LBT-ADOPT
TECHNICAL REPORT

Doc.No : AdOptSW.002
Version : 1.1
Date : 29 July 2006

DPU User's Manual

Luca Fini

INAF - Osservatorio di Arcetri
http://lbtwww.arcetri.astro.it/adopt/

Doc.No : AdOptSW.002 Version : 1.1 Date : 29 July 2006	**DPU User's Manual**

ABSTRACT

The Document Processing Utility (DPU) is a Python procedure which processes source files and generates a LaTeXdocument which contains properly formatted annotated code. The LaTeXdocument is further processed to optionally create either a printable (PDF) or a browseable (HTML) version of the document.
The output document is generated by processing special comment lines from the source code which contain LaTeXcommands plus a few special commands.

Contents

1 Introduction

A key feature of well written software is a proper documentation which will help both developers and maintainers to understand what the software does, and how. Writing well documented software, as D. Knuth pointed out years ago [1, 2], is an art closer to literature than to programming, and art is usually hardly enforceable.

By adopting the principles of "literary programming" we will ensure proper synchronization among the various elements which make up the software system: executable code, software documentation, user manuals and so on.

The DPU utility owes a lot to Knuth's Web program, but tries also to extend the concept to other areas, such as API documentation, which is an important feature for any large software system. Moreover it also copes with some everyday's programmer problems, such as multiple programming languages.

2 How to set up a software package

In order to use DPU for generating software documentation you need a few things:

- The **source files**. These are simply the file you use to write your program, or package. In the simplest case you may think of a set of source files stored together in the same directory.

  In order to use DPU features, source files must contain properly formatted comment lines. Section 3 will show how to write DPU oriented comments to source files.

- A driver file. The driver file is simply a LaTeX source file (but you must use the file extension `.dpu` to distinguish it from plain LaTeX files) which is needed to add to the final document pieces of text which is not pertinent to any source file and to include source files in the proper order (which is usually different from the alphabetical order). Section **??** will describe in detail how to write the driver file.

3 Commands in source files

Formatting instructions for the final document are contained in comment lines for the source language used. The main effect of special comments in the source file is to modify how the text is formatted switching between a *code* format and a *plain text* format. Code is displayed in a non proportional font, respecting new lines, with smaller character size. Plain text is displayed as usual for papers or reports and all LaTeX features and commands may be used.

As a further capability, special formatting blocks may be defined to properly display language elements such as routines, functions, classes, methods and so on.

A special comment block in the source file is defined by any number of comment lines starting enclosed between a *start line* and an *end line*.

A block start line is a comment line, obviously starting with the proper comment start character which depends on the programming language used, followed by one or two '@' characters, followed by a *Block type* designator, optionally followed by an option code (enclosed in parentheses), a colon and a block name.

If the special comment flag is a single '@' the header initiates a new section in the output document, if a double '@@' is used, the header corresponds to a new subsection in the output document. This allows more flixibility in the structure of the document.

Here follows a few examples of block start lines:

`#@C`	Generic comment block, Python language.
`//@Function{API}: Hsearch`	API function header block, C++ language.
`#@@Method: search`	Method header, second level structuring
`;@File: hash.pro`	File header, IDL language.

The block end line is a comment line with only the '@' character, as in the examples:

`;@`	Block end line, IDL language.
`//@`	Block end line, C++ language.

A plain comment block (see below) has no further structure. All other blocks must include a *short description line* which is the first non empty comment line follwing the block start line.

In the following pages you'll find a list of defined blocks with detailed explainations.

3.1 Generic comment

Whenever you want a block of text appear as a paragraph (as opposet to source code) in your output document, you may use a generic comment block, as follows:

```
//@C
//
//  The text in this paragraph will be formatted by
// \LaTeX\  as plain text. All \LaTeX\  commands may be used
// and will have the standard meaning.
//
//@
```

Generic comment blocks are initiated by the `@C` special comment line.

You may also note that in the block body a LaTeX command (i.e.: `\LaTeX`) is used. In the document generated as output this block of text is processed as plain text by LaTeX so that the result will be the same as in usual documents.

3.2 Block type: File

The file comment block **must** be present at the beginning of every source file. If a source file is processed which doesn't include the file block, an error message is issued. The file block must be at the very beginning of the file (except for scripting languages for which the first line must include the indication of the processing program).

Here follows a few examples:

```
//@File: hash.c
//
//  Hash management routines.
//
// The code in this file has been derived from the original
// source code by: Jerry Coffin, with improvements by HenkJ
// and Wolthuis.
//@
```

This is the file header of a C/C++ program. Keywords (such as "file", "function", and the like) are **not** case sensitive, but case in short description and in the rest of the comments is respected.

```
#!/usr/bin/python
#@File:   dpu.py
#
# Documentation Processing Utility
#
# This utility processes a \LaTeX\  source file (in the following referred to
# as the driver file) which contains usual \LaTeX\  formatting commands
# and a few special commands which may be used to include
#@
```

The file header for a Python script. As you may note the very first line is the standard indication of the processing program and it is immediately followed by the file block start line.

3.3 Block type: Function

This header is used to describe functions (i.e.: routines returning values), the keyword may be specified with the option `API` indicating that the function is intended to be part of an Application Programmer's Interface and proper documentation for the programmer will be generated.

Special comment syntax may be used for the function declaration with related arguments, so that entry points can be processed by DPU to generate API documentation.

Here follows an example:

```
// @Function{API}: SetSeqNum

// Sets the Sequence number field of a Message
//
// This routine writes an assigned sequence number in the proper field
// of a MsgBuf structure.
// The other fields of the structure remain unaffected.
//@

int SetSeqNum(int SeqNum,        // @P{SeqNum}: Sequence number
      MsgBuf *msgb)      // @P{msgb}: Message structure
                                 // @R: completion code
```

As you may see in the example special comments must be used to describe arguments and the return value of the function.

3.4 Block type: Procedure

The Procedure block type is very similar to Function, but is used for routines which do not return values to the caller. All details are as described for the Function block, except that the `@R` special comment is of no use.

3.5 Block type: Class

Class definition block, it is structurally equivalent to a Function block.

3.6 Block type: Method

Method definition block, it is structurally equivalent to a Function block. In order to generate meaningful documents in output, it is required that header blocks (and thus the source code, too) for methods of a class are written just after the class declaration.

Because methods are associated with the class, you will usually mark them with the second level structure marker. i.e.:

```
#@Class: HdrTree

# Class to manage the list of headers

# This class is used to manage .....
```

```
  .... more comments and code ...

#@@Method: add

# Adds a header in the proper position in header tree

# Adds info related to ....

  .... more comments and code ...

#@@Method: newID

# Generates a unique ID

#@
```

3.7 Special comments for arguments

When commenting functions, procedures and the like, a specific syntax must be used to specify arguments. This will allow to generate documents to describe API's to be used by programmers who are using libraries of subprograms or classes.

The special syntax is used to describe arguments and possibly return values as in the following table:

`@Pname:`	description of argument named *name*.
`@R:`	description of return value.

4 The driver file

The process of generating the annotated source document is driven by a single file whose name must end in `.dpu`, also referred to as the "driver" file. The driver file contains essentially LaTeX markup and text plus a few special commands used to assemble source files together.

4.1 The `\sourceinclude` command

The `\sourceinclude` command is used to include in the text the content of a source file.

The source file is read from `dpu` and properly formatted in order to generate the final document[1].

Any number of `\sourceinclude` commands may be used to include source files.

4.2 The `\makelist` command

The `\makelist` command will generate in the place where it is found in the driver file, a list of the content of all source files included.

5 Running DPU

The `dpu` utility is run from the command line where the *driver* file must also be specified. A number of option can be provided to modify the default behaviour:

```
Usage:
```

[1] The process of generating the final document will usaully require several steps and is controlled by the `dpu` command options.

```
dpu.py [-pdf] [-ps] [-html] [-t] [-k] [-n] [-v] <drv>

where:

  <drv>  Input file name (without the .dpu extension).

  -html  Generates an HTML version of the document
  -pdf   Generates a PDF printable version of the document
  -ps    Generates a PostScript printable version of the document
  -t     Output the tree structure of the document (other output options
         are ignored).
  -k     Keep (do not delete) temporary files
  -n     Code listing with line numbers
  -v     Verbose mode (adding more -v increases vebosity)
```

A Installing DPU

A.1 Prerequisites

In order to use DPU you must have working support for the following:

- Python interpreter (version 2.2 or greather is required).
- LaTeX
- If you want to generate HTML documents you must also have `LaTeX2html`.

A.2 Installing the program

DPU is written in Python and is provided with an installation kit in the pythonic way.
In order to install DPU you just have to extract files from the distribution archive into any directory and then use the command:

```
python setup.py install
```

Please note that you usually will need root privileges.

References

[1] Knuth, D. E., "Literate Programming". *The Computer Journal*, 27, 2, May 1984, pp. 97-111.

[2] Knuth, D. E., "Literate Programming". (CSLI Lecture Notes, no. 27.)

[3] Knuth, D.E., Levy, S. "The CWEB System of Structured Documentation". Addison-Wesley, 1993.