Fizz buzz - LuaX demo

Christophe Delord - http://cdelord.fr/fizzbuzz

Fri Dec 8, 2023

Contents

Chapter 1

Disclaimer

This document is not about Fizzbuzz. This document is a suggestion to simplify the build process of software projects, a demo of an **homogeneous and consistent** development and documentation environment. Fizzbuzz is just an application example.

Chapter 2

Links

- fizzbuzz_slideshow.pdf: PDF slideshow
- fizzbuzz.pdf: PDF demonstration (specification, implementation, tests, test report, documentation generator, ...)
- github.com/CDSoft/fizzbuzz: Sources

Chapter 3

Introduction

Lots of software projects involve various tools, free as well as commercial, to build the software, run the tests, produce the documentation, ... These tools use different data formats and scripting languages, which makes the projects less scalable and harder to maintain.

Sharing data between configuration files, documentations, tests results can then be painful and counter productive (the necessary glue is often more complex than the tools themselves).

Usually people script their build systems and processes with languages like Bash, Python, Javascript and make them communicate with plain text, YAML, JSON, XML, CSV, INI, TOML. Every script shall rely on specific (existing or not) libraries to read and write these data formats.

This document presents a commun and powerful data format and some tools to script the build process of a project and generate documentation.

To sum up the suggested solution is:

- a **single data format**
- and a **reduced set of highly configurable tools**.

Chapter 4

Lua[1]

Lua is the perfect candidate for both a common data format and a script language.

4.1 What is Lua?

Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode with a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

4.2 Why choose Lua?

Lua is a proven, robust language

Lua has been used in many industrial applications (e.g., Adobe's Photoshop Lightroom), with an emphasis on embedded systems (e.g., the Ginga middleware for digital TV in Brazil) and games (e.g., World of Warcraft and Angry Birds). Lua is currently the leading scripting language in games. Lua has a solid reference manual and there are several books about it. Several versions of Lua have been released and used in real applications since its creation in 1993. Lua featured in HOPL III, the Third ACM SIGPLAN History of Programming Languages Conference, in 2007. Lua won the Front Line Award 2011 from the Game Developers Magazine.

[1] from https://www.lua.org/about.html

Lua is fast

Lua has a deserved reputation for performance. To claim to be "as fast as Lua" is an aspiration of other scripting languages. Several benchmarks show Lua as the fastest language in the realm of interpreted scripting languages. Lua is fast not only in fine-tuned benchmark programs, but in real life too. Substantial fractions of large applications have been written in Lua.

Lua is portable

Lua is distributed in a small package and builds out-of-the-box in all platforms that have a standard C compiler. Lua runs on all flavors of Unix and Windows, on mobile devices (running Android, iOS, BREW, Symbian, Windows Phone), on embedded microprocessors (such as ARM and Rabbit, for applications like Lego MindStorms), on IBM mainframes, etc.

Lua is powerful (but simple)

A fundamental concept in the design of Lua is to provide meta-mechanisms for implementing features, instead of providing a host of features directly in the language. For example, although Lua is not a pure object-oriented language, it does provide meta-mechanisms for implementing classes and inheritance. Lua's meta-mechanisms bring an economy of concepts and keep the language small, while allowing the semantics to be extended in unconventional ways.

Lua is small

Adding Lua to an application does not bloat it. The tarball for Lua 5.4, which contains source code and documentation, takes 353K compressed and 1.3M uncompressed. The source contains around 30000 lines of C. Under 64-bit Linux, the Lua interpreter built with all standard Lua libraries takes 281K and the Lua library takes 468K.

Lua is free

Lua is free open-source software, distributed under a very liberal license (the well-known MIT license). It may be used for any purpose, including commercial purposes, at absolutely no cost. Just download it and use it.

Chapter 5

LuaX

LuaX is a Lua interpreter and REPL based on Lua 5.4, augmented with some useful packages. LuaX can also produce standalone executables from Lua scripts.

LuaX runs on several platforms with no dependency:

- Linux (x86_64, i386, aarch64)
- MacOS (x86_64, aarch64)
- Windows (x86_64, i386)

LuaX can cross-compile scripts from and to any of these platforms.

LuaX comes with a standard Lua interpreter and provides some libraries (embedded in a single executable, no external dependency required):

- LuaX interactive usage: improved Lua REPL
- F: functional programming inspired functions
- fs: file system management
- sh: shell command execution
- mathx: complete math library for Lua
- imath: arbitrary precision integer and rational arithmetic library
- qmath: rational number library
- complex: math library for complex numbers based on C99
- ps: Process management module
- sys: System module
- crypt: cryptography module
- lz4: Extremely Fast Compression algorithm
- lpeg: Parsing Expression Grammars For Lua
- linenoise: light readline alternative
- luasocket: Network support for the Lua language
- inspect: Human-readable representation of Lua tables

More information here: http://cdelord.fr/luax

Chapter 6

Scripting with LuaX

LuaX can be used as a general programming language. There are plenty of good documentations for Lua and LuaX.

A big advantage of Lua is the usage of Lua tables as a common data format usable by various tools. It is Human-readable and structured. It can be generated by Lua scripts but also by any software producing text files.

Typical usages are:

- project/software configuration
  - a Lua table can be used to describe a project or a software configuration
    - read by an embedded Lua interpreter
    - used to generate documentation or source code
- tests results
  - a test suite can generate test results as a Lua table
  - tests results can be used to render documentation (tests reports) and compute a test coverage

The next chapters present some tools written in Lua/LuaX or using Lua as a scripting engine.

Chapter 7

Bang

Bang is a ninja file generator scriptable in LuaX, a Lua interpreter with a bunch of useful modules (file management, functional programming module, basic cryptography, ...). It takes a build description (a LuaX script) and generates a Ninja file.

Bang provides functions to generate ninja primitives (variables, rules, build statements, ...) and some extra features:

- rule/build statement pairs described in a single function call
- file listing and filenames list management using LuaX modules (e.g. F and fs)
- pipe simulation using rule composition
- "clean", "install" and "help" targets

Bang comes with an example that shows how to use bang and LuaX functions to:

- discover source files actually present in the repository: no redundant hard coded file lists (redundancy means painful maintenance)
- cross-compile the same sources for multiple platforms: compilation for several platforms without any dirty copy/paste
- describe static libraries: in the `lib` directory, each sub-directory is a library compiled and archived in its own `.a` file
- describe executables: in the `bin` directory, each C source file is the main file of a binary containing this C file as well as libraries from the `lib` directory.

Bang is currently used to build bang itself but also LuaX and some projects available on my GitHub.

Chapter 8

Ypp

Ypp is a minimalist and generic text preprocessor using Lua macros.

Ypp is compiled by LuaX, i.e. Lua and LuaX functions and modules are available in macros.

More information here: http://cdelord.fr/ypp

Ypp is pretty simple. It searches for Lua expressions and replaces macros with their results.

Macro	Result
`@(...)`	Evaluates the Lua expression `...` and replaces the macro by its result
`@@(...)`	Executes the Lua chunk `...` and replaces the macro by its result (if not `nil`)

Some expression do not require parentheses (function calls).

8.1 Example

```
$$
\sum_{i=1}^{100} i^2 = @F.range(100):map(function(x) return x*x end):sum()
$$
```

is rendered as

$$\sum_{i=1}^{100} i^2 = 338350$$

Macros can also define variables reusable later by other macros.

```
@@[[
    local foo = 42
    N = foo * 23 + 34
    local function sq(x) return x*x end
    function sumsq(n) return F.range(N):map(sq):sum() end
]]
```

defines `N` ($N = 1000$) which can be read in a Lua expression or with `@N` and `sumsq` which computes the sum of squares.

Then

```
$$
\sum_{i=1}^{@N} i^2 = @sumsq(N)
$$
```

becomes

$$\sum_{i=1}^{1000} i^2 = 333833500$$

Chapter 9

Pandoc

Pandoc is a swiss-army knife to convert from and to a bunch of document formats.

A big advantage of Pandoc is the ability to use Lua scripts to define custom readers and writers for unsupported formats and also Lua filters to manipulate the pandoc abstract syntax tree (AST). This is the main pandoc feature exercised in this document.

Pandoc has an excellent documentation:

- main pandoc documentation: https://pandoc.org/MANUAL.html
- Lua filter documentation: https://pandoc.org/lua-filters.html

Fizzbuzz uses pandoc Lua filters with Panda (see next chapter) which bundles some useful filters in a single script.

Chapter 10

Panda

Panda is a Pandoc Lua filter that works on internal Pandoc's AST.

It provides several interesting features:

- variable expansion (minimalistic templating)
- conditional blocks
- file inclusion (e.g. for source code examples)
- script execution (e.g. to include the result of a command)
- diagrams (Graphviz, PlantUML, ditaa, Asymptote, blockdiag, mermaid...)

The documentation of Panda is here: http://cdelord.fr/panda

10.1 Examples

There are lots of examples in the documentation of panda. We will see here two of them.

Documentation extraction from source code

The source code can be documented by adding special marks in comments. The documentation shall be written in Markdown. The default mark is `@@@` and can be customized.

For instance, the following C source contains documentation that can be extracted and included to a Pandoc document.

```
/*@@@
**`answer`** takes any question
and returns the most relevant answer.

Example:
``` c
```
```

```
    const char *meaning
        = answer("What's the meaning of life?");
```
```
@@@*/

const char *answer(const char *question)
{
    return "42";
}
```

To extract the documentation, panda provides a macro to replace a `div` element by the documentation chunks from a file. E.g.:

```
:::{doc=deep_thought.c}
:::
```

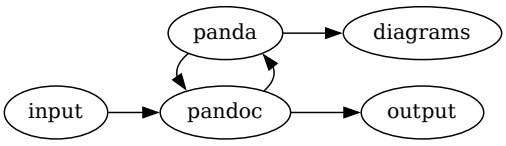
will be replaced by:

> **answer** takes any question and returns the most relevant answer.
>
> Example:
>
> ```
>     const char *meaning
>         = answer("What's the meaning of life?");
> ```

Diagrams

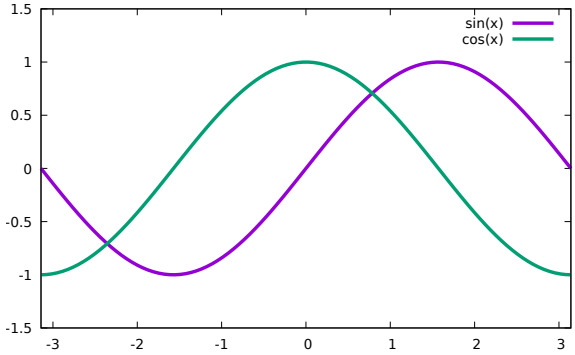
Diagrams can be embedded in Pandoc documents. Diagrams are specified as code blocks and are replaced by an image by panda.

```
```{.dot render="{{dot}}" width=67%}
digraph {
    rankdir=LR;
    input -> pandoc -> output
    pandoc -> panda -> {pandoc, diagrams}
    { rank=same; pandoc, panda }
    { rank=same; diagrams, output }
}
```
```




```
```{render="{{gnuplot}}" width=67%}
```
```

```
set xrange [-pi:pi]
set yrange [-1.5:1.5]
plot sin(x) lw 4, cos(x) lw 4
```



Chapter 11

hey

`hey` is a shell script. It is intended to easily install some tools based on LuaX and Pandoc to pre-process files and generate documents, using Lua as a common, simple and powerful scripting language.

11.1 Example

Easy installation, only `hey` is needed:

```
wget https://raw.githubusercontent.com/CDSoft/hey/master/hey
```

Its usage is very similar to `apt` or `dnf`:

```
$ hey list
all     install all packets

======== CDSoft softwares ========================================================
bang    Ninja file generator scriptable in LuaX
calculadoira
        simple yet powerful calculator
lsvg    LuaX interpreter specialized to generate SVG images
luax    Lua eXtended, a Lua interpreter with a better REPL and useful libraries
panda   Pandoc Lua filter that works on internal Pandoc's AST
tagref  Maintain cross-references in your code
ypp     Yet another preprocessor, scriptable in LuaX

======== Other softwares =========================================================
ditaa   DIagrams Through Ascii Art
pandoc  Swiss-army knife to convert from and to a bunch of document formats
pandoc-latex-template
        Clean pandoc LaTeX template to convert your markdown files to PDF or LaTeX
```

```
pandoc-panam-css
        Pan Am: Simple CSS for Pandoc
plantuml
        PlantUML
typst   Focus on your text and let Typst take care of layout and formatting
```

```
$ hey install all
...
```

Chapter 12

Fizzbuzz

Fizzbuzz is a concrete example of the usage of LuaX/ypp/pandoc/panda to specify and test a software.

12.1 Specification

From Wikipedia:

> Fizz buzz is a group word game for children to teach them about division. Players take turns to count incrementally, replacing any number divisible by three with the word "fizz", and any number divisible by five with the word "buzz".

`fizzbuzz` is a function that returns `"fizz"`, `"buzz"`, `"fizzbuzz"` or `n` for any positive integer `n`.

$$fizzbuzz : \mathbb{N}^+ \to \{fizz, buzz, fizzbuzz\} \cup \mathbb{N}^+$$

$$fizzbuzz(n) = \begin{cases} \text{"fizzbuzz"} & \text{if } (3|n) \wedge (5|n) \\ \text{"fizz"} & \text{if } (3|n) \wedge \neg(5|n) \\ \text{"buzz"} & \text{if } (5|n) \wedge \neg(3|n) \\ n & \text{if } \neg(3|n) \wedge \neg(5|n) \end{cases}$$

12.1.1 Requirements

`SPEC_API`: fizzbuzz command line argument

The fizzbuzz program takes one argument that specify the number for fizzbuzz values to generate.

`SPEC_OUT`: fizzbuzz output on stdout

The fizzbuzz program emits fizzbuzz values on the standard output. Each line contains `n` and `fizzbuzz(n)`.

e.g.:

```
$ fizzbuzz 6
1   1
2   2
3   fizz
4   4
5   buzz
6   fizz
```

`SPEC_FIZZ`: **fizz when n is a multiple of 3 but not 5**

If `n` is a multiple of 3 but not 5, then `fizzbuzz(n)` is `"fizz"`.

`SPEC_BUZZ`: **buzz when n is a multiple of 5 but not 3**

If `n` is a multiple of 5 but not 3, then `fizzbuzz(n)` is `"buzz"`.

`SPEC_FIZZBUZZ`: **fizzbuzz n is a when multiple of 3 and 5**

If `n` is a multiple of 3 and 5, then `fizzbuzz(n)` is `"fizzbuzz"`.

`SPEC_NUM`: **n when n is a not a multiple of 3 and 5**

If `n` is a multiple of 3 and 5, then `fizzbuzz(n)` is `"fizzbuzz"`.

12.1.2 Examples

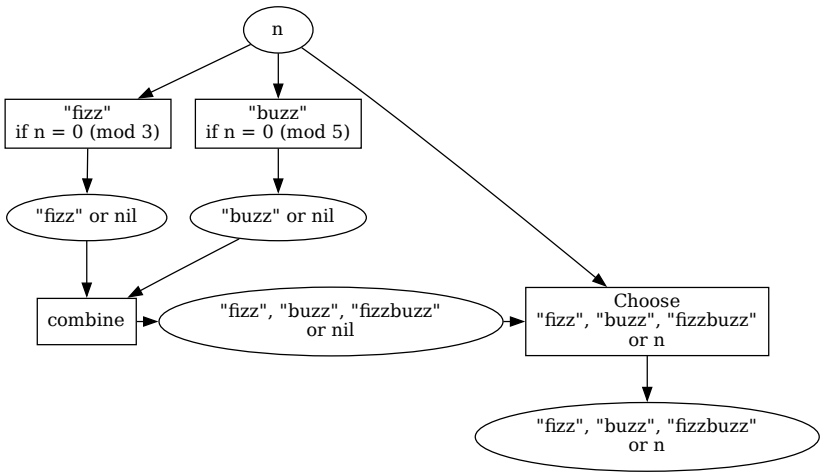
n	fizzbuzz(n)	n	fizzbuzz(n)	n	fizzbuzz(n)	n	fizzbuzz(n)
1	1	6	fizz	11	11	16	16
2	2	7	7	12	fizz	17	17
3	fizz	8	8	13	13	18	fizz
4	4	9	fizz	14	14	19	19
5	buzz	10	buzz	15	fizzbuzz	20	buzz

12.2 Implementation

12.2.1 Lua implementation

The Lua implementation of Fizzbuzz is based on a functional style, using function compositions.

It computes the `"fizz"` and `"buzz"` parts and return them if at least one of them is not `nil`. Otherwise it returns its argument unchanged.




```
local function div(d, s, n)
    return n % d == 0 and s or nil
end

local fizz = F.partial(div, 3, "fizz")
local buzz = F.partial(div, 5, "buzz")

local function combine(a, b)
    return a and (a..(b or "")) or b
end

local function fizzbuzz(n)
    return combine(fizz(n), buzz(n)) or n
end
```

12.2.2 C implementation

The C implementation of Fizzbuzz uses an array of string formats used by `sprintf` to produce `"fizz"`, `"buzz"`, `"fizzbuzz"` or the function argument.

The array index is a 2-bit integer, each bit being the divisilibity of the argument by 3 or 5.

```
const char *fizzbuzz(int i, char *s)
{
    static const char *fmt[] = {
        [0|(0<<1)] = "%d",
        [1|(0<<1)] = "fizz",
```

```
        [0|(1<<1)] = "buzz",
        [1|(1<<1)] = "fizzbuzz",
    };
    const int fizz = (i%3 == 0) << 0;
    const int buzz = (i%5 == 0) << 1;
    sprintf(s, fmt[fizz|buzz], i);
    return s;
}
```

12.2.3 Haskell implementation

The Haskell implementation of Fizzbuzz builds infinite lists of fizzes, buzzes and integers.

The functions `fizzbuzz` builds three infinite lists and combine them.

ns	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
fizzes	.	.	fizz	.	.	fizz	.	.	fizz	.	.	fizz	.	.	fizz	...
buzzes	.	.	.	.	buzz	.	.	.	.	buzz	.	.	.	.	buzz	...

$$fizzbuzz(n) = \begin{cases} fizz + buzz & \text{if } fizz \neq Nothing \vee buzz \neq Nothing \\ n & \text{if } fizz = buzz = Nothing \end{cases}$$

```
fizzbuzz :: [String]
fizzbuzz = zipWith3 combine fizzes buzzes ns
    where
        ws d w = cycle $ replicate (d-1) Nothing ++ [Just w]
        fizzes = ws 3 "fizz"
        buzzes = ws 4 "buzz" -- bug that shall be detected by the tests
        ns = show <$> [1..]
        combine f b n = fromMaybe n (f<>b)
```

12.3 Tests

The results of the Fizzbuzz executables are checked by the test script `fizzbuzz_test.lua`. This script check the fizzbuzz results and produces a Lua table with the test results. This script will later be used to build the test reports.

12.3.1 Test plan

Each fizzbuzz implementation is executed (with 50 values). The results are checked by `fizzbuzz_test.lua` and stored in a Lua table.

The fizzbuzz values are recorded in the `fizzbuzz` field of the test result table.

`TEST_API`: **number of fizzbuzz values**

`SPEC_API`: fizzbuzz command line argument

The fizzbuzz list contains 50 values.

The result of this test is recorded in the `valid_number_of_lines` field of the test result table.

`TEST_OUT`: **output on stdout**

`SPEC_OUT`: fizzbuzz output on stdout

The fizzbuzz list is emitted on stdout.

`TEST_FIZZ`: **"fizz" values**

`SPEC_FIZZ`: fizz when n is a multiple of 3 but not 5

All multiples of 3 but not 5 are `"fizz"`.

The result of this test is recorded in the `valid_fizz` field of the test result table.

`TEST_BUZZ`: **"buzz" values**

`SPEC_BUZZ`: buzz when n is a multiple of 5 but not 3

All multiples of 5 but not 3 are `"buzz"`.

The result of this test is recorded in the `valid_buzz` field of the test result table.

`TEST_FIZZBUZZ`: **"fizzbuzz" values**

`SPEC_FIZZBUZZ`: fizzbuzz n is a when multiple of 3 and 5

All multiples of 3 and 5 are `"fizzbuzz"`.

The result of this test is recorded in the `valid_fizzbuzz` field of the test result table.

`TEST_NUM`: **integral values**

`SPEC_NUM`: n when n is a not a multiple of 3 and 5

All non multiples of 3 and 5 are themselves.

The result of this test is recorded in the `valid_numbers` field of the test result table.

12.4 Test reports

12.4.1 Lua implementation

The Lua fizzbuzz function returns:

1, 2, fizz, 4, buzz, fizz, 7, 8, fizz, buzz, 11, fizz, 13, 14, fizzbuzz, 16, 17, fizz, 19, buzz, fizz, 22, 23, fizz, buzz, 26, fizz, 28, 29, fizzbuzz, 31, 32, fizz, 34, buzz, fizz, 37, 38, fizz, buzz, 41, fizz, 43, 44, fizzbuzz, 46, 47, fizz, 49, buzz

`RES_LUA_API`: **number of fizzbuzz values** [PASS]

`TEST_API`: number of fizzbuzz values

`RES_LUA_OUT`: **output on stdout** [PASS]

`TEST_OUT`: output on stdout

`RES_LUA_FIZZ`: **"fizz" values** [PASS]

`TEST_FIZZ`: "fizz" values

`RES_LUA_BUZZ`: **"buzz" values** [PASS]

`TEST_BUZZ`: "buzz" values

`RES_LUA_FIZZBUZZ`: **"fizzbuzz" values** [PASS]

`TEST_FIZZBUZZ`: "fizzbuzz" values

`RES_LUA_NUM`: **integral values** [PASS]

`TEST_NUM`: integral values

Summary: 5 / 5 tests passed

12.4.2 C implementation

The C fizzbuzz function returns:

1, 2, fizz, 4, buzz, fizz, 7, 8, fizz, buzz, 11, fizz, 13, 14, fizzbuzz, 16, 17, fizz, 19, buzz, fizz, 22, 23, fizz, buzz, 26, fizz, 28, 29, fizzbuzz, 31, 32, fizz, 34, buzz, fizz, 37, 38, fizz, buzz, 41, fizz, 43, 44, fizzbuzz, 46, 47, fizz, 49, buzz

`RES_C_API`: **number of fizzbuzz values** [PASS]

`TEST_API`: number of fizzbuzz values

`RES_C_OUT`: **output on stdout** [PASS]

`TEST_OUT`: output on stdout

`RES_C_FIZZ`: **"fizz" values** [PASS]

`TEST_FIZZ`: "fizz" values

`RES_C_BUZZ`: **"buzz" values** [PASS]

`TEST_BUZZ`: "buzz" values

`RES_C_FIZZBUZZ`: **"fizzbuzz" values** [PASS]

`TEST_FIZZBUZZ`: "fizzbuzz" values

`RES_C_NUM`: **integral values** [PASS]

`TEST_NUM`: integral values

Summary: 5 / 5 tests passed

12.4.3 Haskell implementation

The Haskell fizzbuzz function returns:

1, 2, fizz, buzz, 5, fizz, 7, buzz, fizz, 10, 11, fizzbuzz, 13, 14, fizz, buzz, 17, fizz, 19, buzz, fizz, 22, 23, fizzbuzz, 25, 26, fizz, buzz, 29, fizz, 31, buzz, fizz, 34, 35, fizzbuzz, 37, 38, fizz, buzz, 41, fizz, 43, buzz, fizz, 46, 47, fizzbuzz, 49, 50

`RES_HS_API`: **number of fizzbuzz values** [PASS]

`TEST_API`: number of fizzbuzz values

`RES_HS_OUT`: **output on stdout** [PASS]

`TEST_OUT`: output on stdout

`RES_HS_FIZZ`: **"fizz" values** [FAIL]

`TEST_FIZZ`: "fizz" values

`RES_HS_BUZZ`: **"buzz" values** [FAIL]

`TEST_BUZZ`: "buzz" values

`RES_HS_FIZZBUZZ`: **"fizzbuzz" values** [FAIL]

`TEST_FIZZBUZZ`: "fizzbuzz" values

`RES_HS_NUM`: **integral values** [FAIL]

`TEST_NUM`: integral values

Summary: 1 / 5 tests passed

12.4.4 Lua / C / Haskell comparison

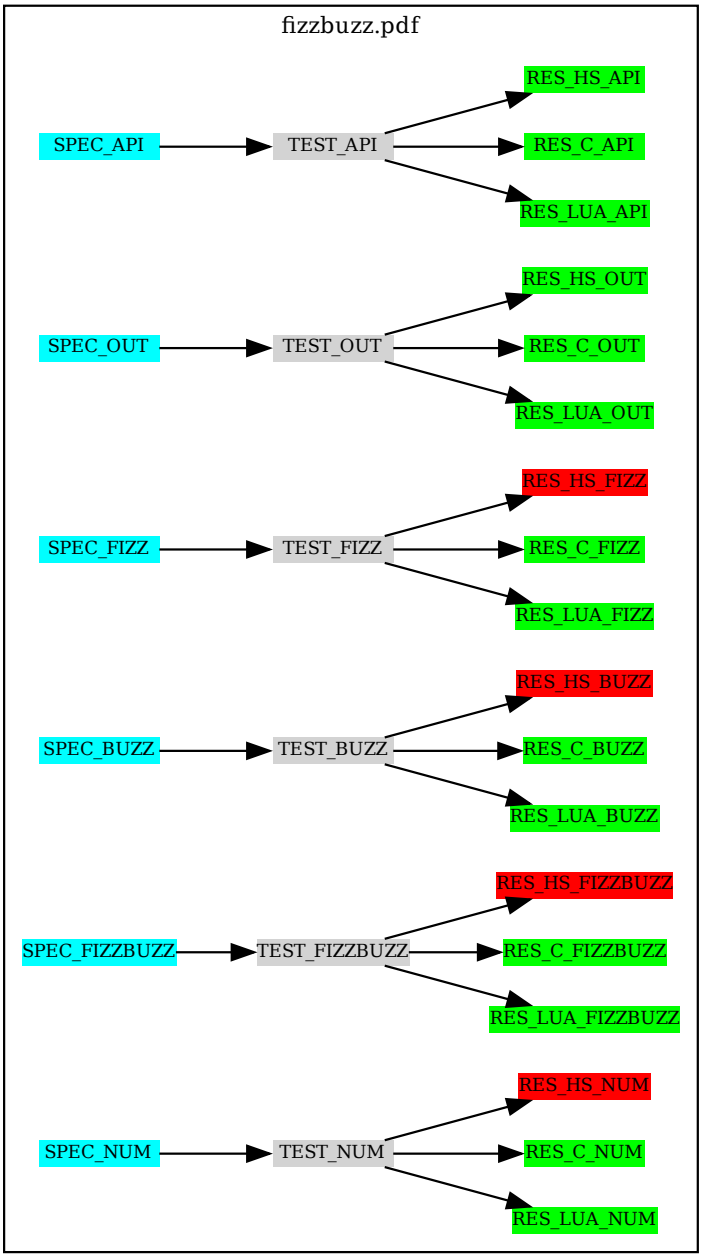
n	Lua	C	Haskell	Comparison
1	1	1	1	*OK*
2	2	2	2	*OK*
3	fizz	fizz	fizz	*OK*
4	4	4	buzz	**FAIL**
5	buzz	buzz	5	**FAIL**
6	fizz	fizz	fizz	*OK*
7	7	7	7	*OK*
8	8	8	buzz	**FAIL**
9	fizz	fizz	fizz	*OK*
10	buzz	buzz	10	**FAIL**

n	Lua	C	Haskell	Comparison
11	11	11	11	*OK*
12	fizz	fizz	fizzbuzz	**FAIL**
13	13	13	13	*OK*
14	14	14	14	*OK*
15	fizzbuzz	fizzbuzz	fizz	**FAIL**
16	16	16	buzz	**FAIL**
17	17	17	17	*OK*
18	fizz	fizz	fizz	*OK*
19	19	19	19	*OK*
20	buzz	buzz	buzz	*OK*
21	fizz	fizz	fizz	*OK*
22	22	22	22	*OK*
23	23	23	23	*OK*
24	fizz	fizz	fizzbuzz	**FAIL**
25	buzz	buzz	25	**FAIL**
26	26	26	26	*OK*
27	fizz	fizz	fizz	*OK*
28	28	28	buzz	**FAIL**
29	29	29	29	*OK*
30	fizzbuzz	fizzbuzz	fizz	**FAIL**
31	31	31	31	*OK*
32	32	32	buzz	**FAIL**
33	fizz	fizz	fizz	*OK*
34	34	34	34	*OK*
35	buzz	buzz	35	**FAIL**
36	fizz	fizz	fizzbuzz	**FAIL**
37	37	37	37	*OK*
38	38	38	38	*OK*
39	fizz	fizz	fizz	*OK*
40	buzz	buzz	buzz	*OK*
41	41	41	41	*OK*
42	fizz	fizz	fizz	*OK*
43	43	43	43	*OK*
44	44	44	buzz	**FAIL**
45	fizzbuzz	fizzbuzz	fizz	**FAIL**
46	46	46	46	*OK*
47	47	47	47	*OK*
48	fizz	fizz	fizzbuzz	**FAIL**
49	49	49	49	*OK*
50	buzz	buzz	50	**FAIL**

12.5 Coverage matrix

File	fizzbuzz.pdf
SPEC_API	fizzbuzz command line argument
SPEC_OUT	fizzbuzz output on stdout
SPEC_FIZZ	fizz when n is a multiple of 3 but not 5
SPEC_BUZZ	buzz when n is a multiple of 5 but not 3
SPEC_FIZZBUZZ	fizzbuzz n is a when multiple of 3 and 5
SPEC_NUM	n when n is a not a multiple of 3 and 5
TEST_API	number of fizzbuzz values • *SPEC_API*: fizzbuzz command line argument
TEST_OUT	output on stdout • *SPEC_OUT*: fizzbuzz output on stdout
TEST_FIZZ	"fizz" values • *SPEC_FIZZ*: fizz when n is a multiple of 3 but not 5
TEST_BUZZ	"buzz" values • *SPEC_BUZZ*: buzz when n is a multiple of 5 but not 3
TEST_FIZZBUZZ	"fizzbuzz" values • *SPEC_FIZZBUZZ*: fizzbuzz n is a when multiple of 3 and 5
TEST_NUM	integral values • *SPEC_NUM*: n when n is a not a multiple of 3 and 5
RES_LUA_API	number of fizzbuzz values [PASS] • *TEST_API*: number of fizzbuzz values
RES_LUA_OUT	output on stdout [PASS] • *TEST_OUT*: output on stdout
RES_LUA_FIZZ	"fizz" values [PASS] • *TEST_FIZZ*: "fizz" values
RES_LUA_BUZZ	"buzz" values [PASS] • *TEST_BUZZ*: "buzz" values
RES_LUA_FIZZBUZZ	"fizzbuzz" values [PASS] • *TEST_FIZZBUZZ*: "fizzbuzz" values
RES_LUA_NUM	integral values [PASS] • *TEST_NUM*: integral values
RES_C_API	number of fizzbuzz values [PASS] • *TEST_API*: number of fizzbuzz values
RES_C_OUT	output on stdout [PASS] • *TEST_OUT*: output on stdout
RES_C_FIZZ	"fizz" values [PASS] • *TEST_FIZZ*: "fizz" values
RES_C_BUZZ	"buzz" values [PASS] • *TEST_BUZZ*: "buzz" values

File	`fizzbuzz.pdf`
`RES_C_FIZZBUZZ`	"fizzbuzz" values [PASS] • *`TEST_FIZZBUZZ`*: "fizzbuzz" values
`RES_C_NUM`	integral values [PASS] • *`TEST_NUM`*: integral values
`RES_HS_API`	number of fizzbuzz values [PASS] • *`TEST_API`*: number of fizzbuzz values
`RES_HS_OUT`	output on stdout [PASS] • *`TEST_OUT`*: output on stdout
`RES_HS_FIZZ`	"fizz" values [FAIL] • *`TEST_FIZZ`*: "fizz" values
`RES_HS_BUZZ`	"buzz" values [FAIL] • *`TEST_BUZZ`*: "buzz" values
`RES_HS_FIZZBUZZ`	"fizzbuzz" values [FAIL] • *`TEST_FIZZBUZZ`*: "fizzbuzz" values
`RES_HS_NUM`	integral values [FAIL] • *`TEST_NUM`*: integral values


fizzbuzz.pdf
SPEC_API
TEST_API
RES_HS_API
RES_C_API
RES_LUA_API
SPEC_OUT
TEST_OUT
RES_HS_OUT
RES_C_OUT
RES_LUA_OUT
SPEC_FIZZ
TEST_FIZZ
RES_HS_FIZZ
RES_C_FIZZ
RES_LUA_FIZZ
SPEC_BUZZ
TEST_BUZZ
RES_HS_BUZZ
RES_C_BUZZ
RES_LUA_BUZZ
SPEC_FIZZBUZZ
TEST_FIZZBUZZ
RES_HS_FIZZBUZZ
RES_C_FIZZBUZZ
RES_LUA_FIZZBUZZ
SPEC_NUM
TEST_NUM
RES_HS_NUM
RES_C_NUM
RES_LUA_NUM

Chapter 13

References

Fizzbuzz repository: https://github.com/CDSoft/fizzbuzz

This document is not about Fizzbuzz. This document is a suggestion to simplify the build process of software projects. Fizzbuzz is just an application example.

Lua: https://www.lua.org

Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

Lua documentation: https://www.lua.org/manual/5.4/

The reference manual is the official definition of the Lua language.

LuaX: https://github.com/CDSoft/luax

LuaX is a Lua interpreter and REPL based on Lua 5.4, augmented with some useful packages. LuaX can also produce standalone executables from Lua scripts.

bang: https://github.com/CDSoft/bang

Bang is a Ninja file generator scriptable in LuaX.

ypp: https://github.com/CDSoft/ypp

Ypp is a minimalist and generic text preprocessor using Lua macros.

Pandoc: https://pandoc.org

Pandoc is a universal document converter. If you need to convert files from one markup format into another, pandoc is your swiss-army knife.

Pandoc manual: https://pandoc.org/MANUAL.html

Pandoc User's Guide

Pandoc's Markdown: https://pandoc.org/MANUAL.html#pandocs-markdown

Pandoc understands an extended and slightly revised version of John Gruber's Markdown syntax. This document explains the syntax, noting differences from original Markdown.

Pandoc Lua filters: https://pandoc.org/lua-filters.html

Pandoc has long supported filters, which allow the pandoc abstract syntax tree (AST) to be manipulated between the parsing and the writing phase. Traditional pandoc filters accept a JSON representation of the pandoc AST and produce an altered JSON representation of the AST. They may be written in any programming language, and invoked from pandoc using the `--filter` option.

Although traditional filters are very flexible, they have a couple of disadvantages. First, there is some overhead in writing JSON to stdout and reading it from stdin (twice, once on each side of the filter). Second, whether a filter will work will depend on details of the user's environment. A filter may require an interpreter for a certain programming language to be available, as well as a library for manipulating the pandoc AST in JSON form. One cannot simply provide a filter that can be used by anyone who has a certain version of the pandoc executable.

Starting with version 2.0, pandoc makes it possible to write filters in Lua without any external dependencies at all. A Lua interpreter (version 5.3) and a Lua library for creating pandoc filters is built into the pandoc executable. Pandoc data types are marshaled to Lua directly, avoiding the overhead of writing JSON to stdout and reading it from stdin.

Panda: https://github.com/CDSoft/panda

Panda is a Pandoc Lua filter that works on internal Pandoc's AST.

Chapter 14

Appendices

This chapter contains the sources of this document.

14.1 LICENSE

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                                Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
```

```
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.
```

```
                    TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
```

```
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
```

```
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
```

```
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:
```

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

```
  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.
```

```
  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.
```

```
  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
```

```
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
```

```
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
```

```
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
```

```
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.
```

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

```
The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<https://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<https://www.gnu.org/licenses/why-not-lgpl.html>.
```

14.2 fizzbuzz.md

```
---
title: Fizz buzz - LuaX demo
date: @DATE
author: @AUTHOR
keywords:
    - Lua
    - Script
    - Documentation
    - Tests
    - Build system

titlepage: true

caption-justification: raggedright

toc-own-page: true

listings-disable-line-numbers: false
listings-no-page-break: true
disable-header-and-footer: false

footnotes-pretty: true
footnotes-disable-backlinks: true

book: true
```

```
classoption: oneside

titlepage-logo: "{{logo}}"
logo-width: 60mm

table-use-row-colors: true

code-block-font-size: "\\small"
---

```meta
logo = os.getenv "LOGO"
```

Disclaimer

This document is not about [Fizzbuzz](https://en.wikipedia.org/wiki/Fizz_buzz).
This document is a suggestion to simplify the build process of software
projects, a demo of an **homogeneous and consistent** development and
documentation environment. Fizzbuzz is just an application example.

Links

- [fizzbuzz_slideshow.pdf](http://cdelord.fr/fizzbuzz/fizzbuzz_slideshow.pdf): PDF slidesho
- [fizzbuzz.pdf](http://cdelord.fr/fizzbuzz/fizzbuzz.pdf): PDF demonstration (specification
- github.com/CDSoft/fizzbuzz: Sources

{width=50%}

Introduction

Lots of software projects involve various tools, free as well as commercial, to
build the software, run the tests, produce the documentation, ... These tools
use different data formats and scripting languages, which makes the projects
less scalable and harder to maintain.

Sharing data between configuration files, documentations, tests results can
then be painful and counter productive (the necessary glue is often more
complex than the tools themselves).

Usually people script their build systems and processes with languages like
Bash, Python, Javascript and make them communicate with plain text, YAML, JSON,
XML, CSV, INI, TOML. Every script shall rely on specific (existing or not)
libraries to read and write these data formats.
```

```
This document presents a commun and powerful data format and some tools to
script the build process of a project and generate documentation.

To sum up the suggested solution is:

- a **single data format**
- and a **reduced set of highly configurable tools**.

# Lua[^lua]

[Lua](https://www.lua.org) is the perfect candidate for both a
common data format and a script language.

## What is Lua?

Lua is a powerful, efficient, lightweight, embeddable scripting language. It
supports procedural programming, object-oriented programming, functional
programming, data-driven programming, and data description.

Lua combines simple procedural syntax with powerful data description constructs
based on associative arrays and extensible semantics. Lua is dynamically typed,
runs by interpreting bytecode with a register-based virtual machine, and has
automatic memory management with incremental garbage collection, making it
ideal for configuration, scripting, and rapid prototyping.

[^lua]: from <https://www.lua.org/about.html>

## Why choose Lua?

**Lua is a proven, robust language**

Lua has been used in many industrial applications (e.g., Adobe's Photoshop
Lightroom), with an emphasis on embedded systems (e.g., the Ginga middleware
for digital TV in Brazil) and games (e.g., World of Warcraft and Angry Birds).
Lua is currently the leading scripting language in games. Lua has a solid
reference manual and there are several books about it. Several versions of Lua
have been released and used in real applications since its creation in 1993.
Lua featured in HOPL III, the Third ACM SIGPLAN History of Programming
Languages Conference, in 2007. Lua won the Front Line Award 2011 from the Game
Developers Magazine.

**Lua is fast**

Lua has a deserved reputation for performance. To claim to be "as fast as Lua"
is an aspiration of other scripting languages. Several benchmarks show Lua as
```

```
the fastest language in the realm of interpreted scripting languages. Lua is
fast not only in fine-tuned benchmark programs, but in real life too.
Substantial fractions of large applications have been written in Lua.

**Lua is portable**

Lua is distributed in a small package and builds out-of-the-box in all
platforms that have a standard C compiler. Lua runs on all flavors of Unix and
Windows, on mobile devices (running Android, iOS, BREW, Symbian, Windows
Phone), on embedded microprocessors (such as ARM and Rabbit, for applications
like Lego MindStorms), on IBM mainframes, etc.

**Lua is powerful (but simple)**

A fundamental concept in the design of Lua is to provide meta-mechanisms for
implementing features, instead of providing a host of features directly in the
language. For example, although Lua is not a pure object-oriented language, it
does provide meta-mechanisms for implementing classes and inheritance. Lua's
meta-mechanisms bring an economy of concepts and keep the language small, while
allowing the semantics to be extended in unconventional ways.

**Lua is small**

Adding Lua to an application does not bloat it. The tarball for Lua 5.4,
which contains source code and documentation, takes 353K compressed and 1.3M
uncompressed. The source contains around 30000 lines of C. Under 64-bit Linux,
the Lua interpreter built with all standard Lua libraries takes 281K and the
Lua library takes 468K.

**Lua is free**

Lua is free open-source software, distributed under a very liberal license (the
well-known MIT license). It may be used for any purpose, including commercial
purposes, at absolutely no cost. Just download it and use it.

# LuaX

[LuaX](https://github.com/CDSoft/luax) is a Lua interpreter and REPL based on
Lua 5.4, augmented with some useful packages. LuaX can also produce
standalone executables from Lua scripts.

LuaX runs on several platforms with no dependency:

- Linux (x86_64, i386, aarch64)
- MacOS (x86_64, aarch64)
```

```
- Windows (x86_64, i386)

LuaX can cross-compile scripts from and to any of these platforms.

LuaX comes with a standard Lua interpreter and provides some libraries (embedded
in a single executable, no external dependency required):

- [LuaX interactive usage](https://github.com/CDSoft/luax/blob/master/doc/repl.md): improve
- [F](https://github.com/CDSoft/luax/blob/master/doc/F.md): functional programming inspired
- [fs](https://github.com/CDSoft/luax/blob/master/doc/fs.md): file system management
- [sh](https://github.com/CDSoft/luax/blob/master/doc/sh.md): shell command execution
- [mathx](https://github.com/CDSoft/luax/blob/master/doc/mathx.md): complete math library f
- [imath](https://github.com/CDSoft/luax/blob/master/doc/imath.md): arbitrary precision int
- [qmath](https://github.com/CDSoft/luax/blob/master/doc/qmath.md): rational number library
- [complex](https://github.com/CDSoft/luax/blob/master/doc/complex.md): math library for co
- [ps](https://github.com/CDSoft/luax/blob/master/doc/ps.md): Process management module
- [sys](https://github.com/CDSoft/luax/blob/master/doc/sys.md): System module
- [crypt](https://github.com/CDSoft/luax/blob/master/doc/crypt.md): cryptography module
- [lz4](https://github.com/CDSoft/luax/blob/master/doc/lz4.md): Extremely Fast Compression
- [lpeg](https://github.com/CDSoft/luax/blob/master/doc/lpeg.md): Parsing Expression Gramma
- [linenoise](https://github.com/CDSoft/luax/blob/master/doc/linenoise.md): light readline
- [luasocket](https://github.com/CDSoft/luax/blob/master/doc/luasocket.md): Network support
- [inspect](https://github.com/CDSoft/luax/blob/master/doc/inspect.md): Human-readable repr

More information here: <http://cdelord.fr/luax>

# Scripting with LuaX

LuaX can be used as a general programming language.
There are plenty of [good documentations for Lua](https://www.lua.org/docs.html)
and [LuaX](http://cdelord.fr/luax).

A big advantage of Lua is the usage of Lua tables as a common data format usable by various
It is Human-readable and structured. It can be generated by Lua scripts but also by any soft

Typical usages are:

- project/software configuration
    - a Lua table can be used to describe a project or a software configuration
        - read by an embedded Lua interpreter
        - used to generate documentation or source code
- tests results
    - a test suite can generate test results as a Lua table
    - tests results can be used to render documentation (tests reports) and compute a test c
```

```
The next chapters present some tools written in Lua/LuaX or using Lua as a scripting engine.

# Bang

[Bang](https://github.com/CDSoft/bang) is a ninja file generator scriptable in
LuaX, a Lua interpreter with a bunch of useful modules (file management,
functional programming module, basic cryptography, ...). It takes a build
description (a LuaX script) and generates a Ninja file.

Bang provides functions to generate ninja primitives (variables, rules, build
statements, ...) and some extra features:

- rule/build statement pairs described in a single function call
- file listing and filenames list management using LuaX modules (e.g. F and fs)
- pipe simulation using rule composition
- "clean", "install" and "help" targets

Bang comes with an example that shows how to use bang and LuaX functions to:

- discover source files actually present in the repository: no redundant hard
  coded file lists (redundancy means painful maintenance)
- cross-compile the same sources for multiple platforms: compilation for several
  platforms without any dirty copy/paste
- describe static libraries: in the `lib` directory, each sub-directory is
  a library compiled and archived in its own `.a` file
- describe executables: in the `bin` directory, each C source file is the main
  file of a binary containing this C file as well as libraries from the `lib`
  directory.

Bang is currently used to build bang itself but also LuaX and some projects
available on my [GitHub](https://github.com/CDSoft).

# Ypp

Ypp is a minimalist and generic text preprocessor using Lua macros.

Ypp is compiled by LuaX, i.e. Lua and LuaX functions and modules are available
in macros.

More information here: <http://cdelord.fr/ypp>

Ypp is pretty simple. It searches for Lua expressions and replaces macros with their results

?(false)
```

````
Macro       Result
----------- ------------------------------------------------------------------------
`@(...)`    Evaluates the Lua expression `...` and replaces the macro by its result
`@@(...)`   Executes the Lua chunk `...` and replaces the macro by its result (if not `nil`

Some expression do not require parentheses (function calls).

?(true)

## Example

?(false)

``````{.markdown}
$$
\sum_{i=1}^{100} i^2 = @F.range(100):map(function(x) return x*x end):sum()
$$
``````

?(true)

is rendered as

> $$
> \sum_{i=1}^{100} i^2 = @F.range(100):map(function(x) return x*x end):sum()
> $$

Macros can also define variables reusable later by other macros.

?(false)
``````{.markdown}
@@[[
    local foo = 42
    N = foo * 23 + 34
    local function sq(x) return x*x end
    function sumsq(n) return F.range(N):map(sq):sum() end
]]
``````
?(true)

@@[[
    local foo = 42
    N = foo * 23 + 34
    local function sq(x) return x*x end
    function sumsq(n) return F.range(N):map(sq):sum() end
````

```
]]

defines `N` ($N = @N$) which can be read in a Lua expression or with ?(false)`@N`?(true)
and `sumsq` which computes the sum of squares.

Then

?(false)

``````{.markdown}
$$
\sum_{i=1}^{@N} i^2 = @sumsq(N)
$$
``````

?(true)

becomes

> $$
> \sum_{i=1}^{@N} i^2 = @sumsq(N)
> $$

Pandoc

[Pandoc](https://pandoc.org/) is a swiss-army knife to convert from and to a
bunch of document formats.

A big advantage of Pandoc is the ability to use Lua scripts to define custom
readers and writers for unsupported formats and also Lua filters to manipulate
the pandoc abstract syntax tree (AST). This is the main pandoc feature
exercised in this document.

Pandoc has an excellent documentation:

- main pandoc documentation: <https://pandoc.org/MANUAL.html>
- Lua filter documentation: <https://pandoc.org/lua-filters.html>

Fizzbuzz uses pandoc Lua filters with Panda (see next chapter) which bundles
some useful filters in a single script.

Panda

Panda is a [Pandoc Lua filter](https://pandoc.org/lua-filters.html) that works
on internal Pandoc's AST.
```

````
It provides several interesting features:

- variable expansion (minimalistic templating)
- conditional blocks
- file inclusion (e.g. for source code examples)
- script execution (e.g. to include the result of a command)
- diagrams (Graphviz, PlantUML, ditaa, Asymptote, blockdiag, mermaid...)

The documentation of Panda is here: <http://cdelord.fr/panda>

## Examples

There are lots of examples in the documentation of panda. We will see here two of them.

**Documentation extraction from source code**

The source code can be documented by adding special marks in comments. The documentation sha

For instance, the following C source contains documentation that can be extracted and includ

``````{.c include=deep_thought.c}

``````

To extract the documentation, panda provides a macro to replace a `div` element by the docu

`````` markdown
:::{doc=deep_thought.c}
:::
``````

will be replaced by:

> :::{doc=deep_thought.c}
> :::

Diagrams

Diagrams can be embedded in Pandoc documents. Diagrams are specified as code
blocks and are replaced by an image by panda.

``` meta
_dot = "{{dot}}"
_gnuplot = "{{gnuplot}}"
```
````

````
    ```{.dot render="{{_dot}}" width=67%}
    digraph {
        rankdir=LR;
        input -> pandoc -> output
        pandoc -> panda -> {pandoc, diagrams}
        { rank=same; pandoc, panda }
        { rank=same; diagrams, output }
    }
    ```

```{.dot render="{{dot}}" name=example-graphviz width=67%}
digraph {
    rankdir=LR;
    input -> pandoc -> output
    pandoc -> panda -> {pandoc, diagrams}
    { rank=same; pandoc, panda }
    { rank=same; diagrams, output }
}
```

    ```{render="{{_gnuplot}}" width=67%}
    set xrange [-pi:pi]
    set yrange [-1.5:1.5]
    plot sin(x) lw 4, cos(x) lw 4
    ```

```{render="{{gnuplot}}" name=example-gnuplot width=67%}
set xrange [-pi:pi]
set yrange [-1.5:1.5]
plot sin(x) lw 4, cos(x) lw 4
```

# hey

`hey` is a shell script. It is intended to easily install some tools based on
LuaX and Pandoc to pre-process files and generate documents, using Lua as a
common, simple and powerful scripting language.

## Example

Easy installation, only `hey` is needed:

``` sh
wget https://raw.githubusercontent.com/CDSoft/hey/master/hey
```
````

````
Its usage is very similar to `apt` or `dnf`:

``` sh
$ hey list
@sh "hey list"
```

``` sh
$ hey install all
...
```

# Fizzbuzz

Fizzbuzz is a concrete example of the usage of LuaX/ypp/pandoc/panda to specify
and test a software.

## Specification

From [Wikipedia](https://en.wikipedia.org/wiki/Fizz_buzz):

> Fizz buzz is a group word game for children to teach them about division.
> Players take turns to count incrementally, replacing any number divisible by
> three with the word "fizz", and any number divisible by five with the word
> "buzz".

`fizzbuzz` is a function that returns `"fizz"`, `"buzz"`, `"fizzbuzz"` or `n` for any posit

$$
    fizzbuzz : \mathbb{N}^+ \to \{fizz, buzz, fizzbuzz\} \cup \mathbb{N}^+
$$
$$
    fizzbuzz(n) =
        \begin{cases}
            \text{"fizzbuzz" }  & \text{if } (3|n) \land (5|n) \\
            \text{"fizz" }      & \text{if } (3|n) \land \lnot (5|n) \\
            \text{"buzz" }      & \text{if } (5|n) \land \lnot (3|n) \\
            n                   & \text{if } \lnot (3|n) \land \lnot (5|n) \\
        \end{cases}
$$

@@[[
    function fizzbuzz(n)
        if n % 15 == 0 then return "fizzbuzz" end
        if n % 3 == 0 then return "fizz" end
````

```
        if n % 5 == 0 then return "buzz" end
        return n
    end
]]

### Requirements

@req "SPEC_API: fizzbuzz command line argument"

The fizzbuzz program takes one argument that specify the number for fizzbuzz
values to generate.

@req "SPEC_OUT: fizzbuzz output on stdout"

The fizzbuzz program emits fizzbuzz values on the standard output.
Each line contains `n` and `fizzbuzz(n)`.

e.g.:

```
$ fizzbuzz 6
@F.range(6):map(function(n) return F{n, fizzbuzz(n)}:str "\t" end)
```

@req "SPEC_FIZZ: fizz when n is a multiple of 3 but not 5"

If `n` is a multiple of 3 but not 5, then `fizzbuzz(n)` is `"fizz"`.

@req "SPEC_BUZZ: buzz when n is a multiple of 5 but not 3"

If `n` is a multiple of 5 but not 3, then `fizzbuzz(n)` is `"buzz"`.

@req "SPEC_FIZZBUZZ: fizzbuzz n is a when multiple of 3 and 5"

If `n` is a multiple of 3 and 5, then `fizzbuzz(n)` is `"fizzbuzz"`.

@req "SPEC_NUM: n when n is a not a multiple of 3 and 5"

If `n` is a multiple of 3 and 5, then `fizzbuzz(n)` is `"fizzbuzz"`.

Examples

@[[
    {
        "n | fizzbuzz(n) | n | fizzbuzz(n) | n | fizzbuzz(n) | n | fizzbuzz(n) ",
```

```
        "---|-------------|---|-------------|---|-------------|---|-------------",
    }
    ..
    F.range(5):map(function(n)
        return F{
            n,      fizzbuzz(n),
            n+5,    fizzbuzz(n+5),
            n+10,   fizzbuzz(n+10),
            n+15,   fizzbuzz(n+15),
        }:str "|"
    end)
]]

## Implementation

### Lua implementation

:::{doc=fizzbuzz.lua shift=3}
:::

### C implementation

:::{doc=fizzbuzz.c shift=3}
:::

### Haskell implementation

:::{doc=fizzbuzz.hs shift=3}
:::

## Tests

The results of the Fizzbuzz executables are checked by the test script `fizzbuzz_test.lua`.
This script check the fizzbuzz results and produces a Lua table with the test results.
This script will later be used to build the test reports.

### Test plan

@@( test_cfg = require "test_config" )

Each fizzbuzz implementation is executed (with @test_cfg.N values). The results are
checked by `fizzbuzz_test.lua` and stored in a Lua table.

The fizzbuzz values are recorded in the `fizzbuzz` field of the test result table.
```

```
@req "TEST_API: number of fizzbuzz values" {
    refs = "SPEC_API",
}

The fizzbuzz list contains @test_cfg.N values.

The result of this test is recorded in the `valid_number_of_lines` field of the test result

@req "TEST_OUT: output on stdout" {
    refs = "SPEC_OUT",
}

The fizzbuzz list is emitted on stdout.

@req "TEST_FIZZ: \"fizz\" values" {
    refs = "SPEC_FIZZ",
}

All multiples of 3 but not 5 are `"fizz"`.

The result of this test is recorded in the `valid_fizz` field of the test result table.

@req "TEST_BUZZ: \"buzz\" values" {
    refs = "SPEC_BUZZ",
}

All multiples of 5 but not 3 are `"buzz"`.

The result of this test is recorded in the `valid_buzz` field of the test result table.

@req "TEST_FIZZBUZZ: \"fizzbuzz\" values" {
    refs = "SPEC_FIZZBUZZ",
}

All multiples of 3 and 5 are `"fizzbuzz"`.

The result of this test is recorded in the `valid_fizzbuzz` field of the test result table.

@req "TEST_NUM: integral values" {
    refs = "SPEC_NUM",
}

All non multiples of 3 and 5 are themselves.

The result of this test is recorded in the `valid_numbers` field of the test result table.
```

```
## Test reports

### Lua implementation

@@(lua_tests = require "result_lua")

The Lua fizzbuzz function returns:

@F.str(lua_tests.fizzbuzz, ", ")

@req.test "RES_LUA_API: number of fizzbuzz values" {
    refs = "TEST_API",
    status = lua_tests.valid_number_of_lines,
}

@req.test "RES_LUA_OUT: output on stdout" {
    refs = "TEST_OUT",
    status = lua_tests.valid_number_of_lines,
}

@req.test "RES_LUA_FIZZ: \"fizz\" values" {
    refs = "TEST_FIZZ",
    status = lua_tests.valid_fizz,
}

@req.test "RES_LUA_BUZZ: \"buzz\" values" {
    refs = "TEST_BUZZ",
    status = lua_tests.valid_buzz,
}

@req.test "RES_LUA_FIZZBUZZ: \"fizzbuzz\" values" {
    refs = "TEST_FIZZBUZZ",
    status = lua_tests.valid_fizzbuzz,
}

@req.test "RES_LUA_NUM: integral values" {
    refs = "TEST_NUM",
    status = lua_tests.valid_numbers,
}

**Summary**: @lua_tests.nb_pass / @lua_tests.nb tests passed

### C implementation

@@(c_tests = require "result_c")
```

```
The C fizzbuzz function returns:

@F.str(c_tests.fizzbuzz, ", ")

@req.test "RES_C_API: number of fizzbuzz values" {
    refs = "TEST_API",
    status = c_tests.valid_number_of_lines,
}

@req.test "RES_C_OUT: output on stdout" {
    refs = "TEST_OUT",
    status = c_tests.valid_number_of_lines,
}

@req.test "RES_C_FIZZ: \"fizz\" values" {
    refs = "TEST_FIZZ",
    status = c_tests.valid_fizz,
}

@req.test "RES_C_BUZZ: \"buzz\" values" {
    refs = "TEST_BUZZ",
    status = c_tests.valid_buzz,
}

@req.test "RES_C_FIZZBUZZ: \"fizzbuzz\" values" {
    refs = "TEST_FIZZBUZZ",
    status = c_tests.valid_fizzbuzz,
}

@req.test "RES_C_NUM: integral values" {
    refs = "TEST_NUM",
    status = c_tests.valid_numbers,
}

**Summary**: @c_tests.nb_pass / @c_tests.nb tests passed

### Haskell implementation

@@(hs_tests = require "result_hs")

The Haskell fizzbuzz function returns:

@F.str(hs_tests.fizzbuzz, ", ")

@req.test "RES_HS_API: number of fizzbuzz values" {
```

```
    refs = "TEST_API",
    status = hs_tests.valid_number_of_lines,
}

@req.test "RES_HS_OUT: output on stdout" {
    refs = "TEST_OUT",
    status = hs_tests.valid_number_of_lines,
}

@req.test "RES_HS_FIZZ: \"fizz\" values" {
    refs = "TEST_FIZZ",
    status = hs_tests.valid_fizz,
}

@req.test "RES_HS_BUZZ: \"buzz\" values" {
    refs = "TEST_BUZZ",
    status = hs_tests.valid_buzz,
}

@req.test "RES_HS_FIZZBUZZ: \"fizzbuzz\" values" {
    refs = "TEST_FIZZBUZZ",
    status = hs_tests.valid_fizzbuzz,
}

@req.test "RES_HS_NUM: integral values" {
    refs = "TEST_NUM",
    status = hs_tests.valid_numbers,
}

**Summary**: @hs_tests.nb_pass / @hs_tests.nb tests passed

### Lua / C / Haskell comparison

@[[
    {
        "n  | Lua | C | Haskell | Comparison",
        "---|-----|---|---------|-----------",
    } .. F.zip {
        lua_tests.fizzbuzz,
        c_tests.fizzbuzz,
        hs_tests.fizzbuzz,
    }:mapi(function (i, res)
        local expected = tostring(fizzbuzz(i))
        local ok = res:all(F.partial(F.op.eq, expected))
        return ({i}..res..{ok and "*OK*" or "**FAIL**"}):str "|"
```

```
    end)
]]

## Coverage matrix

@req.matrix "g"

```{.dot render="{{dot}}" name=coverage-matrix}
@req.dot()
```

References

@@[[
    link = F.curry(function(name, url)
        return F.I{name=name, url=url}"[**$(name)**]($(url)): <$(url)>\n"
    end)
]]

@link "Fizzbuzz repository" "https://github.com/CDSoft/fizzbuzz"
> This document is not about Fizzbuzz. This document is a suggestion to
> simplify the build process of software projects. Fizzbuzz is just an
> application example.

@link "Lua" "https://www.lua.org"
> Lua is a powerful, efficient, lightweight, embeddable scripting language. It
> supports procedural programming, object-oriented programming, functional
> programming, data-driven programming, and data description.

@link "Lua documentation" "https://www.lua.org/manual/5.4/"
> The reference manual is the official definition of the Lua language.

@link "LuaX" "https://github.com/CDSoft/luax"
> LuaX is a Lua interpreter and REPL based on Lua 5.4, augmented with some
> useful packages. LuaX can also produce standalone executables from Lua
> scripts.

@link "bang" "https://github.com/CDSoft/bang"
> Bang is a Ninja file generator scriptable in LuaX.

@link "ypp" "https://github.com/CDSoft/ypp"
> Ypp is a minimalist and generic text preprocessor using Lua macros.

@link "Pandoc" "https://pandoc.org"
> Pandoc is a universal document converter. If you need to convert files from
```

```
> one markup format into another, pandoc is your swiss-army knife.

@link "Pandoc manual" "https://pandoc.org/MANUAL.html"
> Pandoc User's Guide

@link "Pandoc's Markdown" "https://pandoc.org/MANUAL.html#pandocs-markdown"
> Pandoc understands an extended and slightly revised version of John Gruber's
> Markdown syntax. This document explains the syntax, noting differences from
> original Markdown.

@link "Pandoc Lua filters" "https://pandoc.org/lua-filters.html"
> Pandoc has long supported filters, which allow the pandoc abstract syntax
> tree (AST) to be manipulated between the parsing and the writing phase.
> Traditional pandoc filters accept a JSON representation of the pandoc AST and
> produce an altered JSON representation of the AST. They may be written in any
> programming language, and invoked from pandoc using the `--filter` option.
>
> Although traditional filters are very flexible, they have a couple of
> disadvantages. First, there is some overhead in writing JSON to stdout and
> reading it from stdin (twice, once on each side of the filter). Second,
> whether a filter will work will depend on details of the user's environment.
> A filter may require an interpreter for a certain programming language to be
> available, as well as a library for manipulating the pandoc AST in JSON form.
> One cannot simply provide a filter that can be used by anyone who has a
> certain version of the pandoc executable.
>
> Starting with version 2.0, pandoc makes it possible to write filters in Lua
> without any external dependencies at all. A Lua interpreter (version 5.3) and
> a Lua library for creating pandoc filters is built into the pandoc
> executable. Pandoc data types are marshaled to Lua directly, avoiding the
> overhead of writing JSON to stdout and reading it from stdin.

@link "Panda" "https://github.com/CDSoft/panda"
> Panda is a Pandoc Lua filter that works on internal Pandoc's AST.

::::::{.if output_file=".build/fizzbuzz.pdf"}

# Appendices

This chapter contains the sources of this document.

## LICENSE

```{.markdown include=LICENSE}
```
```

````
## fizzbuzz.md

```{.markdown include=fizzbuzz.md}
```

## project_data.lua

```{.lua include=project_data.lua}
```

## fizzbuzz.lua

```{.lua include=fizzbuzz.lua}
```

## fizzbuzz.c

```{.c include=fizzbuzz.c}
```

## fizzbuzz.hs

```{.hs include=fizzbuzz.hs}
```

## test_config.lua

```{.lua include=test_config.lua}
```

## fizzbuzz_test.lua

```{.lua include=fizzbuzz_test.lua}
```

## build.lua

```{.lua include=build.lua}
```

::::::
````

14.3 project_data.lua

```
AUTHOR = "Christophe Delord - <http://cdelord.fr/fizzbuzz>"
DATE = os.date("%a %b %e, %Y", sh "git log -1 --format=%ct")
```

14.4 fizzbuzz.lua

```
#!/usr/bin/env luax

--[[@@@

The Lua implementation of Fizzbuzz is based on a functional style,
using function compositions.

It computes the `"fizz"` and `"buzz"` parts and return them
if at least one of them is not `nil`{.lua}.
Otherwise it returns its argument unchanged.

```{ .dot render="{{dot}}" name=fizzbuzz-lua width=100% }
digraph {

n [label="n" shape=oval]

compute_fizz [label="\"fizz\"\nif n = 0 (mod 3)" shape=box]
compute_buzz [label="\"buzz\"\nif n = 0 (mod 5)" shape=box]
combine [label="combine" shape=box]
select [label="Choose\n\"fizz\", \"buzz\", \"fizzbuzz\"\nor n" shape=box]

fizz [label="\"fizz\" or nil" shape=oval]
buzz [label="\"buzz\" or nil" shape=oval]
fizzbuzz [label="\"fizz\", \"buzz\", \"fizzbuzz\"\nor nil" shape=oval]
fizzbuzz_n [label="\"fizz\", \"buzz\", \"fizzbuzz\"\nor n" shape=oval]

n -> compute_fizz -> fizz -> combine
n -> compute_buzz -> buzz -> combine
n -> select
combine -> fizzbuzz -> select
select -> fizzbuzz_n

{ rank=same; combine, fizzbuzz, select }

}
```
```

```
```{.lua include="fizzbuzz.lua" pattern="%-%-%s*fizzbuzz%s*{%s*(.-)%s*%-%-%s*}" format="%1"}
```
@@@]]

local F = require "F"

-- fizzbuzz {

local function div(d, s, n)
    return n % d == 0 and s or nil
end

local fizz = F.partial(div, 3, "fizz")
local buzz = F.partial(div, 5, "buzz")

local function combine(a, b)
    return a and (a..(b or "")) or b
end

local function fizzbuzz(n)
    return combine(fizz(n), buzz(n)) or n
end

-- }

local n = tonumber(arg[1])
assert(n, tostring(arg[1])..": not a number")

F.range(n)
    : map(fizzbuzz)
    : foreachi(print)
```

14.5 fizzbuzz.c

```
/*@@@

The C implementation of Fizzbuzz uses an array of string formats
used by `sprintf`{.c} to produce `"fizz"`, `"buzz"`, `"fizzbuzz"`
or the function argument.

The array index is a 2-bit integer, each bit being the divisilibity
of the argument by 3 or 5.

```{.c include="fizzbuzz.c" pattern="[c]onst.-%b{}"}
```
```

```
```
@@@*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static const char *fizzbuzz(int i, char *s)
{
    static const char *fmt[] = {
        [0|(0<<1)] = "%d",
        [1|(0<<1)] = "fizz",
        [0|(1<<1)] = "buzz",
        [1|(1<<1)] = "fizzbuzz",
    };
    const int fizz = (i%3 == 0) << 0;
    const int buzz = (i%5 == 0) << 1;
    sprintf(s, fmt[fizz|buzz], i);
    return s;
}

int main(int argc, const char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "argument expected\n");
        exit(1);
    }
    const int n = atoi(argv[1]);
    char s[64];
    for (int i = 1; i <= n; i++)
    {
        printf("%d\t%s\n", i, fizzbuzz(i, s));
    }
    return EXIT_SUCCESS;
}
```

## 14.6 fizzbuzz.hs

```
{-@@@

The Haskell implementation of Fizzbuzz builds infinite lists
of fizzes, buzzes and integers.
```
```

```
The functions `fizzbuzz` builds three infinite lists and combine them.

ns      1     2     3     4     5     6     7     8     9     10    11    12    13    14
------- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
fizzes  .     .     fizz  .     .     fizz  .     .     fizz  .     .     fizz  .     .
buzzes  .     .     .     .     buzz  .     .     .     .     buzz  .     .     .     .

$$
fizzbuzz(n) =
    \begin{cases}
        fizz + buzz     & \text{if } fizz \ne Nothing \lor buzz \ne Nothing \\
        n               & \text{if } fizz = buzz = Nothing \\
    \end{cases}
$$

```{.hs include="fizzbuzz.hs" pattern="%-%-%s*fizzbuzz%s*{%s*(.-)%s*%-%-%s*}" format="%1"}
```
@@@-}

import Control.Monad
import Data.Maybe
import System.Environment

-- fizzbuzz {

fizzbuzz :: [String]
fizzbuzz = zipWith3 combine fizzes buzzes ns
    where
        ws d w = cycle $ replicate (d-1) Nothing ++ [Just w]
        fizzes = ws 3 "fizz"
        buzzes = ws 4 "buzz" -- bug that shall be detected by the tests
        ns = show <$> [1..]
        combine f b n = fromMaybe n (f<>b)

-- }

main :: IO ()
main = do
    n <- read . head <$> getArgs
    forM_ (zip [1..n] fizzbuzz) $ \(i, s) ->
        putStrLn $ show i ++ "\t" ++ s
```

14.7 test_config.lua

```
return {
    N = 50, -- number of fizzbuzz values to test
}
```

14.8 fizzbuzz_test.lua

```
#!/usr/bin/env luax

local F = require "F"
local fs = require "fs"

local result_file = arg[1]
local N = tonumber(arg[2])

assert(result_file and N, "Wrong arguments")

local indices, fizzbuzzes = fs.read(result_file)
    : lines()
    : map(string.words)
    : unzip()
indices = indices:map(tonumber)

local tests = F{}

-- fizzbuzz list used to render the test results
tests.fizzbuzz = fizzbuzzes

-------------------------------------------------------------------------------
-- The number of line shall be N
-------------------------------------------------------------------------------

tests.valid_number_of_lines =
    #indices == N and #fizzbuzzes == N
    and F.op.ueq(indices, F.range(N))

-------------------------------------------------------------------------------
-- Multiples of 3 but not 5 are "fizz"
-------------------------------------------------------------------------------

tests.valid_fizz =
    fizzbuzzes
        : filteri(function(i, _) return i%3 == 0 and i%5 ~= 0 end)
```

```
        : all(F.partial(F.op.eq, "fizz"))

-------------------------------------------------------------------------------
-- Multiples of 5 but not 3 are "buzz"
-------------------------------------------------------------------------------

tests.valid_buzz =
    fizzbuzzes
        : filteri(function(i, _) return i%3 ~= 0 and i%5 == 0 end)
        : all(F.partial(F.op.eq, "buzz"))

-------------------------------------------------------------------------------
-- Multiples of 3 and 5 are "fizzbuzz"
-------------------------------------------------------------------------------

tests.valid_fizzbuzz =
    fizzbuzzes
        : filteri(function(i, _) return i%3 == 0 and i%5 == 0 end)
        : all(F.partial(F.op.eq, "fizzbuzz"))

-------------------------------------------------------------------------------
-- Non multiples of 3 and 5 are themselves
-------------------------------------------------------------------------------

tests.valid_numbers =
    fizzbuzzes
        : mapi(function(i, s)
            return i%3 == 0 or i%5 == 0 or F.read(s) == i
        end)
        : land()

-------------------------------------------------------------------------------
-- Statistics
-------------------------------------------------------------------------------

local results = tests
    : filtert(function(res) return type(res) == "boolean" end)
    : values()

tests.nb = #results
tests.nb_pass = #results:filter(F.partial(F.op.eq, true))
tests.nb_fail = #results:filter(F.partial(F.op.eq, false))

-------------------------------------------------------------------------------
-- Format test results
```

```
-------------------------------------------------------------------------------

print("--[[ Fizzbuzz output")
print("indices", F.show(indices))
print("fizzbuzzes", F.show(fizzbuzzes))
print("]]")

print("return", F.show(tests, {indent=4}))
```

14.9 build.lua

```
local F = require "F"

-------------------------------------------------------------------------------
section "Project directories"
-------------------------------------------------------------------------------

var "builddir" ".build"
var "img" "img"

clean "$builddir"

local all = {}
require "atexit"(function() default(all) end)

-------------------------------------------------------------------------------
section "Help"
-------------------------------------------------------------------------------

help.description "Fizzbuzz build system"
help "all" "compile, test and document FizzBuzz"

-------------------------------------------------------------------------------
section "Tests"
-------------------------------------------------------------------------------

local test_config = require "test_config"

rule "run_test" {
    command = { "$in", test_config.N, "> $out" }
}

section "Lua test"
```

```
acc(all) {
    build "$builddir/tests/fizzbuzz_lua" { "fizzbuzz.lua",
        command = "luax -q -o $out $in",
    },
    build "$builddir/tests/fizzbuzz_lua.txt" {
        "run_test", "$builddir/tests/fizzbuzz_lua",
    },
}

section "C test"

acc(all) {
    build "$builddir/tests/fizzbuzz_c" { "fizzbuzz.c",
        command = "gcc $in -o $out",
    },
    build "$builddir/tests/fizzbuzz_c.txt" {
        "run_test", "$builddir/tests/fizzbuzz_c",
    },
}

section "Haskell test"

acc(all) {
    build "$builddir/tests/fizzbuzz_hs" { "fizzbuzz.hs",
        command = "ghc -outputdir ${out}_tmp $in -o $out",
    },
    build "$builddir/tests/fizzbuzz_hs.txt" {
        "run_test", "$builddir/tests/fizzbuzz_hs",
    },
}

section "Test results"

rule "check" {
    command = { "luax", "$in", test_config.N, "> $out" }
}

acc(all) {
    F"lua c hs":words():map(function(lang)
        return build("$builddir/tests/result_"..lang..".lua") { "check",
            "fizzbuzz_test.lua",
            "$builddir/tests/fizzbuzz_"..lang..".txt",
        }
    end)
}
```

```
-------------------------------------------------------------------------------
section "Documentation"
-------------------------------------------------------------------------------

local env = {
    'export LUA_PATH="$builddir/tests/?.lua;./?.lua";',
    'export REQDB="$builddir/reqdb.lua";',
    'export REQTARGET="fizzbuzz.pdf";',
}

local ypp_flags = {
    "-p .",
    "-l project_data",
    "-l req",
}

rule "ypp" {
    command = { env, "ypp", ypp_flags, "--MD --MF $depfile", "$in -o $out" },
    depfile = "$builddir/dependencies/$out.d",
}

local pandoc_flags = {
    "--table-of-content",
}

local html_flags = {
    pandoc_flags,
    "--to html5",
    "--css", "$$PANDOC_USER_DATA_DIRECTORY/panam.css",
    "--embed-resources --standalone",
    "--mathml",
}

rule "panda_html" {
    command = {
        env,
        "export PANDA_TARGET=$out;",
        "export PANDA_DEP_FILE=$depfile;",
        "export LOGO=$logo_html;",
        "export PANDOC_USER_DATA_DIRECTORY=`pandoc -v | awk -F': *' '$$1==\"User data direct
        "panda", html_flags, "$in -o $out",
    },
    depfile = "$builddir/dependencies/$out.d",
    implicit_in = {
        "$logo_html",
```

```
    },
}

local pdf_flags = {
    pandoc_flags,
    "--number-sections",
    "--highlight-style tango",
    "--top-level-division=chapter",
}

rule "panda_pdf" {
    command = {
        env,
        "export PANDA_TARGET=$out;",
        "export PANDA_DEP_FILE=$depfile;",
        "export LOGO=$logo_pdf;",
        "panda", pdf_flags, "$in -o $out",
    },
    depfile = "$builddir/dependencies/$out.d",
    implicit_in = {
        "$logo_pdf",
    },
}

local markdown_flags = {
    pandoc_flags,
    "--to gfm",
    "--number-sections",
    "--highlight-style tango",
    "--top-level-division=chapter",
}

rule "panda_gfm" {
    command = {
        env,
        "export PANDA_TARGET=$out;",
        "export PANDA_DEP_FILE=$depfile;",
        "export LOGO=$logo_html;",
        "panda", markdown_flags, "$in -o $out",
    },
    depfile = "$builddir/dependencies/$out.d",
    implicit_in = {
        "$logo_html",
    },
}
```

```
local beamer_flags = {
    "--to beamer",
    "-V theme:Madrid",
    "-V colortheme:default",
}

rule "panda_beamer" {
    command = {
        env,
        "export PANDA_TARGET=$out;",
        "export PANDA_DEP_FILE=$depfile;",
        "export LOGO=$logo_pdf;",
        "panda", beamer_flags, "$in -o $out",
    },
    depfile = "$builddir/dependencies/$out.d",
    implicit_in = {
        "$logo_pdf",
    },
}

var "logo_pdf" "$builddir/logo.pdf"
var "logo_html" "$img/logo.svg"

rule "lsvg" { command = "lsvg $in $out" }

acc(all) {
    build "$logo_pdf"  { "lsvg", "logo.lua" },
    build "$logo_html" { "lsvg", "logo.lua" },
}

local fizzbuzz_md = build "$builddir/fizzbuzz.md" { "ypp", "fizzbuzz.md",
    implicit_in = {
        "$builddir/tests/result_lua.lua",
        "$builddir/tests/result_c.lua",
        "$builddir/tests/result_hs.lua",
    },
}
acc(all) {
    build "$builddir/fizzbuzz.html" { "panda_html", fizzbuzz_md },
    build "$builddir/fizzbuzz.pdf"  { "panda_pdf",  fizzbuzz_md },
    build "README.md"               { "panda_gfm",  fizzbuzz_md },
}

acc(all) {
    build("$builddir/fizzbuzz_slideshow.pdf") { "panda_beamer",
```

```
        build "$builddir/fizzbuzz_slideshow.md" { "ypp", "fizzbuzz_slideshow.md" }
    }
}
```