

Lua2D_OX_{lua}

0.2

Copyright (c) 2012-13 Simon Dales

7th February 2013

This is a hack to enable `doxygen` to document lua.
It uses the well-known `doxygen` filter mechanism to allow `doxygen` to read
extra computer languages.
If this works for you then enjoy.

Contents

1	Introduction	2
1.1	Literate Programming	2
1.2	Autodoc	2
1.2.1	Doxygen	2
1.2.2	Lua2D _O X _{lua}	3
1.3	Origins	3
2	Installation	4
2.1	System requirements	4
2.1.1	Luatex/lua	4
2.1.2	doxygen	4
2.2	Linux/other Unices	4
2.3	Mac OS X	4
2.4	Windows	4
3	Useage	5
3.1	Learn to use Doxygen	5
3.2	running Lua2D _O X _{lua}	5
3.2.1	make Doxyfile	5
3.2.2	run doxygen	5
3.3	In use	5
4	Design	6
4.1	Classes	6
4.1.1	Methods	6
4.2	Multiline function declarations	6
4.3	Multiple magic comments	6
5	ChangeLog	7

1 Introduction

Documenting sourcecode is a hassle. There are two basic methods of providing documented sourcecode: “literate programming” and “autodoc”. Without these tools a human has to keep the documentation and code aligned, which rarely happens.

1.1 Literate Programming

If one is sufficiently organised, and it fits in with your existing work practices, then there is a lot going for literate programming. Here you write a combined document that contains both code and documentation fragments. When you want code or documentation you run it through a program which makes the appropriate files.

This is theoretically the neatest way of making code because no code is generated without documentation. The best known example of literate programming is the source of \TeX .

1.2 Autodoc

In the real world we get source from many sources, usually where literate programming wasn’t used or isn’t available.

One solution is to use an automatic documentation tool: feed it sourcecode and it will extract what documentation it can.

If you feed it raw source then all it can do is tell you where it found features such as function and class declarations.

If your source has been suitably prepared, then more detailed documentation can be produced. This preparation takes the form of “magic comments”. These are source comments that contain “magic” features that let the autodoc tool know to make use of them.

1.2.1 Doxygen

Doxygen is one of the many autodoc tools use to document C-like languages. It is widely used and supported in the linux world and has Mac OS X and Windows ports.

```

83 void helloWorldOnce(){
84     printf("hello_world\n");
85 }
86 /*!
87  | brief writes hello world *n
88
89  | This is a demo of how to loop round
90  | and print something
91
92  | param N number of times round
93 */
94 void helloWorldN(int N){
95     for (int i=0;i<N;i++)
96         printf("hello_world_%d\n", i);
97 }
```

```

void helloWorldN(int N)
writes hello world *n
This is a demo of how to loop round and print something
Parameters:
    N number of times round
Definition at line 94 of file hello.cpp.

void helloWorldOnce()
Definition at line 83 of file hello.cpp.
```

1.2.2 Lua2D_OX_{lua}

In the case of `doxygen`, which is intended for C-like languages, it expects ‘`///`’ or ‘`///
 comments and ‘/**...*/’/‘/*!...*/’ for C-style comments. For Lua2DOXlua we recycle this convention and use ‘--!’ and ‘--[[!...]]’.`

The internal format of the magic comments is identical to `doxygen`. This is not suprising in that `lua2dox_filter` is a `doxygen` filter. What it does is to read lua code and output something not entirely unlike C++.

This lets us make Doxygen output some documentation to let us develop this code.

```

83 function helloWorldOnce()
84     io.write("hello_world\n")
85 end
86 --///  

87     \brief writes hello world *n
88
89     This is a demo of how to loop round
90     and print something
91
92     \param N number of times round
93     //
94 function helloWorldN(N)
95     for i=0,N do
96         io.write("hello_world" .. i .. "\n")
97     end
98 end

```

```

function helloWorldN(N)
    writes hello world *n
    This is a demo of how to loop round and print something
    Parameters:
        N number of times round
    Definition at line 94 of file hello.lua.

function helloWorldOnce()
    Definition at line 83 of file hello.lua.

```

1.3 Origins

Lua2D_OX_{lua} inspired by the functionality of a Perl module called “lua2dox”¹. Found on CPAN when looking for something else; kinda handy.

Improved from lua2dox to make the `doxygen` output more friendly. Runs faster in lua than in Perl.

Because this Perl based system is called “lua2dox”, I have decided to add “_lua” to the name to keep the two separate, so it is called “lua2dox_lua”.

¹<http://search.cpan.org/~alec/Doxygen-Lua-0.02/lib/Doxygen/Lua.pm>

2 Installation

2.1 System requirements

Should run on any reasonably modern computer (PC or Mac OS X).

Requires a reasonably recent: `doxygen` and `luatex` or `lua`.

- `doxygen` \geq 1.8.1
- `luatex` \geq 0.6 / `lua` 5.1

2.1.1 Luatex/lua

Whilst `luatex` isn't strictly required to make it run; T_EX users are more likely to have a recent version (via `texlive`) of `texlua` rather than relying on your, possibly aging, distro's `lua`.

It should work with either `texlua` or `lua`. The calling bash script determines which one is available (`texlua` then `lua`) and selects that one.

2.1.2 doxygen

Install this using whatever mechanism you usually use.

It is also helpful if you know how to use `doxygen`.

2.2 Linux/other Unices

Only tested for linux, but should work for any Unix-like OS.

For other OS you need to figure it out yourself. If you do, then please email me with the code/suggestions.

1. make an appropriately named temporary directory ("`tmp`").
2. `cd tmp`
3. `untar` into this directory.
4. manually check if `lua2dox.lua` runs on your system.
From the commandline run "`./lua2dox_filter --help`". If it runs and produces a sensible result then it probably works.
5. Manually pick an appropriate directory for this system.
Make this new directory and `copy/mv` all these files into there.
If you are installing it as a personal app then put it somewhere in your userspace. For system-wide: ask your sysadmin.
In this example called "`~/programs/lua2dox`".
6. "`cd ~/programs/lua2dox`".
7. Run "`./install.sh`". This makes the links to the active code.
Default is to put in '`~/bin`'. For system-wide: ask your sysadmin.

2.3 Mac OS X

Being a unix-like OS then install as for Linux.

2.4 Windows

As for unix but use the `.BAT` files.

3 Useage

Because this is a `doxygen` filter then it is assumed that the endusers are familiar with the use of this app. When familiar then one can move on to documenting lua code.

3.1 Learn to use Doxygen

Ensure `doxygen` is installed on your system and that you are familiar with its use. Best is to try to make and document some simple C/C++/PHP to see what it produces.

Included with this distribution are some examples of php and lua code. They are two simple programs that create some classes and then show their use. The documentation should be nearly the same as each other.

3.2 running Lua2D_OX_{lua}

3.2.1 make Doxyfile

Run “`doxygen -g`” to create a default `Doxyfile`. This is the configuration file that `doxygen` needs for making the documentation.

Then alter it to let it recognise lua. Add the following lines:

FILE_PATTERNS	=	*.lua
FILTER_PATTERNS	=	*.lua=lua2dox_filter

You might also like to edit some other entries:

PROJECT_NAME	=	<name of project>
PROJECT_NUMBER	=	<version number>
OUTPUT_DIRECTORY	=	docs
SOURCE_BROWSER	=	yes
GENERATE_LATEX	=	no

For example:

PROJECT_NAME	=	"A_Test_Project"
PROJECT_NUMBER	=	"0.1_20120926a"

Either add them to the end or find/edit the appropriate entry in `Doxyfile`.

3.2.2 run doxygen

Once `Doxyfile` has been edited run `doxygen` and `Lua2DOXlua` will be called as a filter.

3.3 In use

As with `doxygen` the HTML output is the most useful for development. When the project is finished then PDF output (via `LATEX`) may be useful.

Typically you will have some source files open in your editor/IDE. Also keep a web-browser window open pointing at the documentation.

After a bit of editing the documentation will become stale. At this point run `doxygen` and refresh the HTML pages.

4 Design

Lua2D_OX_{lua} is a `doxygen` filter. It has a basic parser that will read each line of lua and output an equivalent in pseudo-C++. It loops round until it has found the end of the file. It only has to be good enough for `doxygen` to see it as legal. Therefore our lua interpreter is fairly limited, but “good enough”.

The parser is relatively crude and assumes that the programmer writes lua in a fairly clean way. Some bits of lua will confuse it.

In order for the source browser to work in the `doxygen` output it must preserve the number of lines. The output of each line is either written as-is or a converted line. When `doxygen` re-reads the source to make the source-HTML/L^AT_EX/... it uses the line number to set the position.

It only supports comments that preceded the function/class declaration. Any in the middle will act as cruft in the resulting documentation. This will be slightly out of place but at least should refer to somewhere near to where it was intended. In particular “\todo”. This will appear in the ToDo list, but not associated with the right function.

4.1 Classes

Class declarations are assumed to be using a “well-known” userdefined function “`class()`”. It processes “`TAA = class(TA)`”_{lua} → “`class TAA: public TA{}`”_{C++}.

4.1.1 Methods

Methods are mangled as: “`TA.fred(this,A)`”_{lua} to “`/*! \memberof TA */ fred(this,A)`”_{C++}. This is a `doxygen` feature introduced in 1.8.1. It allows the filter to work in a single pass.

4.2 Multiline function declarations

Because Lua2D_OX_{lua} can only process one line at a time it cannot be guaranteed to correctly process function declarations with multiline parameter list. This is because the parser isn’t sophisticated enough to guarantee finding the close paren on some random later line.

So I have put in a hack that will insert the “missing” close paren. It looks at the function declaration that you have got and checks for a close paren. If it is not there then it will add a spoof parameter and the close.

The effect is that you will get the function documented, but not with the parameter list you might expect.

```

94 function foo(A
95     ,B
96     ,C)

```

Becomes

```

94 function foo(A, ___MissingCloseParenHere___)

```

4.3 Multiple magic comments

The hacking of lua to C++ by Lua2D_OX_{lua} has the limitation that some magic comments will not get associated with the correct function/class. This is a “feature” that might get correct in a later version.

5 ChangeLog

0.2 2013-02-07

* removed reliance on temporary files, but now requires doxygen \geq 1.8.1.

0.1a 2012-09-26

* Cleaned up documentation.

0.1 2012-07-04

* Initial release