iocdoc Documentation

Release 0.0.2

Pete R. Jemian

May 06, 2016

Contents

CHAPTER 1

Overview

version 0.0.2

published May 06, 2016

IOCDOC: Document the configuration of an EPICS IOC

In EPICS, an IOC is the server, coordinating hardware actions with software configuration, command, and control.

This project arose from a user request: *What does my EPICS system provide? Tell me about all the PVs.*

Operation of an IOC begins with a command to a executable, compiled for the host computer architecture, and a startup script, often called `st.cmd` by convention. All necessary configuration information is provided in the startup script or through the environment variables from the host operating system when the startup script is launched.

It is possible, by parsing the startup script, to document the implementation of the IOC and that is the goal of this package. Reference to well-known IOC commands (such as `dbLoadRecords`), packages (such as `synApps`), and the chosen EPICS base version will be provided.

Output is in a set of restructured text files for compilation by Sphinx into HTML and/or PDF. Reference to any additional HTML documentation provided in the IOC's directory structure will be included.

Contents:

1.1 IOCDOC: Document the configuration of an EPICS IOC

iocdoc: Document the configuration of an EPICS IOC

docs http://iocdoc.readthedocs.org

git https://github.com/prjemian/iocdoc.git

PyPI https://pypi.python.org/pypi/iocdoc

TODO list https://github.com/prjemian/iocdoc/issues

author Pete R. Jemian

email jemian@anl.gov

The user will provide the directory path (absolute or relative) to an EPICS IOC startup command file. The IOC startup file will specify the starting point to discover the configuration of each IOC.

This code will parse the startup command file and discover the commands and databases used to start the IOC. An attempt is made to read the database files. This program will report if a file or database is not found.

The goal is to generate a set of WWW pages with appropriate links to standard documentation for all the commands and other common support. Links will be made to any local documentation provided in the ./documentation directory of the top level directory.

1.2 Techniques used in coding this package

Text file parsing relies on the Python `tokenize` package ...

Sphinx is used to prepare documentation ...

The `lxml` package ...

1.3 module: command_file

1.4 module: database

EPICS database file analysis

class `iocdoc.database.`**`Database`**(*parent*, *dbFileName*, *ref*, ***env*)

Bases: `object`

call for one EPICS database file with a given environment

`makeProcessVariables`()

make the EPICS PVs from the record definitions

`parse`()

interpret records for PV declarations

1.5 module: ioc

1.6 module: macros

support for macro substitutions

From the EPICS Application Developer's Guide

see http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide/node7.html

6.3.2 Unquoted Strings

In the summary section, some values are shown as quoted strings and some unquoted. The actual rule is that any string consisting of only the following characters does not have to be quoted unless it contains one of the above keywords:

```
a-z A-Z 0-9 _ -- : . [ ] < > ;
```

```
my regexp:  [\w_\-:.[\]<>;]+([.][A-Z0-9]+)?
```

These are also the legal characters for process variable names. Thus in many cases quotes are not needed.

6.3.3 Quoted Strings

A quoted string can contain any ascii character except the quote character ". The quote character itself can given by using as an escape. For example """" is a quoted string containing the single character ".

6.3.4 Macro Substitution

Macro substitutions are permitted inside quoted strings. Macro instances take the form:

```
$(name)
```

or

```
${name}
```

There is no distinction between the use of parentheses or braces for delimiters, although the two must match for a given macro instance. The macro name can be made up from other macros, for example:

```
$(name_$(sel))
```

A macro instance can also provide a default value that is used when no macro with the given name is defined. The default value can be defined in terms of other macros if desired, but cannot contain any unescaped comma characters. The syntax for specifying a default value is as follows:

```
$(name=default)
```

Finally macro instances can also contain definitions of other macros, which can (temporarily) override any existing values for those macros but are in scope only for the duration of the expansion of this macro instance. These definitions consist of name=value sequences separated by commas, for example:

```
$(abcd=$(a)$(b)$(c)$(d),a=A,b=B,c=C,d=D)
```

class iocdoc.macros.**KVpair**(*parent*, *key*, *value*, *ref=None*)

Bases: object

any *single* defined key:value pair in an EPICS IOC command file

- PV field
- Record field
- Macro
- Symbol

class iocdoc.macros.**Macros**(***env*)

Bases: object

manage a set of macros (keys, substitutions)

exists(*key*)

is there such a *key*?

get(*key*, *missing=None*)

find the *key* macro, if not found, return *missing*

items()

get the full database, like dictionary.items()

keys()
get the list of macros, like dictionary.keys()

replace(*text*)
Replace macro parameters in source string

set(*key*, *value*, *parent=None*, *ref=None*)
define the *key* macro

setMany(**env*)
define several macros

iocdoc.macros.**identifyEpicsMacros**(*source*)
Identify any EPICS macro substitutions in the source string. Multiple entries of the same substitution (redundancies) are ignored. Does not include nested macros such as:

```
$(P=$(S))${S_$(P)}
$(PJ=$(P))${S_$(P)}
```

For these, only the innermost are returned:

```
['$(S)', '$(P)']
['$(P)']
```

Note This routine will also properly identify command shell macro substitutions.

Parameters **source** – string with possible (EPICS) macro substitution expressions

Returns list of macro substitutions found

1.7 module: record

EPICS record

class iocdoc.record.**PV**(*record_object*, *ref*, **env*)
Bases: object

single instance of an EPICS record, will expand all macros as provided

class iocdoc.record.**Record**(*parent*, *rtype*, *rname*, *ref*, **env*)
Bases: object

definition of an EPICS record, macros are not expanded

1.8 module: reports

1.9 module: template

1.10 module: text_file

text_file.py - describe any text file used by an EPICS IOC

code	description
read()	get a file either from the cache or from storage
items()	get the cache as a set of dictionary items
keys()	get the names of files in the cache
values()	get the Python objects of items in the cache
remove_comments()	strip out a C-style comment
FileNotFound	Exception: raised when filename does not exist
_FileCache	(internal) supports "load each file only once"
_TextFile	(internal) superclass: common handling of text file

Example:

```
# filename must have all macros expanded
file_object = text_file.read(filename)
```

iocdoc.text_file.**items**()
get the cache as a set of dictionary items

iocdoc.text_file.**keys**()
get the names of files in the cache

iocdoc.text_file.**read**(*filename*)
get a file either from the cache or from storage

Parameters **filename** (*str*) – relative or absolute path to file

Always use filenames with all macros expanded.

Ok?	filename
Ok	st.cmd
Ok	./testfiles/templates/example.template
not Ok	$(SSCAN)/sscanApp/Db/scanParms.db

iocdoc.text_file.**values**()
get the Python objects of items in the cache

1.11 module: token_support

Applies Python tokenize analysis to each line of a text file.

class iocdoc.token_support.**TokenLog**
Applies the Python <code>tokenize</code> analysis to each line of a file. This allows a lexical analysis of the file, line-by-line. This is powerful and makes some complex analyses more simple but it assumes the file resembles Python source code.

:note The <code>tokenize</code> analysis is not robust. Some files will cause exceptions for various reasons.

:see http://docs.python.org/library/tokenize.html :see http://docs.python.org/library/token.html

get(*index*)
retrieve the indexed token from the list

getCrossReferences()

Returns dictionary of token cross-references

getFullWord(*skip_list=None*)
parse the token stream for a contiguous word and return it as str

Some words in template files might not be enclosed in quotes and thus the whole word is broken into several tokens. This command rebuilds the word, without stripping quotes (if provided).

getKeyValueSet()
parse a token sequence as a list of macro definitions into a dictionary

example:

```
{ P=12ida1:,SCANREC=12ida1:scan1 }
{P=12ida1:,SCANREC=12ida1:scan1,Q=m1,POS="$(Q).VAL",RDBK="$(Q).RBV"}
```

getTokenList()

Returns list of token dictionaries

lineAnalysis()
analyze the tokens by line

:return dictionary with all the lines, including tokenized form

next()
return the next token or raise a StopIteration exception upon reaching the end of the sequence

Returns token object

Raises **StopIteration** – reached the end of the sequence

nextActionable(*skip_list=None*)
walk through the tokens and find the next actionable token

Parameters **skip_list** (*(str)*) – list of tokens to ignore

default list: ('COMMENT', 'NEWLINE', 'ENDMARKER', 'ERRORTOKEN', 'INDENT', 'DEDENT')

Returns token object or None if no more tokens

previous()
return the previous token

Returns token object

Raises **StopIteration** – reached the beginning of the sequence

processFile(*filename*)
process just one file

setTokenPointer(*position=None*)
set the token pointer to the given position

Parameters **position** – index position within list of tokens

Raises **Exception** – token pointer position errors

tokenName(*tokType*)
convert token number to a useful string

tokenReceiver(*tokType*, *tokStr*, *start*, *end*, *tokLine*)
called by tokenize package, logs tokens as they are called

tokens_to_list()
parse an enclosed list of tokens into a list

Assume `token_pointer` is pointing at start terminator

examples:

```
        (DESC, "motor $(P)$(M)") --> ['DESC', 'motor $(P)$(M)']
        {P,      S, BL,    T1, T2, A}  --> ['P', 'S', 'BL', 'T1', 'T2', 'A']
        {12ida1: A  "##ID" 1   2   1}  --> ['12ida1:', 'A', '##ID', '1', '2', '1']

        TODO: alias($(IOC):IOC_CPU_LOAD,"$(IOC):load")
```

iocdoc.token_support.**parse_bracketed_macro_definitions**(*tokenLog*)
walk through a bracketed string, keeping track of delimiters

verify we start on an opening delimiter

iocdoc.token_support.**reconstruct_line**(*tokens=[]*, *firstIndex=1*, *no_comments=True*)
reconstruct the line from the list of tokens presented

Parameters

- **tokens** (*[tok_dict]*) – as used throughout this module
- **firstIndex** (*int*) – first index in tokens list to use
- **no_comments** (*bool*) – True (default) to stop reconstructing at the first comment token

Returns reconstructed line

iocdoc.token_support.**token_key**(*tkn*)
developer use, short string identifying the type and text of this token

1.12 module: utils

common routines for many modules

support	description
chdir()	change current IOC shell directory
datenow()	current date/time, as a str
detailedExceptionLog()	log the details of an exception
FileRef	associate filename and line number of an object
logMessage()	log a message
strip_outer_pair()	remove outer symbols from text
strip_outer_quotes()	strip outer quotes (either single or double) from text
remove_c_comments()	strip out a C-style comment
LOG_FILE	default log file name (must be defined *before* logging is started)
LOGGING_DETAIL	maximum level of detail to report in log (default=2, range: 0-5)
strip_parentheses()	remove outer parentheses from text
strip_quotes()	strip outer double quotes from text

class iocdoc.utils.**FileRef**(*filename*, *linenumber*, *colnumber*, *obj*)
Bases: object

associate filename and line number of an object

iocdoc.utils.**chdir**(*newDir*, *nfsMounts={}*)
change the current working directory for the IOC shell

Parameters **newDir** – name of new directory

Returns success (True) or failure (False)

iocdoc.utils.**datenow**()
return date and time now as a string

iocdoc.utils.**detailedExceptionLog**(*title=''*, *print_traceback=True*)
enter details of an exception to the log (developer tool)

- always log that an exception was reported
- the full traceback details are logged at a higher level

iocdoc.utils.**logMessage**(*text*, *detail=2*)
log a message

Parameters

- **text** (*obj*) – item to be logged, assumed to be a string but will be rendered with `str(text)`
- **detail** (*int*) – interest level for this logging item, must be <= LOGGING_DETAIL to be logged

iocdoc.utils.**remove_c_comments**(*text*)
strip out a C-style comment

```
/* such as this */
```

Parameters **text** (*str*) – text with possible comment

See http://stackoverflow.com/questions/241327/python-snippet-to-remove-c-and-c-comments

iocdoc.utils.**setLogDetailLevel**(*detail=2*)
define the logging (reporting) detail level

Parameters **detail** (*int*) – interest level for logging items, must be <= LOGGING_DETAIL to be logged

iocdoc.utils.**setLogFile**(*logFile*)
define the log file name

Parameters **logFile** (*str*) – name of log file to be used

Raises **RuntimeError** – if called after logging has started

iocdoc.utils.**strip_outer_pair**(*text*, *left*, *right=None*)
remove outer symbols from text

Parameters

- **text** – string
- **left** – symbol on left side
- **right** – symbol on right side (default is left-side symbol)

Returns modified string

Raises **Exception** – left and right must have len(..) == 1

iocdoc.utils.**strip_outer_quotes**(*text*)
strip outer quotes (either single or double) from text

Returns text without comments

iocdoc.utils.**strip_parentheses**(*text*)
remove outer parentheses from text

Parameters **text** – string

Returns modified string

iocdoc.utils.**strip_quotes**(*text*, *quote=""*)
strip outer double quotes from text

Returns text without comments

1.13 CHANGES

2016-03-24 0.0.2 pre-release

2016-01-29 0.0.1 - project to be refactored from topdoc

2014 TopDoc - https://subversion.xray.aps.anl.gov/trac/bcdaext/browser/topdoc

1.14 Source Code License

```
Copyright (c) 2011-2016, UChicago Argonne, LLC

All Rights Reserved

IOCDOC

Advanced Photon Source, Argonne National Laboratory


OPEN SOURCE LICENSE

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.  Software changes,
   modifications, or derivative works, should be noted with comments and
   the author and organization's name.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the names of UChicago Argonne, LLC or the Department of Energy
   nor the names of its contributors may be used to endorse or promote
   products derived from this software without specific prior written
   permission.

4. The software and the end-user documentation included with the
   redistribution, if any, must include the following acknowledgment:

   "This product includes software produced by UChicago Argonne, LLC
   under Contract No. DE-AC02-06CH11357 with the Department of Energy."

*****************************************************************************

DISCLAIMER

THE SOFTWARE IS SUPPLIED "AS IS" WITHOUT WARRANTY OF ANY KIND.

```

```
Neither the United States GOVERNMENT, nor the United States Department
of Energy, NOR UChicago Argonne, LLC, nor any of their employees, makes
any warranty, express or implied, or assumes any legal liability or
responsibility for the accuracy, completeness, or usefulness of any
information, data, apparatus, product, or process disclosed, or
represents that its use would not infringe privately owned rights.

*****************************************************************************
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

i

T

V