**Beyond4P Automatic Document Generation Scheme**

**Automatic generation of HTML and online documentation**

**Georg A. zur Bonsen**

* **Feature**
* **Feature**
* **Feature**

All rights reserved, copyright © 2012…2020 by Georg zur Bonsen

[1 Introduction 5](#_Toc30828931)

# Introduction

In the past, all Beyond4P features have been described in a single document from which PDF's have been derived. In the following versions, the documents will be shifted to an online scheme using standardized data formats.

* Original documentation should be located in the same file where the implementation has been made, i.e. in the C/C++ or Beyond4P library code in form of comment blocks. More generalized document sections may be located in separate text files
* The decentral nature of original documentation allows easy integration of additional contents, e.g. from persons who provided a library file with additional functions
* JSON is to be used as original structural framework to provide overseeable data stucture
* Text contents should be in HTML, allowing some text styles and including tables and illustrations where needed
* A Beyond4P application script will scan all applicable text files to capture documentations made and generate various outputs.
* Outputs can include:  
  - A collection of HTML files  
  - Rich text format  
  - Online help text for Beyond4P

# Document Source

## Source Files

Document sources are C/C++ files, Beyond4P and other text files. UTF-8 character format is assumed to support non-ANSI characters.

## Documentation Zones

In source code files (C/C++) and Beyond4P, the documentations are included as comments. Start and end markers indicate the location of applicable documentation called **documentation zones**. Use full comment sections (like /\* … \*/ ) and not line comments (like //) to include documentation.

...

...

// C/C++ or Beyond4P statements or any other text (ignored)

/\* **B4PDOCU.START**

// Documentation contents in JSON

// Format according to rules

// described in the follwing sections

// Comments inside documentation zones are considered as comments and

// are not included in the documentation.

**B4PDOCU.STOP** \*/

// C/C++ or Beyond4P statements or any other text follow next(ignored)

...

...

Multiple documentation zones are allowed. I suggest to include the documentation zones closely located to their implementations.

Beyond4P document generator will search for rows containing 'B4PDOCU.START' and 'B4PDOCU.STOP' to capture contents. These markers must be in capital letters and not used for any other purpose, e.g. printf("B4PDOCU.START\n"); somewhere else in the code. If needed, then do printf("B4PDOCU"); printf(".START\n");

The JSON structure is a standardized format and fully supported by Beyond4P. The Document generator implemented with Beyond4P can read, process and write JSON contents very easily.

## Documentation Schemes

Following **documentation schemes** exist:

1. Generic Text  
   Generic contents, e.g. introduction to Beyond4P.
2. Value Descriptions  
   Defining various terms, typically in a table. Example: Variable types, different consolidation actions
3. Function Descriptions  
   Structural definition of procedures and functions including their parameters, return values and examples

All docu contents must be associated with a specific section. This is needed for ordering purposes so the final document generated is in meaningful structure and ordering

### Text Formats

Unicode is fully supported (UTF-8.

Entities (e.g. &amp; &eaigu;) are supported

Quotation marks as part of contents must precede with backslash, e.g. \" to avoid confusing with end of string

### Color Conventions in Examples

Different colors are used to better describe the document structure to use.

|  |  |
| --- | --- |
| **Style** | **Description** |
| "*Blue italic text*" | Text of choice, but without any HTML markups or other formatting features |
| "*Light blue italic text*" | Text of choice, HTML markups allowed for formatting purposes |
| "Green text" | Required identifiers |
| "Light green text" | Optional identifiers |
| *Light gray italic text* | Remarks (information only in this manual) |
|  |  |

For various descriptions arrays may be required or at least optional.

Arrays for descriptive contents facilitate describing long text. Text in different array elements do not imply new line or new paragraphs. For HTML contents, specify <br/> for new lines.

JSON standards require all text to be in quotation marks.

### General Description

All procedure and function must be used using the **documentation scheme** "Function Description". This scheme provides clear rules to structure the function description.

The descriptive items may be in any order (except parameters as they must be described in the given sequence).

Syntax:

"*Title of generic description*" :

{

"Documentation": "General Description", // Required value

"Feature Names": [ "*name 1*", "*name 2*", *etc.* ],

"Keywords": [ "*keyword 1*", "*keyword 2*", *etc.* ],

"Library": "*Library name*",

"Version": "7.00"

"Description 01 *text*": [ "*html contents*", " *html contents* ", *etc.* ],

"Examples 01": [ "*contents*", "*contents*", *etc.* ]

"Output 01": [ "*contents*", "*contents*", *etc.* ]

"Description 02 *text*": [ "*html contents*", " *html contents* ", *etc.* ],

"Examples 02": [ "*contents*", "*contents*", *etc.* ]

"Output 02": [ "*contents*", "*contents*", *etc.* ]

… etc …

"Restrictions": [ "*html e.g. No indirect parameter passing*", *etc.* ],

"OS differences": [ "*html contents.details about deviating behavior in different OS*", *etc.* ],

"Exceptions": [ "*html contents*", "*html contents*", *etc.* ]

"See also": [ "*contents*", "*contents*", *etc.* ]

"Notes": [ "*html contents*", "*html contents*", *etc.* ]

},

"*Title of next generic description*" :

{

*( Additional contents like above )*

}

### Function Descriptions

All procedure and function must be used using the **documentation scheme** "Function Description". This scheme provides clear rules to structure the function description.

The descriptive items may be in any order (except parameters as they must be described in the given sequence).

Syntax:

"*Title of function description*" :

{

"Documentation": "Function Description", // Required value

"Function Names": [ "*name 1*", "*name 2*", *etc.* ],

"Keywords": [ "*keyword 1*", "*keyword 2*", *etc.* ],

"Library": "*Library name*",

"Synopsis": [ "*html contents, e.g. way how function is called*", *etc.* ],

"Version": "7.00"

"Description": [ "*html contents*", " *html contents* ", *etc.* ],

"Call as": "*procedure or function*",

"Restrictions": [ "*html e.g. No indirect parameter passing*", *etc.* ],

"OS differences": [ "*html contents.details about deviating behavior in different OS*", *etc.* ],

"Parameter count": "*1*",

"Parameters":

[ { "Number": "1.",

"Types": [ "*Boolean*", "*Numeral*", *etc.* ],

"Name": "*Name of input parameter*",

"Direction": "*input*",

"Description": [ "*html contents*", "*html contents*", *etc.* ]

"Default value": "*contents*"

},

{ "Number": "2.",

*( Additional contents like above )*

}

]

"Return value":

[ { "Name": "*Name of input parameter*",

"Type": "*Boolean*",

"Description": [ "*html contents*", "*html contents*", *etc.* ]

},

{ "Name": "2.",

*( Additional contents like above )*

}

]

"Exceptions": [ "*html contents*", "*html contents*", *etc.* ]

"Examples": [ "*contents*", "*contents*", *etc.* ]

"Output": [ "*contents*", "*contents*", *etc.* ]

"See also": [ "*contents*", "*contents*", *etc.* ]

"Notes": [ "*html contents*", "*html contents*", *etc.* ]

},

"*Title of next function description*" :

{

*( Additional contents like above )*

}

### Convenient Syntax

JSON looks more like a programming language than free text and makes documentation contents quite sophisticated. A convenient syntax has been included to allow one or multiple rows of free text without quotation marks and commas at every line. This feature allows pasting and editing longer text blocks, example code, etc, and overall readability is improving.

Follow some basic rules in order use convenient syntax rules:

* Convenient syntax begins with a descriptor **ending with two consecutive colons and no further text behind.**Example: "Description" ::
* Lines below are payload texts. Quotation marks will be treated as part of the text.
* Sections with code example (e.g. after descriptors "Example", "Output") automatically recognize tabs and multiple spaces which will not be discarded.
* HTML markups like <b>…</b> or inserting links, images, etc, are supported.
* Every line generates one item in the JSON array.
* Comments (like "//" are part of the payload data")
* Payload texts end if one of the following conditions are met:  
  - Another descriptor as described above (with 2 consecutive colons :: at the end)  
  - Line begins with 3 plus signs (+++). JSON code may follow immediately afterwards, typically commas, closing brace '}' and brackets ']' typically follow next according to JSON syntax.

Example:

"*Title of function description*" :

{

Documentation:

Function Description

Function Names:

name 1

name 2

Description:

This is the 1st row of text.

This is the 2nd row of text.

Every row will be put into a separate array element.

Note:

In the statement 'Note:': Add a blank behind the coon.

Blank rows inbetween will translate to HTML &gt;br/&lt;

Contents end either with next descriptor or lines beginning

eiter with " , (comma) } or ] or end of text block

+++ ,

"Version": "7.00", // Begins with comma

etc.

|  |  |  |
| --- | --- | --- |
| **Descriptor** | **Description** | **Arrays** |
| "*Title of function description*" | Use a unique title which has not been used elsewhere. | n/a |
| "Documentation" :  "Function Description" | Must specify "**Function Description**" to identify that this structure is describing a procedure and/or a function and applies the rules listed below: | No |
| "Function Names" | Provide 1 or more function names.  You are allowed to describe multiple similar functions together. | Optional |
| "Keywords" | List of keywords. Will be used to facilitate searching | Required |
| "Library" | Specify library if needed, e.g. "Office Library". | No |
| "Synopsis" | Describe how the function is called. | Optional |
| "Version" | Earliest Beyond4P version supporting this function | No |
| "Description" | Describe the behavior of the function. Use arrays for multiple lines. | Optional |
| "Call as" | Rules if function is to be called as procedure, function (inside an expression), or both | No |
| "Restrictions" | Specify any applicable restrictions | Optional |
| "OS differences" | Specify any operating-system specific deviating behavior | Optional |
| "Parameter count" | Specify number of parameters. Value must be in text form. You are allowed to specify "1 or 2", etc. | No |
| "Parameters" | Describe the parameters (if applicable) | Required |
| "Number" | Parameter number. Use text form to specify "1 or 2" where needed | No |
| "Name" | Name of parameter | No |
| "Types" | List the supported parameter types  See separate table | Optional |
| "Direction" | Parameter passing direction:  "input" "output" "io" "reference" Must specify a variable being referenced "comparison " Code piece "RH expression" " "LH expression" " "statements" " | No |
| "Description" | Describe the parameter. | Optional |
| "Default value" | Describe the default value which applies if the parameter is not specified. | No |
| "Return value" | Specify the return value (if appliable). Use multiple entries if function may return values of different types. | Required |
| "Name" | Name of return value | No |
| "Types" | Variable type returned (e.g. numeral) | Optional |
| "Description" | Describe the return value | Optional |
| "Exceptions" | Exceptions | Optional |
| "Examples" | Code examples | Optional |
| "Output" | Output of code examples  To run the code example listed above automatically and insert the text output in here, then specify  **"Output" : "automatic".**  Otherwise, it's hand-written output. | Optional |
| "See also" | Specify further function names.  Beyond4P will automatically generate links to their function descriptions. | Optional |
| "Notes" | Optional: Further notes | Optional |
|  |  |  |

## Types

|  |  |  |
| --- | --- | --- |
| **Code** | **Type Designation** | **Remarks** |
| A | "all types" | All types |
| B | "boolean" |  |
| C | "table columns" |  |
| D | "date" |  |
| E | "all" | Forces conversion to da te |
| F | "numeral or date" |  |
| G | "numeral or date or blank literal" |  |
| H | "numeral or literal or date or boolean" | valid types except parameter set |
| I | "numeral or boolean" |  |
| J | "date or literal" |  |
| K | "numeral or parameter set of numerals" |  |
| L | "literal" |  |
| M | "all types" | The type is converted to literal |
| N | "numeral" |  |
| O | "all types" | Converted to numeral |
| P | "parameter set" |  |
| Q | "parameter set or literal or numeral" | Softquoted literal with commas inside translate into multiple parameters  Softquoted '' translates to empty parameter set |
| R | "parameter set " | Like P, Members are converted to literal |
| S | "parameter set or literal or numeral" |  |
| T | "parameter set or literal" | Take both types, but no conversion made |
| U | "numeral or literal" |  |
| V | "valid types" |  |
| W | "numeral or literal or date" |  |
| X | "valid except boolean" | Like W, but includes parameter set |
| Y | "numeral or blank literal" |  |
| Z | "numeral or blank literal" | Blank literals are converted to 0 |
| 1 | "comparison expression" |  |
| 2 | "expression" |  |
| 3 | "variable", ":literal" |  |
| 4 | "statements" | 1 or more statemetns |

# Markdowns

## Itemizations

2-level itemizations using filled and unfilled round bullet symbols are supported.

First level items must begin with one \* followed by space. 2nd level items must begin with \*\* followed by space. White spaces before the asterisk aer allowed.

\* Countries

\*\* France

\*\* Spain

\* States

\*\* Ohio

\*\* Quebec

## Tables

Text blocks may contain tables. A dedicated markdown syntax allow easy definition of tables. Limit tables of total 1000 pixel width. Start the table on a new row with === symbol followed by column widths and optionally table classes and style instructions. 'noheader' suppresses creating a header row with different format (e.g. bold text) with the data in the 1st row.

=== 100, 200, 500, noheader, noframe\_1stcolbold

Column 1 in bold | column 2 | column 3

Column 1 in bold | column 2 | column 3

Column 1 in bold | column 2 | column 3

Column 1 in bold | column 2 | column 3

===

Multiple table classes can be inherited. In this case, list the classes (as defined in style.css file ) with space as separator, not comma.

Available class names:

|  |  |  |
| --- | --- | --- |
| **Class** | **Description** | **Remarks** |
| **noheader** | 1st row is considered part of contents | This is symbol is treated by Beyond4P and is not a CSS class to format HTML contents |
| **noframe** | No frames |  |
| **bold\_1st\_column** |  |  |
|  |  |  |

## Markdown for Hyperlinks

All keywords in "see also" will be checked for references to other pages. If yes, then the first available cross-reference will be initiated with a hyperlink.

Inside any text (except code), specific keywords may be checked for available cross-referencing. If found, then they will be associated with hyperlinks. It will check for headers, feature names (includes function and procedure names) and all keywords.

Use 2 consecutive underscore symbols to encapsulate a term which should be hyperlinked. the underscore symbols will vanish in the output document.

The Beyond4P function \_\_table consolidate\_\_ provides simple means to reduce a lot of data in many rows to condensed information you want to have and is easy to understand.

Special case: Hyperlinks to function names

This markdown \_\_literal\_\_ refers to the definition of the literal data type.

This markdown \_\_literal(\_\_) refers to the definition of the function name called literal

## Template

/\* B4PDOCU.START

"Function Title" :

{

"Function Names": [ "...", "..." ],

"Documentation": "Function Description",

"Keywords": [ "optional", "..." ],

"Library": "optional",

"Version": "optional",

"Synopsis": [ "fuction name (...);" ],

"Description": [

"... ",

"... " ],

"Call as": "procedure or function",

"Restrictions": "No indirect parameter passing",

"OS differences": "None",

"Parameter count": "1",

"Parameters": [

{

"Number": "..",

"Name": "...",

"Direction": "input,output,io,reference,comparison,RH expression,LH expression,LH variable, statements",

"Types": [ "literal,..." ],

"Description": [

"...",

"… " ],

"Default value": "optional"

}

],

"Exceptions": "optional",

"Examples": [

"optional",

"..." ]

"Output": "optional or automatic"

"See also": [ "optional", "..." ],

"Notes": [ "optional", "..." ],

}

B4PDOCU.STOP \*/

# Revolving Table of Contents

The document generation tool will use a revolving table of content scheme to manage all the contents and put them into one sequential and structured order like in a user manual. The docu generator reads the most recent file for existing structure, makes necessary updates and saves the file again.

Input File name: Revolving TOC\*.csv (most recent file available)

Output File name: Revolving TOC YYYY-MM-DD hh.mm.ss.csv

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Status** | **Category** | **Level** | **Section Nr** | **HTML File Name** | **Section Name** (used as unique identifier) | **Feature Names** | **Keywords** | **Remarks** |
| OK | Body | 1 | Body 1 | functions\_trig.html |  | sin,cos,tan | trigonometric functions, sine, cosine, tangent | This is a remark |
| Check | Appendix | 2 | Appendix 1.1 |  |  |  |  |  |

**Status** is updated by the tool and can take following values:

New Has been added automatically (Level and section number still blank)

Missing Specified Section Name cannot be matched with an existing document Section

OK Confirmed entry and used for docu generation.

Check E.g. Level missing

**Category**: Maintained by user, used by tool

Intro Beginning Part

Body Document Body

Appendix Appendix Part

**Level**: Maintained by user, used by tool

x

**Section Nr**.: Maintained by the tool

Section numbers will be derived from the level and sequence.

Will only be generated if a level is specified (0 or blank: None, 1, 2, 3 = 1, 1.1, 1.1.1) etc.

**HTML File Name**: Maintained by the tool

**Section Name:** Maintained by both user and the tool (both can update)

**Feature Names:** Maintained by the tool

**Keywords**: Maintained by the tool

**Remarks:** Maintained by the user, and not processed by the tool

# Further References

<https://stackoverflow.com/questions/9725675/is-there-a-standard-format-for-command-line-shell-help-text>

Syntax rules