SetDesc(*Desc: str = ''*) → None

Set the description of the program for usage-string.

Parameters

Desc (*str, optional*) – A descriptive string for the Program. printed bevore the parameters. Defaults to ‘’.

Raises

TypeError – if Desc is not a string.

SetTranslation(*translation: dict*, *IsChild: bool = False*) → None

Seta net translation-table

Parameters

- **translation** (*dict*) – Dictionary with translated error-messages
- **IsChild** (*bool, optional*) – True if we are the root-parent. You should only use this Option if you know what you do. If it is True the **translation** dict is ignored and the translation of the parent is used. Defaults to False

There are 2 'Hidden' function to help debug the translations:

_PrintInitTranslation() and

_PrintAktualTranslation()

Theses functions do exactly what they say: print the values out to stdout. You can use them to get the exact values used. The following default may not be accurate at all - do not rely on this info, print the dict yourselve.

defaults to:

```
{
'PrefixError':
    "Error in prefixed parameter {OptionName}",
'JsonError':
    "Import failed '{wMsg}' in {OptionPath} ({FullPath}) for parameter
↪{OptionName}",
'PathNoFile':
    "The path '{OptionPath}' ({FullPath}) for parameter {OptionName} is not a␣
↪file",
'PathNoDir':
    "The path '{OptionPath}' ({FullPath}) for parameter {OptionName} is not a␣
↪directory",
'PathNoPath':
    "The path '{OptionPath}' ({FullPath}) for parameter {OptionName} is not␣
↪valid on this filesystem",
'LessLow':
    "Value '{OptValue}' for parameter {ParKey} is less than lower limit (
↪{LowLimit})",
'HigherUp':
    "Value '{OptValue}' for parameter {ParKey} is bigger than upper limit (
↪{UppLimit})",
'NoInt':
    "Value '{OptValue}' for parameter {ParKey} is not a valid integer",
'NoFloat':
    "Value '{OptValue}' for parameter {ParKey} is not a valid floating point␣
↪number",
'NoBool':
    "Value '{OptValue}' for parameter {ParKey} is not valid boolean␣
↪(YyTtJj1NnFf0)",
'OptionNotDefined':
    "No action defined for {OptionName}",
'OptionRequired':
    "{DefArgName} ({ParList}) required but not given",
'TypePath':
    "path",
'TypeInteger':
    "integer",
'TypeBool':
    "bool",
'TypeFloat':
    "float",
'TypeFile':
    "file",
```

(continues on next page)

(continued from previous page)

```
'TypeDir':
    "directory",
'TypeCount':
    "counter",
'TypeHelp':
    "help",
'TypeImport':
    "import",
'TypeExport':
    "export",
'TypeGlobImport':
    "global import",
'TypeGlobExport':
    "global export",
'TypeStr':
    "string",
'HelpDefault':
    "Default",
'HelpValue':
    "value",
'HelpUsage':
    "Usage:",
'HelpVersion':
    "Version:",
'HelpOptions':
    "Options:",
'HelpOptionInline':
    "[OPTIONS ...]",
'OptionRequiresArgumentLong':
    "option --{opt} requires argument",
'OptionNeedNoArgs':
    "option --{opt} must not have an argument",
'OptionNotRecognizedLong':
    "option --{opt} not recognized",
'ParNoUniquePrefix':
    "option --{opt} not a unique prefix",
'OptionRequiresArgumentShort':
    "option -{opt} requires argument",
'OptionNotRecognizedShort':
    "option -{opt} not recognized",
'UndefinedOptionSingle':
    "option {OptStr} not recognized",
'UndefinedOptionMultiple':
    "options {OptStr} not recognized",
}
```

SetUserKeys(*UserPars: Optional[dict] = None*, *UserModes: Optional[dict] = None*) → None

summary

Parameters

- **UserPars** (*Optional[dict], optional*) – ignored if None. Defaults to None. Dictionary of keys used within the definition-dictionary. All key-value pairs are optional. Only

the keys from self.__WorkPars are valid. The value has to be a string. This string replaces the keysting for this key. After all changes are made the values within self.__WorkPars have to be unique!, defaults to None

- **UserModes** (*Optional[dict], optional*) – ignored if None. Defaults to None. Dictionary of modes used within the definition-dictionary. All key-value pairs are optional. Only the keys from self.__WorkModes are valid. The value has to be a string. This string replaces the keysting for this key. After all changes are made the values within self.__WorkModes have to be unique!, defaults to None

Raises

- **TypeError** – if invalid type
- *DeclarationError* – if declaration is invalid

property ShortOptsList: list

Return copied list of short options

property TestCommandLineParameter: str

Helper for programmers. Can be used before "Process". Helps to find problems with the command-line interface. Prevent missunderstandig the interface by the user. Should not be used in production environment

Returns
A formatted sting giving all informations about errors or possible problems within the definition(s).

Return type
str

property UnusedArgs: list

Return list of not defined args from the commandline This list is for the current Param-object and all of the children of this Param-object. So under normal conditions it makes only sense on the root of all Param-objects.

Example:
root defines '-z' child defines '-a' commandline is "-a -z –test"

UnusedArgs of child is ['-z','–test'] UnusedArgs of root = ['–test'] which is the correct list of undefined args.

Returns:
list: list of undefined args (str)

Usage(*ShowPrefixHeader: bool = True*) → str

Return the helptext

Returns
The help-text as would be printet if a "Help" option is set on command-line

Return type
str

property UsedLongCommandLineParameter: str

Helper for programmers. Can be used before "Process". Should not be used in production environment

Returns
A formatted, sorted list of all used long options broken to 68 characters a line.

Return type
str

property UsedShortCommandLineParameter: str

Helper for programmers. Can be used before "Process". Should not be used in production environment

Returns
A formatted string giving all information about short options used, broken to 68 characters a line.

Return type
str

_PrintAktualTranslation()

Prints the actual translation dict to stdout.

Only usefull during development!

_PrintInitTranslation()

Prints the initial translation dict to stdout.

Only usefull during development!

__init__(*, *Def: dict = {}*, *Args: Optional[list] = None*, *Chk=None*, *Desc: str = ''*, *AddPar: str = ''*, *AllParams: bool = True*, *UserPars: Optional[dict] = None*, *UserModes: Optional[dict] = None*, *ErrorOnUnknown: bool = True*, *HelpType: int = 0*, *Children: dict = {}*, *translation: dict = {}*, *Version: str = ''*, *License: Optional[Union[str, tuple, list, dict]] = None*, *ShowPrefixOnHelp: bool = True*, *ShowConfigName: bool = False*, *_Child=False*)

This is the constructor of the Param-class.

Most of the parameters can be set also later on (if nessasary)

Note: all parameters are only by name and NOT positional!

Parameters

- **Def** (*dict, optional*) – For details check out *SetDef()*, defaults to None
- **Args** (*list, optional*) – For details check out *SetArgs()*, defaults to None
- **Chk** (*callable, optional*) – For details check out *SetChk()*, defaults to None
- **Desc** (*str, optional*) – For details check out *SetDesc()*, defaults to ""
- **AddPar** (*str, optional*) – For details check out *SetAddPar()*, defaults to ""
- **AllParams** (*bool, optional*) – For details check out *SetAllParams()*, defaults to True
- **UserPars** (*Optional[dict], optional*) – For details check out *SetUserKeys()*, defaults to None
- **UserModes** (*Optional[dict], optional*) – For details check out *SetUserKeys()*, defaults to None
- **ErrorOnUnknown** (*bool, optional*) – If True an error is raised if there are undefined options on the commandline, if False: no error is raised.

  UnusedArgs is always populated with all undefined args from the commandline. This error is raised only on the topmost Param-object (not on children) Defaults to True. So if set to False you can test the 'UnusedArgs' property after return of the 'Process'-function to get the list of undefined args and process this situation on your own. Defaults to True

- **HelpType** (*int, optional*) – Type of helptext.

  0: No Type, no standard defaults, 1: No Type, all defaults, 2: Type, no standard defaults, 3: Type and all defaults,

  defaults to 0

- **Children** (*dict, optional*) – Dictionary of Child-definition for this class.

```
{ 'Name': {'Def': {}, 'Desc': str, 'AddPar': str, 'Children': {} },␣
↪.... }
```

  Name = The name of this child. Must be unique.
  Is translated to lower case. Can not be "global" (this is the name of the root-class)

  Def = A definition dictionary like our own "Def" parameter,

  Children (optional) = dict of type Children, describes the grand-childer,
  this can be done to any level.

  Desc (optional) = A string that describes this class
  (like our own "Desc"-parameter).

  AddPar (optional) = String used as additional info
  (like our own "AddPar"-parameter). Defaults to {}

- **translation** (*dict, optional*) – For details check out *SetTranslation()*, defaults to None

- **License** (*list[str], optional*) – List of license-texts. Thie texts are displayed if a '§' or 'L' -type option is applied. If '§': only the first entry within this list is displayed, if 'L': all entries separated by a newline ('n') are displayed. To help there is a separate module "GPL3" which includes the folowing entries:

  "GPL_Preamble" the suggested preamble (after the copyright notice)

  "GPL_Preamble_DE" the same preamble in German language.

  "LGPL_Preamble" the suggested preamble for the LGPL
  (use after the copyright notice)

  "LGPL_Preamble_DE" the same preamble in German language.

  "LGPL3_2007" the additional terms of the Lesser GPL Version 3 from
  June, 29th 2007, you shold append also the following (GPL3_2007) if you use this license.

  "GPL3_2007" the complete text of the GPL Version 3 from June, 29th 2007

  you can use this by

```
from pcs_argpass.GPL3 import GPL_Preamble, LGPL3_2007, GPL3_2007
```

  and in the constructor of Param par example with:

```
MyParam = Param (
    ....
License=('\nCopyright (c) <date> <your name>\n' + GPL_Preamble_DE,
        GPL_Preamble,
        LGPL3_2007,
        GPL3_2007),
```

(continues on next page)

(continued from previous page)

```
    ....
    )
```

but, of course you can use every other license text you want, as long as this license is compatible with the license of this module, whitch IS LGPL3.

- **Version** (*str, optional*) – Version string for help-display, defaults to ‘’
- **ShowPrefixOnHelp** – if True a header block in the form

```
------------------------------------------------------------
<child-name>
------------------------------------------------------------
```

is printed within the help-output to divide child help from each other defaults to True :type ShowPrefixOnHelp: bool, optional :param ShowConfigName: Anzeige des Parameters innerhalb der Config-Datei, defaults to False :type ShowConfigName: bool, optional :param _Child: True if this instance should be a child, defaults to False :type _Child: bool, optional

---

Note: All long options can be abbreviated to at least 2 characters or to the length making them unique within the defined long options.

Example:

you define ‘automatic’, ‘autonom’ and ‘testopt’ at the commandline this can be abbreviated to

```
PROG --autom --auton --te
```

but not to

```
PROG --au
```

because this is not unique within the optionlist.

Note: If children are used, the prefix-name is always optional, but if given, the option is ONLY recognized for this child. if there are the same long options for more then one child and NO prefix is given, this option is recognized by ALL children having this long option within their definition.

Example:

you define ‘auto’ within the root as ‘MyOpt’ you define ‘auto’ within the child ‘alpha’ as ‘DoAuto’

the commandline should be

```
PROG --auto=yes
```

then BOTH options are set to ‘yes’ e.g. MyParam[‘MyOpt’] == ‘yes’ AND MyParam[‘DoAuto’] == ‘yes’ with the commandline

```
PROG --alpha.auto=yes
```

only MyParam[‘DoAuto’] == ‘yes’ and MyParam[‘MyOpt’] is either the default or not set depending on the definition of this option with the commandline

```
PROG --alpha.auto=yes --global.auto=no
```

MyParam['MyOpt'] == 'no' AND MyParam['DoAuto'] == 'yes' REMEMBER: 'global' is ALWAYS the name of the root Param-object!

please inform your users about this possibility if you use child-definitions!

---

items() → list

Return the items list including the items of all parents

Returns
return the items list

Return type
list

keys() → list

Return the keys list including the keys of all parentsof

Returns
return the keys list

Return type
list

values() → list

Return the values list including the values of all parents

Returns
return the values list

Return type
list

3.3 Examples

3.3.1 Simple example: One file

let's view an example (my native language is German, so I use this with translation):

Ex1.py:

```
#!/usr/bin/env python3
# vim: expandtab:ts=4:sw=4:noai
"""Example 1"""
import sys
from pcs_argpass.Param import Param #, Translation_de_DE
from pcs_argpass.GPL3 import LGPL_Preamble_DE, GPL3_2007, LGPL_Preamble, LGPL3_2007

MyParam:Param = None              # to produce an error if not initialized!

Def_LogMqtt = {                                 # This is the declaration of the command-line
                                                # options
    'Help':                                     # This is the key value for this Option
                                                # must be str and of couse unique
```

(continues on next page)

(continued from previous page)

```
        {
        's': 'h',                       # the short option for this (e.g. -h)
                                        # if more characters are given all of
                                        # them are matched.
        'l': 'help',                    # the long option(s) (e.g. --help). Long
                                        # options can be abbrevated to minimal 2
                                        # characters as long as the abbrevation
                                        # is unique within all long options,
                                        # so also --he or --hel is supported. If this
                                        # entry is a list or tuple then any of this
                                        # entries match.
        'm': 'H',                       # The modus for this option (see details
                                        # in "SetDef" function)
                                        # 'H' means this is one of the "help"
                                        # entries. Since "Help" is a "special" case
                                        # as also "Export" os "Licence" are it is
                                        # possible to have more than one such entry
                                        # in the definition
        'd': 'Show this text and quit.' # the help-text for this option. Can of
                                        # course be also multi-line.

        },
    'MqttBroker':
        {
        's': 'm',
        'l': 'mqttbroker',
        'm': 't',                       # this is a text-entry
        'r': True,                      # it is required
        'o': True,                      # it needs an parameter ("-m" allone makes
                                        # no sense, must be "-m 10.11.12.13" or
                                        # --mqtt=10.11.12.13)
        'v': 'localhost',               # the default value used if this is not on␣
↪the
                                        # command-line
        'd': 'Address of MQTT-Broker',
        },
    'MqttPort':
        {
        's': 'p',
        'l': 'port',
        'm': 'i',                       # This is an integer
        'L': 1024,                      # lower limit. The entered value must
                                        # be >= this value
                                        # (works also for float and str)
        'U': 32766,                     # upper limit. The entered value must
                                        # be <= this value
        'r': True,
        'o': True,
        'v': 1883,
        'd': 'Port of MQTT-Broker',
        },
    'Topic':
        {
```

(continues on next page)

(continued from previous page)

```
        's': 't',
        'l': 'topic',
        'm': 't',                           # also a text entry
        'o': True,
        'M': True,                          # but allow multiple occurences.
                                            # The variable in the resulting
                                            # dictionary is a list!
        'v': [],
        'd': 'Topic to dump',
        },
    'OutFile':
        {
        's': 'f',
        'l': 'file',
        'm': 'p',                           # this is a path, not a file, because
                                            # a file (mode = f) must exist. File is
                                            # used for input-files, path for non
                                            # existing output (file or dir). It is
                                            # only tested if the format is valid
                                            # on this operating system.
        'o': True,
        'v': '',
        'd': 'Output path',
        },
    'License':
        {
        's': '§l',                          # here we have 2 possible short options
        'l': 'license',
        'm': '§',                           # This will show the "short" license
        'd': 'Show license and exit'
        },
    'GPL':
        {
        's': 'g',
        'l': ('gpl','GPL','Gpl'),           # here all of this entries are valid
                                            # on the command-line
        'm': 'L',                           # This will show the complete license
                                            # in this example realy a lot of text.
        'd': 'Show complete license and exit'
        },
}

Version = "1.0.0"
try:                                            # catch illegal definitions
    MyParam = Param(Def=Def_LogMqtt,
                    Desc="dump MQTT-Topics to file",
                    AllParams=True,
                    Version=Version,
#                    translation=Translation_de_DE, # remove this line for english
                                                   # messages
                    License=('\\nCopyright (c) 2022 <your name>\\n' + LGPL_Preamble_DE,
                        LGPL_Preamble,
```

(continues on next page)

(continued from previous page)

```
                        LGPL3_2007,
                        GPL3_2007),
                    )

    if not MyParam.Process():                   # This does the "REAL" processing of
                                                # the command-line args. return True
                                                # if everything is done and the program
                                                # should exit (e.g. Help etc.)

# do your work here.

        # You can use the Param-class like a normal dictionary,
        # so this is perfectly legal
        if len(MyParam['Topic']) == 0:          # no topics given
            MyParam['Topic'].append('#')        # use "ALL" topics

        # if you want to display the given command line options do the following:
        print(MyParam.ParamStr())   # this function returns the
                                                # complete parameters entered

except Param.ParamError as RunExc:      # here we catch any parameter errors and inform␣
↪the user
    print(f"{RunExc }",file=sys.stderr)
    sys.exit(1)
sys.exit(0)
```

Try start this program with “-h” and next time with “-§” or “-L”. Try start it with illegal parameters and look what happens.

If you start the program without any parameters the result will be:

```
------------------------------------------------------------
global
------------------------------------------------------------
global -> MqttBroker (-m, --mqttbroker) : 'localhost'
global -> MqttPort   (-p, --port)       : 1883
global -> OutFile    (-f, --file)       : ''
global -> Topic      (-t, --topic)      : ['#']
```

If you give “-h” or “–help” the result is:

```
Version:: 1.0.0
Usage:

    Ex1 [OPTIONS ...]

dump MQTT-Topics to file
Options:

-h   --help                Show this text and quit.

-m   --mqttbroker=value    Default: 'localhost'
                           Address of MQTT-Broker
```

(continues on next page)

(continued from previous page)

```
-p   --port=value          (1024 ... 32766), Default: 1883
                           Port of MQTT-Broker

-t   --topic=value         Topic to dump

-f   --file=value          Output path

-§   --license             Show license and exit
-l

-g   --gpl                 Show complete license and exit
     --GPL
     --Gpl
```

The module will enshure that you get all requested parameter at least initiated with the default values. All not requested options are found in **UnusedArgs**. If you prefer to not having keys that are not on the command-line set **AllParams** to False. An error will be raised in this case if a "required" parameter ('r' is True) is not given.

It is up to you if you check yourselve for a default value (e.g. '' at OutFile) or let the module check if the parameter is given.

If you simply set the output to sys.stdout if not given you MUST set **AllParams** to True and test the resultvalue yourselfe.

3.3.2 Normal but still simple example: Two files

Normally you put the parameter definition in its own file to make the program more readable:

1.) Ex2_Args.py

```
#!/usr/bin/env python3
# vim: expandtab:ts=4:sw=4:noai
"""Example 2 Args"""
Def_LogMqtt = {
    'Help':
        {
        's': 'h',
        'l': 'help',
        'm': 'H',
        'd': 'Show this text and quit.'
        },
    'MqttBroker':
        {
        's': 'm',
        'l': 'mqttbroker',
        'm': 't',
        'r': True,
        'o': True,
        'v': 'localhost',
        'd': 'Address of MQTT-Broker',
        },
    'MqttPort':
        {
```

(continues on next page)

(continued from previous page)

```
        's': 'p',
        'l': 'port',
        'm': 'i',
        'L': 1024,
        'U': 32766,
        'r': True,
        'o': True,
        'v': 1883,
        'd': 'Port of MQTT-Broker',
        },
    'Topic':
        {
        's': 't',
        'l': 'topic',
        'm': 't',
        'o': True,
        'M': True,
        'v': [],
        'd': 'Topic to dump',
        },
    'OutFile':
        {
        's': 'f',
        'l': 'file',
        'm': 'p',
        'o': True,
        'd': 'Output path',
        },
    'License':
        {
        's': '§l',
        'l': 'license',
        'm': '§',
        'd': 'Show license and exit'
        },
    'GPL':
        {
        's': 'g',
        'l': ('gpl','GPL','Gpl'),
        'm': 'L',
        'd': 'Show complete license and exit'
        },
}
```

2.) Ex2.py

```
#!/usr/bin/env python3
# vim: expandtab:ts=4:sw=4:noai
"""Example 2"""
import sys
from pcs_argpass.Param import Param #, Translation_de_DE
from pcs_argpass.GPL3 import LGPL_Preamble_DE, GPL3_2007, LGPL_Preamble, LGPL3_2007
```

(continues on next page)

(continued from previous page)

```
from Ex2_Args import Def_LogMqtt

MyParam:Param = None              # to produce an error if not initialized!
Version = "1.0.0"

def main():
    """ Do your work here """
    print(MyParam.ParamStr())       # only to do something

if __name__ == '__main__':
    try:                                            # catch illegal definitions
        MyParam = Param(Def=Def_LogMqtt,
                        Desc="dump MQTT-Topics to file",
                        AllParams=True,
                        Version=Version,
#                        translation=Translation_de_DE, # remove this line for english␣
↪messages
                        License=('\\nCopyright (c) 2022 <your name>\\n' + LGPL_Preamble_
↪DE,
                            LGPL_Preamble, LGPL3_2007, GPL3_2007))
        if not MyParam.Process():
            main()
    except Param.ParamError as RunExc:      # here we catch any parameter errors and␣
↪inform the user
        print(f"{RunExc }",file=sys.stderr)
        sys.exit(1)
    sys.exit(0)
```

Now we see that the program is very clear and short as long as it is only for parameter handling.

3.3.3 More complex usage

Let's assume you write a program that uses some type of data handling.

Let's say there is a part that adds a person to some datastructure another part that send the data to an file and at least a possibility to delete this person by an id returned at the time we add it.

We can define a switch for adding, another for printing and so on. The definition will not be very clear to the user. Now we split this in to 4 parts.

1.) Ex3_Args.py

```
#!/usr/bin/env python3
# vim: expandtab:ts=4:sw=4:noai
"""Example 2 Args"""
Def_Main = {
    'Help':
        {
        's': 'h',
        'l': 'help',
        'm': 'H',
        'd': 'Show this text and quit.'
        },
```

(continues on next page)

(continued from previous page)

```
    'License':
        {
        's': '§',
        'l': 'licence',
        'm': '§',
        'd': 'Show license text and quit.'
        },
    'GPL':
        {
        's': 'g',
        'l': 'gpl',
        'm': 'L',
        'd': 'Show full license text and quit.'
        },
    'Verbose': {
        's': 'v',
        'l': 'verbose',
        'r': False,
        'm': 'C',
        'd': "Be more verbose in logging"
        },
}

Def_Add = {
    'Help':
        {
        'l': 'help',
        'm': 'H',
        'd': 'Show this text and quit.'
        },
    'Add':
        {
        's': 'a',
        'l': 'add',
        'm': 't',
        'r': True,
        'o': True,
        'v': '',
        'd': 'Name to add',
        },
    'Department':
        {
        'l': 'department',
        'm': 't',
        'r': False,
        'o': True,
        'v': '',
        'd': 'Optional department',
        },
}

Def_Print = {
```

(continues on next page)

(continued from previous page)

```
    'Print':
        {
        's': 'p',
        'l': 'print',
        'm': 'b',
        'v' : False,
        'd': 'Print values'
        },
}

Def_Del = {
    'Del':
        {
        's': 'd',
        'l': 'del',
        'm': 'i',
        'L': 1,
        'r': True,
        'o': True,
        'v': 0,
        'd': 'Id to delete',
        },
}

Child_Def = {
    'add':
        {
        'Desc': 'Add a new name with optional department',
        'Def': Def_Add,
        },
    'del':
        {
        'Desc': 'delete an Id',
        'Def': Def_Del,
        },
    'print':
        {
        'Desc': 'Print values',
        'Def': Def_Print,
        },
}
```

2.) Ex3.py

```
#!/usr/bin/env python3
# vim: expandtab:ts=4:sw=4:noai
"""Example 3"""

import sys
from Ex3_Args import Def_Main, Child_Def
from pcs_argpass.Param import Param
from pcs_argpass.GPL3 import GPL3_2007, GPL_Preamble
```

(continues on next page)

(continued from previous page)

```
MyParam:Param = None              # to produce an error if not initialized!
Version = "1.0.0"

def main():
    """ Do your work here """
    print(MyParam.ParamStr())       # only to do something

    AddPar = MyParam.Child['add']   # you can get a sub-part of your definitions
    for key,value in AddPar.items():
        print(f"{key} -> {value}")

if __name__ == '__main__':
    try:                                            # catch illegal definitions
        MyParam = Param(Def=Def_Main,
                        Desc="Manage names",
                        AllParams=True,
                        Children=Child_Def,
                        Version=Version,
                        ShowPrefixOnHelp=False,
#                        translation=Translation_de_DE, # remove this line for english␣
↪messages
                        License=('\nCopyright (c) 2022 <your name>\n' + GPL_Preamble,␣
↪GPL3_2007))
        print(MyParam.TestCommandLineParameter)
        if not MyParam.Process():
            main()
    except Param.ParamError as RunExc:      # here we catch any parameter errors and␣
↪inform the user
        print(f"{RunExc }",file=sys.stderr)
        sys.exit(1)
    sys.exit(0)
```

If you invoke this program with "-h" then the output look like this:

```
Version:: 1.0.0
Usage:

    Ex3 [OPTIONS ...]

Manage names
Options:

-h   --help             Show this text and quit.

-§   --licence          Show license text and quit.

-g   --gpl              Show full license text and quit.

-v   --verbose=value    Be more verbose in logging
```

(continues on next page)

(continued from previous page)

```
    Add a new name with optional department

         --[add.]help                Show this text and quit.

    -a   --[add.]add=value           Name to add

         --[add.]department=value    Optional department


    delete an Id

    -d   --[del.]del=value    (1 ...)
                                Id to delete


    Print values

    -p   --[print.]print          Print values
```

As you see the different options are grouped and (in this simple case not relly necassary) vied as "function groups". But all keys are accessible by their names from the normal Param instance.

Look at –[add.]help:
this means there is a second help entry within this declaration. It only has a "long" option an can therefore be invoked by

```
Ex3.py --add.help
```

in this case the folowing output is generated:

```
#------------------------------------------------------------
# add
#------------------------------------------------------------


Add a new name with optional department

     --[add.]help                Show this text and quit.

-a   --[add.]add=value           Name to add

     --[add.]department=value    Optional department
```

I know this is not a great benefit for THIS application but if you have a lot of parameters in an big cli it is helpfull. It is up to you if you put a separate "help"-entry in your definition, but the rest is done by the module.

It is possible to get the sub-instances by the "Child" property and work with them exact the same way as with the main instance. It is also possible to nest this system ad infinitum.

Remember:
All sub-instances inherit the keys of all of their parents! A parent has also all keys of all of his children, grand-childen and so on.

If you run this program without any parameter the result is:

```
------------------------------------------------------------
global
------------------------------------------------------------
global    -> Verbose    (-v, --verbose) : 0
    ------------------------------------------------------------
    add
    ------------------------------------------------------------
    add   -> Add        (-a, --add)     : ''
    add   -> Department (--department)  : ''
    add   -> Verbose                    : 0
    ------------------------------------------------------------
    del
    ------------------------------------------------------------
    del   -> Del        (-d, --del)     : 0
    del   -> Verbose                    : 0
    ------------------------------------------------------------
    print
    ------------------------------------------------------------
    print -> Print      (-p, --print)   : False
    print -> Verbose                    : 0
Add ->
Department ->
Verbose -> 0
```

Now we see that all children have the **Verbose** option of their parent also within their keys, but that are no copies but only references to their parent. So the key-linking is up and down the tree.

The last 3 lines is the result of the printing of the clild-instance in the **main** function. It is exactly what we expect!

PYTHON MODULE INDEX

INDEX

Symbols

A

C

D

F

G

I

K

L

M

O

P

S

T

U

V