Documentation style guide

RobotFramework AIO team

01.02.2023

Contents

Chapter 1

Introduction

The documentation of the RobotFramework AIO and also the documentation of every single component of the RobotFramework AIO is provided in PDF format. Most of the sources of the documentation (except the interface definitions of Python modules) are written in LaTeX. Parts of the LaTeX sources are written manually, other parts of the LaTeX sources are generated out of the interface definitions of Python modules automatically by a converter tool named Pandoc.

The layout of the PDF output is determined by certain LaTeX style files. These style files are e.g. responsible for the layout of code listings (syntax highlighting), the definition of LaTeX commands used by Pandoc and certain admonitions (like text boxes indicating e.g. a warning or an error).

In the context of RobotFramework AIO the documentation files in PDF format are available at two different levels, generated by two different tool chains. This is based on the fact that the code of the complete RobotFramework AIO is developed in several repositories (every component of the RobotFramework AIO has it's own repository).

1. The tool `GenPackageDoc` creates the documentation of a Python package within a single repository.
2. The tool `GenMainDoc` creates the main documentation of the RobotFramework AIO (including the documentation of it's components).

Both tool chains work with the same LaTeX style files:

1. `admonitions.sty`
   Defines admonitions like: `boxwarning` , `boxerror` , `boxgoodpractice` , `boxhint` , `boxtip`
2. `common.sty`
   Defines common commands like support for `history.tex`
3. `pandoc.sty`
   Defines commands used by Pandoc when computing code directives in rst sources
4. `robotframeworkaio.sty`
   Defines RobotFramework AIO specific elements like syntax highlighting for Robot Framework and Python code

All required style files are imported by `preamble.tex` . In case you plan to introduce LaTeX files outside the scope of `GenPackageDoc` and `GenMainDoc` , make sure to import the preamble in the following way (like in this LaTeX source file):

```
\input{./styles/preamble}
```

We recommend to use this preamble in all LaTeX files to give all resulting PDF files a unique look&feel.

Chapter 2

Admonitions

`admonitions.sty` contains the definition of certain text boxes with certain symbols and background colors. The purpose is to highlight important parts of the documentation.

Warnings

```
\begin{boxwarning}{Warning}
This is a box which gives you a warning.
\end{boxwarning}
```

Warning

This is a box which gives you a warning.

Errors

```
\begin{boxerror}{Error}
This is a box which informs you about an error.
\end{boxerror}
```

Error

This is a box which informs you about an error.

Good practices

```
\begin{boxgoodpractice}{Good practice}
This is a box which informs you about a good practice.
\end{boxgoodpractice}
```

Good practice

This is a box which informs you about a good practice.

Hints

```
\begin{boxhint}{Hint}
This is a box which gives you a hint
\end{boxhint}
```

Hint

This is a box which gives you a hint

Tips

```
\begin{boxtip}{Tip}
This is a box which gives you a tip
\end{boxtip}
```

Tip

This is a box which gives you a tip

Chapter 3

Common styles

Every component repository contains a LaTeX file `History.tex` .
A table containing the package history, can be written in the following way:

```
\begin{packagehistory}

\historyversiondate{0.1.0}{07/2022}
\historychange{Initial version}

\historyversiondate{0.2.0}{08/2021}
\historychange{- new feature\newline
- another change}

\end{packagehistory}
```

0.1.0	07/2022
Initial version	
0.2.0	08/2021
- new feature *- another change*	

Chapter 4

Pandoc

In the context of RobotFramework AIO the converter tool Pandoc is used to convert the interface definitions (that are the content of the docstrings of the corresponding Python modules) to LaTeX format. For this purpose Pandoc introduced some LaTeX commands that need to be defined. A user writing documentation has no direct contact to this and can ignore the content of `pandoc.sty` .

Chapter 5

RobotFramework AIO styles

Syntax highlighting is available for RobotFramework AIO code and for Python code. Also for console listings and file system informations (like file names) an own layout is available.

The common layout for RobotFramework AIO code and Python code is the same. Differences only belong to the keyords that are language specific.

The layouts are available in two different characteristics: complete text block and inline.

The main difference between code listings and console listings is the background color of the text boxes containing the listing:

- Code listings are indicated by a background color of light orange: `code`
- Console listings are indicated by a background color of light blue: `console`

5.1 Abbreviations

Framework names

To ease a unique typing of the frameworks the following LaTeX commands are available: `rfw` and `rfwcode`:

```
The \rfw\ is derived out of the \rfwcore.
```

The RobotFramework AIO is derived out of the Robot Framework.

Tabulators

The usage of tabulators should be avoided but it is possible to use the `tab` command to realize indentations (3 blanks):

```
\texttt{line 1}\\
\tab \texttt{line 2}\\
\tab \texttt{line 3}\\
\tab \texttt{line 4}\\

line 1
   line 2
   line 3
   line 4
```

Test results

The following commands can be used to type test results; it's not necessary to take care about the corresponding colors manually:

```
The result of the first test is \passed, the result of the second test is \failed.
We also have one test with result \unknown\ and the last test is \aborted.
```

The result of the first test is passed, the result of the second test is failed. We also have one test with result unknown and the last test is aborted.

5.2 Code listings

RobotFramework AIO code block

```
\begin{robotcode}
*** Test Cases ***
Hello World
   FOR    ${index}    IN RANGE    1    11
      Log  Hello world ${index}
   END
\end{robotcode}
```

```
*** Test Cases ***
Hello World
   FOR    ${index}    IN RANGE    1    11
      Log  Hello world ${index}
   END
```

RobotFramework AIO code block (extended)

It is possible to give the listing a caption. It is also possible to highlight certain lines within the listing. The highlighting can be realized by a list of numbers of all lines, that shall be highlighted (`hlcode`):

```
\begin{robotcode}[caption=Robot code example,
                  linebackgroundcolor=\hlcode{1,4}]
*** Test Cases ***
Hello World
   FOR    ${index}    IN RANGE    1    11
      Log  Hello world ${index}
   END
\end{robotcode}
```

```
*** Test Cases ***
Hello World
   FOR    ${index}    IN RANGE    1    11
      Log  Hello world ${index}
   END
```

Listing 5.1: Robot code example

RobotFramework AIO inline code

```
The variable is set in this way: \rcode{$\{prev\}\ \ \ \ Set Variable\ \ \ \ $\{0\}}
```

The variable is set in this way: `${prev}    Set Variable    ${0}`

Consider that inside `rcode` the brackets { and } and also the blanks have to be masked!

Python code block

```
\begin{pythoncode}
def print_pattern():
    values = ("A", "B", "C")
    for value in values:
        print(f"value : {value}")
\end{pythoncode}
```

```
def print_pattern():
    values = ("A", "B", "C")
    for value in values:
        print(f"value : {value}")
```

Python code block (extended)

It is possible to give the listing a caption. It is also possible to highlight certain lines within the listing. The highlighting can be realized by a list of numbers of all lines, that shall be highlighted (`hlcode`):

```
\begin{pythoncode}[caption=Python code example,
                   linebackgroundcolor=\hlcode{1,3}]
def print_pattern():
    values = ("A", "B", "C")
    for value in values:
        print(f"value : {value}")
\end{pythoncode}
```

```
def print_pattern():
    values = ("A", "B", "C")
    for value in values:
        print(f"value : {value}")
```

Listing 5.2: Python code example

Python inline code

```
The variable is set in this way: \pcode{values = ("A", "B", "C")}
```

The variable is set in this way: `values = ("A", "B", "C")`

5.3 Console listings

Console listings are also available for both RobotFramework AIO and Python.

To keep this layout stuff simple, the console listings layout is also used for all informations related to the file system (e.g. paths and file names).

RobotFramework AIO console listing

```
\begin{robotlog}
==============================================================================
HelloWorld
==============================================================================
Hello World                                                           | PASS |
------------------------------------------------------------------------------
HelloWorld                                                            | PASS |
1 test, 1 passed, 0 failed, 0 unknown
==============================================================================
Output:  C:\RobotTest\testcases\output.xml
Log:     C:\RobotTest\testcases\log.html
Report:  C:\RobotTest\testcases\report.html
\end{robotlog}
```

```
==============================================================================
HelloWorld
==============================================================================
Hello World                                                           | PASS |
------------------------------------------------------------------------------
HelloWorld                                                            | PASS |
1 test, 1 passed, 0 failed, 0 unknown
==============================================================================
Output:  C:\RobotTest\testcases\output.xml
Log:     C:\RobotTest\testcases\log.html
Report:  C:\RobotTest\testcases\report.html
```

RobotFramework AIO console listing (extended)

It is possible to give the listing a caption. It is also possible to highlight certain lines within the listing. The highlighting can be realized by a list of numbers of all lines, that shall be highlighted (`hllog`):

```
\begin{robotlog}[caption=Robot log example,
                 linebackgroundcolor=\hllog{7,11}]
==============================================================================
HelloWorld
==============================================================================
Hello World                                                           | PASS |
------------------------------------------------------------------------------
HelloWorld                                                            | PASS |
1 test, 1 passed, 0 failed, 0 unknown
==============================================================================
Output:  C:\RobotTest\testcases\output.xml
Log:     C:\RobotTest\testcases\log.html
Report:  C:\RobotTest\testcases\report.html
\end{robotlog}
```

```
==============================================================================
HelloWorld
==============================================================================
Hello World                                                           | PASS |
------------------------------------------------------------------------------
HelloWorld                                                            | PASS |
1 test, 1 passed, 0 failed, 0 unknown
==============================================================================
Output:  C:\RobotTest\testcases\output.xml
Log:     C:\RobotTest\testcases\log.html
Report:  C:\RobotTest\testcases\report.html
```

Listing 5.3: Robot log example

RobotFramework AIO inline console output

```
The test results are shown like this: \rlog{1 test, 1 passed, 0 failed, 0 unknown}
```

The test results are shown like this: `1 test, 1 passed, 0 failed, 0 unknown`

```
The results of the test execution can be found in the following files:
\begin{robotlog}
C:\RobotTest\testcases\output.xml
C:\RobotTest\testcases\log.html
C:\RobotTest\testcases\report.html
\end{robotlog}
```

The results of the test execution can be found in the following files:

```
C:\RobotTest\testcases\output.xml
C:\RobotTest\testcases\log.html
C:\RobotTest\testcases\report.html
```

```
The file \rlog{log.html} contains the log file in html format.
```

The file `log.html` contains the log file in html format.

Python console listing

```
\begin{pythonlog}
==============================================================================
HelloWorld
==============================================================================
Hello World                                                           | PASS |
------------------------------------------------------------------------------
HelloWorld                                                            | PASS |
1 test, 1 passed, 0 failed, 0 unknown
==============================================================================
Output:  C:\RobotTest\testcases\output.xml
Log:     C:\RobotTest\testcases\log.html
Report:  C:\RobotTest\testcases\report.html
\end{pythonlog}
```

```
==============================================================================
HelloWorld
==============================================================================
Hello World                                                           | PASS |
------------------------------------------------------------------------------
HelloWorld                                                            | PASS |
1 test, 1 passed, 0 failed, 0 unknown
==============================================================================
Output:  C:\RobotTest\testcases\output.xml
Log:     C:\RobotTest\testcases\log.html
Report:  C:\RobotTest\testcases\report.html
```

Python console listing (extended)

It is possible to give the listing a caption. It is also possible to highlight certain lines within the listing. The highlighting can be realized by a list of numbers of all lines, that shall be highlighted (`hllog`):

```
\begin{pythonlog}[caption=Python log example,
                  linebackgroundcolor=\hllog{7,11}]
==============================================================================
HelloWorld
==============================================================================
Hello World                                                           | PASS |
------------------------------------------------------------------------------
HelloWorld                                                            | PASS |
1 test, 1 passed, 0 failed, 0 unknown
==============================================================================
Output:  C:\RobotTest\testcases\output.xml
Log:     C:\RobotTest\testcases\log.html
Report:  C:\RobotTest\testcases\report.html
\end{pythonlog}
```

```
==============================================================================
HelloWorld
==============================================================================
Hello World                                                           | PASS |
------------------------------------------------------------------------------
HelloWorld                                                            | PASS |
1 test, 1 passed, 0 failed, 0 unknown
==============================================================================
Output:  C:\RobotTest\testcases\output.xml
Log:     C:\RobotTest\testcases\log.html
Report:  C:\RobotTest\testcases\report.html
```

Listing 5.4: Python log example

Python inline console output

```
The test results are shown like this: \plog{1 test, 1 passed, 0 failed, 0 unknown}
```

The test results are shown like this: `1 test, 1 passed, 0 failed, 0 unknown`

```
The results of the test execution can be found in the following files:
\begin{pythonlog}
C:\RobotTest\testcases\output.xml
C:\RobotTest\testcases\log.html
C:\RobotTest\testcases\report.html
\end{pythonlog}
```

The results of the test execution can be found in the following files:

```
C:\RobotTest\testcases\output.xml
C:\RobotTest\testcases\log.html
C:\RobotTest\testcases\report.html
```

```
The file \plog{log.html} contains the log file in html format.
```

The file `log.html` contains the log file in html format.

Chapter 6

Documentation style guide

This chapter contains some common guidelines about how to write a good documentation.

6.1 Think didactically

We can assume: During the hurry of daily work no one is in the mood to read documentations. The only reason for any audience to read a documentation is: They have a problem! They have to solve the problem urgently. To solve the problem, they need information. And their hope is to find the required information within the documentation. If they find, they will be happy. If not, they will not trust the documentation any more and most probably they never will take a lock at the documentation again.

Writing a good documentation means: Anticipate what might be the problems of the audience. And then think about in which way informations have to be given, to solve these problems. Think about how to structure the documentation to make it as easy as possible for the audience to find inside the documentation exactly the position at which the needed information is given.

Do not jump too fast into technical details. Spend some preliminaries to tell about the scope of the following. In this context it is also important to define new technical terms before they are used in following parts of the documentation.

Think in usecases. A listing of functions is helpful also - but should be placed at the end of the document in the appendix (like already realized in the interface description part). The common part of a documentation should focus on the hardships of the audience. Already by using good headlines you help the audience to find required informations faster. For example it is a usecase to import parameters. In this case a headline like "How to import parameters" is much easier to capture than the shorter "Import parameters".

6.2 Check dependencies to RobotFramework AIO

In case of a component is designed to be stand alone, you should not use the term "RobotFramework AIO" any where in the documentation. Use the term "Robot Framework" instead. Additionally you should avoid to use the abbreviation "AIO" anywhere in the component code.

In case of a component is used as part of the RobotFramework AIO and in case of this circumstance does change anything, then the documentation of these things should be handled as add-on and should be placed in a separate follow up chapter of the documentation. In this case both possible worlds (with and without RobotFramework AIO) are clearly separated from each other. This will make it easier for the audience, to understand how to work with the component.

6.3 Typing

Please consider also, that too much typing mistakes in a document are really annoying for the audience and disturb the concentration on the content.

6.4 Release

Before you think about to release a new or maintained documentation, you should read the entire documentation once again.
And after this once again.
And after this once again.
And after this once again.

This will definitely decrease the number of errors in the documentation dramatically!