

Lua [placeholders]*

Erik Nijenhuis <erik@xerdi.com>

9th May 2026

This file is maintained by **Xerdi**.
Bug reports can be opened at
<https://github.com/Xerdi/lua-placeholders>.

Abstract

A package for producing ‘example’ documents — parameters rendered as placeholders — and ‘actual copy’ documents — parameters replaced with real values — driven by YAML and a small Lua core. Supports Lua \LaTeX via `lua-placeholders.sty` and plain Lua \TeX via `lua-placeholders.tex`.

Contents

1	Introduction	2
1.1	Pros	2
1.2	Cons	2
1.3	Security Considerations	2
1.4	Dependencies	3
1.4.1	YAML Support	3
1.4.2	\TeX vs \LaTeX differences	3
2	Usage	3
2.1	Configuration	4
2.2	Displaying Parameters	4
2.3	Engine bridges	4
3	Parameter Specification	6
4	References	10
5	Change Log	11
6	Example	14

*This document corresponds to `lua-placeholders` version 2.0.1, written on 2026-05-09

1 Introduction

This package is meant for setting parameters in a document programmatically via YAML. Parameters can be specified by adding a ‘recipe’ file. These recipe files describe the parameter’s type, placeholders and/or default values. From thereon, the placeholders can be displayed in the document and an ‘*example*’ document can be created. An ‘*actual copy*’ document can be created by loading additional payload files, which all must correspond to a recipe file.

A note on engines and naming. lua-placeholders is a Lua module wrapped by two thin engine bridges: a \LaTeX package, lua-placeholders.sty for use under Lua \LaTeX , and a plain-format input, lua-placeholders.tex for use under Lua \TeX . Throughout this manual, where we write \LaTeX we specifically mean Lua \LaTeX , and where we write \TeX we mean plain Lua \TeX — there is no support for any other engine, since the runtime requires Lua \TeX primitives. Each macro section also states the plain- \TeX form when it differs from the \LaTeX one (typically by lacking the optional $\langle namespace \rangle$ argument).

1.1 Pros

1. Create an ‘*example*’ or ‘*actual copy*’ document from the same source and YAML recipe.
2. Integration is as easy as compiling a normal document, especially thanks to the fallback support to JSON.
3. Supports multiple data types and formatting macros which work in most environments, like enumerate or tabular.
4. Same Lua core powers both the \LaTeX and the \TeX bridge; new macros land in both at once.

1.2 Cons

1. Requires a Lua-based engine: Lua \LaTeX or plain Lua \TeX . Other engines are not supported.
2. In order for the files to be loaded, commandline option ‘--shell-restricted’ is required.

Important: In the 1.x.x series this documentation suggested ‘--shell-escape’ instead, which allows the execution of programs and isn’t a requirement for this package since the introduction of lua-tinyyaml.

1.3 Security Considerations

When using --shell-escape instead of --shell-restricted, one should be aware of the security vulnerabilities it might introduce. While it may not seem to be a risk for most users and use cases, for best practices, one should avoid using --shell-escape, since this package is meant for seamless workflow integration, which could introduce vulnerabilities in an unknown variety of ways.

Therefore, I urge everyone that still requires the execution of programs to whitelist those programs in shell_escape_commands instead. This can be configured system-wide in texmf.cnf or within a Lua initialization script using the texconfig table, thoroughly explained in section 4.1.3, 4.2.4 and 10.4 of the Lua \TeX manual[6].

1.4 Dependencies

The Lua core has *no hard dependency* on any \LaTeX package. The \LaTeX bridge (.sty) loads `textcomp` for the placeholder symbols and `xspace` for trailing-space handling inside the `paramobject` environment; the \TeX bridge (.tex) needs neither. The following packages are *optional* — the \LaTeX bridge falls back gracefully if they are absent:

- `numprint[3]` for number formatting via `\paramnumberformat`; without it, numeric values render as-is.
- `isodate[2]` for date formatting via `\paramdateformat` when `lua-tinyyaml` produces a Lua date table from a YYYY-MM-DD payload scalar.

Booleans are handled by a self-contained `\newif`-style flag bridge built on `\let` and `\csname`; `ifthen` is no longer required.

1.4.1 YAML Support

During the 1.x.x series the preferred YAML implementation changed from `lyaml[1]` to `lua-tinyyaml[4]`. The reason for this change is that `lua-tinyyaml` doesn't require any platform-specific dependencies, such as `libYAML[5]`.

The older YAML implementation will still function for older installations that do not have `lua-tinyyaml`. As before, when no YAML implementation is found, `lua-placeholders` will fall back to JSON support.

1.4.2 \TeX vs \LaTeX differences

The Lua core is identical for both engines; only the bridge file changes. Concretely:

- **Optional arguments.** \LaTeX macros take an optional [`\langle namespace \rangle`]; the \TeX forms have no optional argument and use `\curnamespace` (settable via `\setnamespace`).
- **Environments.** The `paramobject` environment is a `\begin/\end` pair under \LaTeX and a `\paramobject/\endparamobject` macro pair under \TeX .
- **Hooks.** Under \LaTeX each `\loadrecipe` fires `namespace/\langle name \rangle` via `lthooks`, and each `\loadpayload` fires `namespace/\langle name \rangle/loaded`; under \TeX no hook system is available so the emissions are skipped at the Lua level. The parameter state is fully populated either way — only attached callbacks are unavailable in \TeX .
- **Booleans.** Same call shape on both sides (`\paramnewbool` / `\paramsetbool`); the bridge backs them with `\let`-based flags rather than `\newif` so they work identically without `ifthen`.
- **Engine wrappers** (`\paramnumberformat`, `\paramdateformat`, `\paramfieldterm`). \LaTeX wires them to `numprint`, `isodate` and `\xspace`; \TeX leaves them as plain pass-throughs. See §2.3.

2 Usage

This section describes the basic commands of `lua-placeholders`. For more detail about type specific commands or the behavior of types with commands described here, see section 3.

2.1 Configuration

- `\strictparams` In order to give an error when values are missing, the `\strictparams`¹ command can be used. Make sure to do it before loading any `<recipe>` and `<payload>` files. Identical form in \TeX .
- `\loadrecipe` In order to load a recipe the macro `\loadrecipe[<namespace>]{<filename>}` can be used. Where the `<filename>` is a YAML file with its corresponding extension. The optional `<namespace>` is only a placeholder in order to prevent any conflicts between duplicate `<key>`s. If left out, the `<namespace>` defaults to the base name of the filename.
- `\loadpayload` The same behaviour counts for `\loadpayload[<namespace>]{<filename>}`. The order of loading `<recipe>` and `<payload>` files doesn't matter. If the `<payload>` file got loaded first, it will be yielded until the corresponding `<recipe>` file is loaded. The hook emissions on each load are described in section 1.4.2 (\LaTeX only).
- Plain \TeX form: `\loadrecipe{<filename>}` and `\loadpayload{<filename>}` take only the filename. The active namespace is whatever `\curnamespace` expands to at call time — defaulting to `\jobname` and adjustable with `\setnamespace`.
- All other macros of this package also take the optional `<namespace>`, which by default is equal to `\jobname`. This default `<namespace>` can be changed with `\setnamespace{<new default namespace>}`. Identical form in \TeX .
- `\setnamespace`

2.2 Displaying Parameters

- For displaying variables, the commands `\param` and `\PARAM` share the same interface.
- `\param` The most trivial, displaying the variable as-is, is `\param[<namespace>]{<key>}`. The
- `\PARAM` `\PARAM` however, shows the value as upper case. Plain \TeX forms: `\param{<key>}` and `\PARAM{<key>}` (no optional argument; namespace comes from `\curnamespace`).
- In some cases, it's required to output the text without any \TeX related functionality. Another case is that some environments don't take macros with optional arguments well. For these cases there is `\rawparam[<namespace>]{<key>}`, which takes the namespace as mandatory argument, instead of optional, and doesn't output fancy \TeX placeholders. Plain \TeX form: `\rawparam{<key>}` (single argument; namespace from `\curnamespace`).
- `\rawparam`
- `\hasparam` To check whether a parameter is set, the `\hasparam[<namespace>]{<key>}{<true case>}{<false case>}` command is used. Plain \TeX form: drop the optional namespace — `\hasparam{<key>}{<true case>}{<false case>}`.

2.3 Engine bridges

The Lua core never references package-specific commands directly; it emits the following wrappers, which the bridge file (`.sty` for \LaTeX , `.tex` for \TeX) backs with engine-appropriate behaviour. Override any of them in your own preamble before the bridge is loaded if you need different formatting.

¹The `\strictparams` command is still under development.

\paramnumberformat{ $\langle N \rangle$ } Numeric values pass through here. \LaTeX default: `\numprint{ $\langle N \rangle$ }` when `numprint` is loaded, otherwise $\langle N \rangle$ as-is. \TeX default: $\langle N \rangle$ as-is.

\paramdateformat{ $\langle Y \rangle$ }{ $\langle M \rangle$ }{ $\langle D \rangle$ } YYYY-MM-DD payload scalars (parsed as a Lua date table by `lua-tinyyaml`) pass through here. \LaTeX default: `\printdateTeX{ $\langle Y/M/D \rangle$ }` when `isodate` is loaded, otherwise $\langle Y \rangle/\langle M \rangle/\langle D \rangle$ as text. \TeX default: $\langle Y \rangle/\langle M \rangle/\langle D \rangle$ as text.

\paramfieldterm Appended after each field binding inside the `paramobject` environment. \LaTeX default: `\xspace`. \TeX default: empty (the author writes `\name\xspace` or `\name{}` explicitly).

3 Parameter Specification

Every parameter specified has a *type* set. Optionally there is a choice between setting a *default* or a *placeholder* for the parameter.

bool Next to the textual representation of *true* and *false*, the bridge tracks each boolean as a `\newif`-style flag. Therefore, only the *default* setting makes sense.

	Recipe	Payload
	<pre> 1 bool example: 2 type: bool 3 default: false </pre>	<pre> 1 bool example: true </pre>
<code>\param</code> <code>\ifparam</code>	<p>With a boolean type the <code>\param[<i>namespace</i>]{<i>name</i>}</code> returns either <i>true</i> or <i>false</i>. Additionally, it provides the <code>\ifparam[<i>namespace</i>]{<i>name</i>}{<i>true code</i>}{<i>false code</i>}</code> command for top level boolean types. The flag is set via <code>\paramsetbool</code> and tested with <code>\csnameif<i>name</i>\endcsname</code>, which supports spaces in names without any package dependency (no <code>ifthen</code> is needed). Plain T_EX form: <code>\ifparam{<i>name</i>}{<i>true code</i>}{<i>false code</i>}</code> (no optional namespace).</p>	

string representing a piece of text. Values are passed through to T_EX verbatim — only embedded newlines are normalised to spaces. T_EX-special characters (`\`, `%`, `#`, `&`, `_`, `$`, `^`, `~`) are *not* escaped: pass an unescaped `%` and T_EX will treat the rest of the value as a comment, pass a bare `&` and you'll get a misplaced-alignment-tab error, etc. Escape them in the YAML payload using their usual L^AT_EX forms (e.g. `\%` for a literal percent, `\&` for an ampersand, `\textbackslash` for a backslash).

	<pre> 4 string example: 5 type: string 6 placeholder: A string </pre>	<pre> 2 string example: PeelInc. </pre>
<code>\param</code>	<p>A string type can easily be placed in L^AT_EX using the <code>\param</code> command.</p>	

YAML date caveat. When the YAML payload is parsed by `lua-tinyyaml`, scalars matching the ISO 8601 date pattern `YYYY-MM-DD` are returned as a Lua table `{year=, month=, day=}` rather than a string. `lua-placeholders` hands the three components to `\paramdateformat{Y}{M}{D}` (see 2.3); the L^AT_EX bridge wraps that in `\printdateTex` from `isodate[2]` when available, and the T_EX bridge prints `<Y>/<M>/<D>` as text. The `date example` field below illustrates this:

<pre> 7 date example: 8 type: string 9 placeholder: A date </pre>	<pre> 3 date example: 2024-04-02 </pre>
---	---

Note that `\rawparam` bypasses this conversion and will not produce useful output for a date-typed value. The `\yaml` fallback and JSON do not perform this conversion — the same `YYYY-MM-DD` payload is delivered as a plain string instead.

number representing a number, like the number type of Lua. In most cases it's necessary to use `<default>` instead of `<placeholder>`, especially when the number is used in calculations, since a placeholder will cause errors in \LaTeX calculations.

```
10 number example:
11   type: number
12   default: -1.21
```

```
4 number example: 1.21
```

`\param` A number type can be used with `\param`, just like the string type. In the 1.x.x series there was a special command `\numparam`, which is now deprecated as it is the default implementation for number types using `\param`. Numeric values pass through `\paramnumberformat` (see 2.3); the \LaTeX bridge wraps in `\numprint` from `numprint[3]` when loaded, the \TeX bridge prints the value as-is. The same behaviour applies to number values inside a list, object or table. If you need a nonformatted version of the number, use `\rawparam` instead.

list representing a list of values. The simplest form sets the item shape with `<item type>` (one of string, number, bool). For nested complex item types, use the long form `<item>`: a full nested parameter spec, e.g. `item: {type: object, fields: ...}` or `item: {type: table, columns: ...}`. A `<default>` setting can be set. Due to its structure, a `<placeholder>` would be somewhat incompatible with the corresponding macros; instead, set placeholder content as children of the `<default>` list.

```
13 list example:
14   type: list
15   item type: string
16   default:
17     - A string
18     - A second string
```

```
5 list example:
6   - Tomatoes
7   - Potatoes
```

`\param` Command `\param` concatenates every item with command `\paramlistconjunction`.
`\paramlistconjunction` By default, the conjunction is set to `' , ~ '`.

`\forlistitem` There's also the `\forlistitem[<namespace>]{<name>}{<cname>}` command, which takes an additional `<cname>` and will execute it for every item in the list. For primitive item types the `<cname>` is invoked with the value as its single argument. For complex item types (object, list, table) each item is pushed onto the lookup context: object items expose every field as a direct `\<fieldname>` macro, while list/table items are bound under the synthetic key `self` so the row macro can drill in with `\fortablerow{<self>}{<...>}` or `\forlistitem{<self>}{<...>}`. Plain \TeX form: `\forlistitem{<name>}{<cname>}` (no optional namespace).

object representing a list of key value pairs. This parameter type requires a `<fields>` specification to be set. Each field is itself a parameter spec, so bool, number, string as well as nested list, object and table types are all permitted.

```

19 object example:
20   type: object
21   fields:
22     name:
23       type: string
24       placeholder: Your name
25     email:
26       type: string
27       placeholder: Your email
28     grade:
29       type: number
30       default: 5.5

```

```

8  object example:
9    name: John Doe
10   email: j.doe@example.com
11   grade: 9.5

```

`\paramfield`

There is no support for the `\param` command. In order to show to contents there is the `\paramfield[<namespace>]{<name>}{<field>}` command. However, unlike the common command `\param`, the command `\hasparam` does work with object types. Plain \TeX form: `\paramfield{<name>}{<field>}`.

`paramobject (env)`

There's also the `paramobject` environment, which takes an optional *<namespace>* and the *<name>* of the object as arguments and then defines for every primitive field name a corresponding command. Each binding is appended with `\paramfieldterm` (\LaTeX under \LaTeX , empty under \TeX) so the author doesn't have to end the command with accolades `{}` to get the expected output. Complex (list, object, table) sub-fields are not flattened to a macro; they remain reachable inside the environment via `\forlistitem{<field>}{<...>}`, `\paramfield{<field>}{<...>}` (for nested objects), `\fortablerow{<field>}{<...>}`, etc. Plain \TeX form: `\paramobject{<name>} ... \endparamobject` (a macro pair instead of a `\begin/\end` environment).

table representing a table. This parameter type requires a *<columns>* specification to be set. The *<columns>* describes each column by name with its own type specification. Each column may be of any type — `bool`, `number`, `string`, or a nested list, object or table — and is reached from the row macro via the corresponding type-specific command (`\paramfield`, `\forlistitem`, nested `\fortablerow`).

```

31 table example:
32   type: table
33   columns:
34     description:
35       type: string
36       placeholder: The description
37     price:
38       type: number
39       placeholder: The price

```

```

12 table example:
13   - description: Peeling ↩
14     tomatoes
15     price: 50
16   - description: Peeling ↩
17     potatoes
18     price: 25

```

`\fortablerow`

Like the object, the table has no support for `\param`, but comes with a table-specific command `\fortablerow[<namespace>]{<name>}{<cname>}`. The control sequence name *<cname>* is a user-defined command with no arguments, contain-

ing any of the column names in a command form. For example, the name `example` would be accessible as `\example` in the user-defined command body. Cells do not require braces; the column macros expand naturally inside \TeX , so `\example` works the same way as `\example{}`. Plain \TeX form: `\fortablerow{<name>}{<cname>}` (no optional namespace).

Internally, `\fortablerow` pushes a frame onto a context stack shared with `paramobject` and `\forlistitem`, then rebinds every primitive column to a global control sequence via `token.set_macro` just before each invocation of `<cname>`. Because the column macros are real \TeX control sequences — not string substitutions — prefix collisions between column names (e.g. a column `item` alongside `item_name`) are no longer a concern, and unreferenced columns in `<cname>` are silently dropped instead of leaking partial substitutions.

Underscore in a column name. `\token.set_macro` is catcode-agnostic, so it can define a control sequence `\item_name` regardless of the catcode of `_` at the time `\fortablerow` runs. *Using* that control sequence in `<cname>`, however, requires `_` to have letter catcode (11) when the row macro is parsed — otherwise \TeX reads `\item_name` as `\item` followed by the subscript token `_` and the letters `name`. The standard idiom is to wrap the row macro definition in `\ExplSyntaxOn ... \ExplSyntaxOff`, which temporarily flips `_` (and `:`) to letter catcode for the parser:

```
\ExplSyntaxOn
\cs_new:Npn \rowU {\item & \item_name & \unit_price \}
\ExplSyntaxOff
\fortablerow{my-table}{rowU}
```

Once `\rowU` has been tokenised, the catcode of `_` no longer matters at expansion time.

4 References

- [1] Andrew Danforth. *lyaml*. <https://github.com/gvvaughan/lyaml> and <https://luarocks.org/modules/gvvaughan/lyaml>. Accessed: 6 January, 2024.
- [2] Harald Axel Sommerfeldt Eslava. *The isodate package. Tune the output format of dates according to language*. Version 2.70. Sept. 13, 2010. URL: <https://ctan.org/pkg/isodate> (visited on 05/09/2026).
- [3] Harald Harders. *The numprint package. Print numbers with separators and exponent if necessary*. Version 1.39. 2012. URL: <https://ctan.org/pkg/numprint> (visited on 02/12/2024).
- [4] Zeping Lee. *The lua-tinyyaml package. A tiny YAML (subset) parser for pure Lua*. Version 0.4.3. URL: <https://ctan.org/pkg/lua-tinyyaml> (visited on 02/12/2024).
- [5] *libYAML*. <https://pyyaml.org/wiki/LibYAML> and https://packages.msys2.org/package/mingw-w64-x86_64-libyaml. Accessed: 6 January, 2024.
- [6] The LuaTeX Team. *The luatex package. The LuaTeX engine*. URL: <http://luatex.org> (visited on 04/05/2024).

5 Change Log

Release 2.0.1 _____ 9th May 2026

4ed04fc Update workflows: specify ‘TEXMFHOME’ in publish; upgrade artifact action to v4 in build

Release 2.0.0 _____ 9th May 2026

38795b4 Release 2.0.0

e3e66e3 Remove ‘ifthen’ dependency; improve documentation for plain LuaTeX, parameter types, and engine-specific behaviors

8163f93 Refactor token handling and formatting for plain LuaTeX; add comprehensive plain LuaTeX support and tests

e29dc0c Add hook detection for plain LuaTeX; gate hook usage; add basic plain LuaTeX test

b4af6ee Improve placeholder handling for complex types; add tests for empty-data rendering

e640adf Add support for nested list, object, and table compositions; update tests and documentation

1333b34 Refactor context stack for parameter resolution; add list and object support; update tests and documentation

d5b9f34 Implement row-specific parameter resolution; add tests and examples for list and object cell types

e4a95b6 Document and test table support for special characters in cell values

2ee9c6d Refactor row-binding logic for table parameters; add tests and documentation.

268cbdb Fix fixed dates in expected output

fe5cb40 Document and test Lua dates

52e13fa Add example test

_____ 4th December 2025

ead96e2 Add date support

_____ 18th September 2024

b47b281 Fix build files

8945393 Ignore merge commits in docs

_____ 23rd July 2024

38988b9 Create FUNDING.yml

_____ 6th April 2024

62469a4 Add git as shell escape cmd

c62c3a2 Make tinyyaml default YAML parser

f2850c1 Describe security considerations

3d4708b Improve Change Log

de69206 Enlarge page width

179f7cc Add page headings to the manual

_____ 2nd April 2024

7e46622 Add latexmk config and exclude config file for packaging

Release 1.0.3 _____ 2nd April 2024

ab5ecd5 Release 1.0.3

66a63f7 Add documentation

- Address YAML preferred implementation - Describe custom LaTeX hooks

_____ 29th March 2024

31864e7 Record YAML files

_____ 27th March 2024

0200615 Fix LaTeX Hooks

Predeclare namespace hooks when \loadrecipe is called

_____ 23rd February 2024

bda1fe7 Add status badge

1616b2d Build master branch on push

0598a5c Create release in draft mode

ac5b8bd Fix Makefile for win32

Release 1.0.2 _____ 21st February 2024

8469b2a Release 1.0.2

77b5a3e Add Continuous Integration and Delivery

_____ 20th February 2024

812b746 Fix Makefile for Windows

_____ 19th February 2024	cc91bd4 Refactor project name	_____ 11th January 2024
7389753 Fix faulty line endings and fix Makefile		
_____ 1st February 2024	8ee2ed3 Set listings columns to fullflexible	_____ 10th January 2024
7890fc8 Update README.md		
Add CTAN version badge	d415eb2 Add tar prefix	
Release 1.0.1 _____ 12th February 2024	Release 0.0.1 _____ 9th January 2024	
bfa9c9f Release 1.0.1	89f61f2 Update package date	
ca8a0f0 Add documentation	c00eb0d Set version in tarball filename	
- Add a git changelog - Note numprint dependency -	ed745f0 Add Makefile and README	
Describe new behavior of number type	c4443c9 Add license	
69c4cba Fix info print statement	cd3cbb1 Update the docs	_____ 6th January 2024
d93d88e Update example document	09e6f94 Add prerequisites to docs	_____ 5th January 2024
- Uses floating number for formatting demonstration		
purposes - Adds an inner number type for object - Add	579c3c7 Add \numparam macro	_____ 4th January 2024
numprint support		
2ec041f Enhance number output	014423e Add uppercase variant for params	
Numbers will be formatted with numprint if present.	1f9f19c Add \PARAM and \rawparam macros	
This is especially useful when numbers are used in ta-	7939698 Move commandline features to xdp and add	
bles or objects, since those environments are hard to	namespace hooks	_____ 19th December 2023
typeset using lua-placeholders (expansion order).		
Release 1.0.0 _____ 23rd January 2024	a9127d3 Update manual	
1c5fd4f Release 1.0.0	bde1a5d Refactor examples directory	
632681d Update documentation	6be9088 Add namespace support	_____ 7th December 2023
20db0a3 Add tiny yaml support	2b3d747 Cleanup	
Adds fallback support for YAML files with package	a91cec5 Fix table format in a macro way	
lua-tinyyaml. Included for Windows users, where	a12caf9 Provide better examples	_____ 1st December 2023
libYAML is too hard to install.	183b25c Make container types able to have other com-	
_____ 15th January 2024	plex children in Lua	
646e7cc Fix license header in manual	ee2376a Fix formatting table rows	_____ 27th November 2023
_____ 12th January 2024	e73810b Split up lua files	
0e8f28b Replace last occurrence of ELPI		
Release 0.1.0 _____ 12th January 2024		
0e5fa24 Set version 0.1.0		

4035601 Update the docs

17th November 2023


25th November 2023

063aecc Add sources

99a1342 Init

6 Example

The source file `example.tex` is a perfect demonstration of all macros in action. It shows perfectly what happens when there's a `<payload>` file loaded and when not. The result

of this example  is attached in the digital version of this document.

Listing 1: `example.tex`

```

20 ''
21 \documentclass{article}
22 \usepackage{gitinfo-lua}
23 \usepackage{lua-placeholders}
24 \usepackage{listings}
25 \usepackage{amsmath}
26 \usepackage{calc}
27 \usepackage[dutch,english]{babel}
28 \usepackage[autolanguage]{numprint}
29 \usepackage[english]{isodate}
30
31 \loadrecipe[\jobname]{example-specification.yaml}
32
33 \setlength{\parindent}{0pt}
34
35 \begin{document}
36   \title{Lua \paramplaceholder{placeholders} Example\thanks{This example corresponds to \texttt{↵
      lua-placeholders} version \gitversion{} written on \gitdate.}}
37   \author{\dogitauthors{\\}}
38   \maketitle
39
40   \section*{Basics}
41   Wrong parameter:\\
42
43   \lstinline[style=TeX,morekeywords={param}]{\param{non existing}}|
44   $\implies$
45   \param{non existing}\\
46
47   Conditional Parameter:\\
48
49   \lstinline[style=TeX,morekeywords={hasparam}]{\hasparam{list example}{is set}{is not set}}|
50   $\implies$
51   \hasparam{list example}{is set}{is not set}
52
53   \section*{Before values loaded}
54
55   Boolean example:\\
56
57   \lstinline[style=TeX,morekeywords={param}]{\param{bool example}}|
58   $\implies$
59   \param{bool example}\\

```

```

60
61 \lstineline[style=TeX,morekeywords={ifparam}]{\ifparam{bool example}{TRUE}{FALSE}}|
62 $\implies$
63 \ifparam{bool example}{TRUE}{FALSE}\
64
65 String example:\
66
67 \lstineline[style=TeX,morekeywords={param}]{\param{string example}}|
68 $\implies$
69 ``\param{string example}''\
70
71 Date example (a string whose payload value matches \texttt{YYYY-MM-DD}):\
72
73 \lstineline[style=TeX,morekeywords={param}]{\param{date example}}|
74 $\implies$
75 \param{date example}\
76
77 Number example:\
78
79 \lstineline[style=TeX,morekeywords={rawparam}]{\rawparam{\jobname}{number example}}|
80 $\implies$
81 \rawparam{\jobname}{number example}\
82
83 \lstineline[style=TeX,morekeywords={param}]{\param{number example}}|
84 $\implies$
85 \lstineline[style=TeX,morekeywords={numprint}]{\numprint{\rawparam{\jobname}{number example}}\verb|}|
86 $\implies$
87 \param{number example}\
88
89 \clearpage
90
91 Number in foreign language:\
92
93 \lstineline[style=TeX,morekeywords={param,select language}]{\select language{dutch}\param{number example}}|
94 $\implies$
95 \begingroup\select language{dutch}\param{number example}\endgroup\
96
97 List example:\
98
99 \lstineline[style=TeX,morekeywords={param}]{\param{list example}}|
100 $\implies$
101 \param{list example}\
102
103 \begin{lstlisting}[language={LaTeX}TeX,morekeywords={formatitem,forlistitem}]
104 \begin{enumerate}
105 \newcommand\formatitem[1]{\item #1}
106 \forlistitem{list example}{formatitem}
107 \end{enumerate}

```

```

108 \end{lstlisting}
109 $\implies$
110 \begin{enumerate}
111   \newcommand\formatitem[1]{\item #1}
112   \forlistitem{list example}{formatitem}
113 \end{enumerate}
114
115 Object example:\\
116
117 \lstinline[style=TeX,morekeywords={paramfield}]{\paramfield{object example}{name}}\\
118 \lstinline[style=TeX,morekeywords={paramfield}]{\paramfield{object example}{email}}\\
119 \lstinline[style=TeX,morekeywords={paramfield}]{\paramfield{object example}{grade}}\\
120 $\implies$
121 \paramfield{object example}{name}
122 \paramfield{object example}{email}
123 \paramfield{object example}{grade}\\
124
125 \begin{lstlisting}[style=TeX,morekeywords={name,email,grade}]
126 \newcommand\name{...}
127 \begin{paramobject}{object example}
128   \name \email \grade
129 \end{paramobject}
130 % And here it works again
131 \name
132 \end{lstlisting}
133 $\implies$
134 \newcommand\name{...}%
135 \parbox{\linewidth}{
136   \begin{paramobject}{object example}
137     \name \email \grade
138   \end{paramobject}
139   \name
140 }\\
141
142 Table example:\\
143
144 \begin{lstlisting}[style=TeX,morekeywords={nprouddigits,npnround,formatrow,fortablerow,↵
  description,price}]
145 \nprouddigits{2}
146 \newcommand\formatrow{\description & \price \\}%
147 \begin{tabular}{l | l}
148   \textbf{Description} & \textbf{Price} \\ \hline
149   \fortablerow{table example}{formatrow}
150 \end{tabular}
151 \npnround
152 \end{lstlisting}
153 $\implies$
154 \nprouddigits{2}
155 \newcommand\formatrow{\description & \price \\}%
156 \begin{tabular}{l | l}

```



```

157     \textbf{Description} & \textbf{Price} \\ \hline
158     \fortablerow{table example}{formatrow}
159 \end{tabular}
160 \nporound
161
162
163 \section*{After values loaded}
164 \loadpayload[\jobname]{example.yaml}
165
166 Boolean example:\\
167
168 \lstinline[style=TeX,morekeywords={param}]\param{bool example}|
169 $\implies$
170 \param{bool example}\\
171
172 \lstinline[style=TeX,morekeywords={ifparam}]\ifparam{bool example}{TRUE}{FALSE}|
173 $\implies$
174 \ifparam{bool example}{TRUE}{FALSE}\\
175
176 String example:\\
177
178 \lstinline[style=TeX,morekeywords={param}]\param{string example}|
179 $\implies$
180 ``\param{string example}''\\
181
182 Date example:\\
183
184 \lstinline[style=TeX,morekeywords={rawparam}]\rawparam{\jobname}{date example}|
185 $\implies$
186 \rawparam{\jobname}{date example}\\
187
188 \lstinline[style=TeX,morekeywords={param}]\param{date example}|
189 $\implies$
190 \param{date example}\\
191
192 Number example:\\
193
194 \lstinline[style=TeX,morekeywords={rawparam}]\rawparam{\jobname}{number example}|
195 $\implies$
196 \rawparam{\jobname}{number example}\\
197
198 \lstinline[style=TeX,morekeywords={param}]\param{number example}|
199 $\implies$
200 \lstinline[style=TeX,morekeywords={numprint}]\numprint{\rawparam{\jobname}{number example}} \leftarrow
    \verb|}|
201 $\implies$
202 \param{number example}\\
203
204 Number in foreign language:\\
205

```

```

206 \lsthline[style=TeX,morekeywords={param,select language}]{\select language{dutch}\param{number ↵
    example}}\l
207 $\implies$
208 \lsthline[style=TeX,morekeywords={numprint}]{\numprint{\rawparam{\jobname}{number example}\ ↵
    \lsthline{}}
209 $\implies$
210 \begingroup\select language{dutch}\param{number example}\endgroup\l
211
212 List example:\l
213
214 \lsthline[style=TeX,morekeywords={param}]{\param{list example}}
215 $\implies$
216 \param{list example}\l
217
218 \begin{lstlisting}[language={LaTeX}TeX],morekeywords={formatitem,forlistitem}]
219 \begin{enumerate}
220 \newcommand\formatitem[1]{\item #1}
221 \forlistitem{list example}{formatitem}
222 \end{enumerate}
223 \end{lstlisting}
224 $\implies$
225 \begin{enumerate}
226 \newcommand\formatitem[1]{\item #1}
227 \forlistitem{list example}{formatitem}
228 \end{enumerate}
229
230 Object example:\l
231
232 \lsthline[style=TeX,morekeywords={paramfield}]{\paramfield{object example}{name}}\l
233 \lsthline[style=TeX,morekeywords={paramfield}]{\paramfield{object example}{email}}\l
234 \lsthline[style=TeX,morekeywords={paramfield}]{\paramfield{object example}{grade}}\l
235 $\implies$
236 \paramfield{object example}{name}
237 \paramfield{object example}{email}
238 \paramfield{object example}{grade}\l
239
240 \begin{lstlisting}[style=TeX,morekeywords={name,email,grade}]
241 \newcommand\name{...}
242 \begin{paramobject}{object example}
243 \name \email \grade
244 \end{paramobject}
245 % And here it works again
246 \name
247 \end{lstlisting}
248 $\implies$
249 \parbox{\linewidth}{
250 \begin{paramobject}{object example}
251 \name \email \grade
252 \end{paramobject}
253 \name

```

```

254 }\\
255
256 Table example:\\
257
258 \begin{lstlisting}[style=TeX,morekeywords={nprouddigits,npnoround,formatrow,fortablerow,↵
    description,price}]
259 \nprouddigits{2}
260 \newcommand\formatrow{\description & \price \\}%
261 \begin{tabular}{l | l}
262 \textbf{Description} & \textbf{Price} \\ \hline
263 \fortablerow{table example}{formatrow}
264 \end{tabular}
265 \npnoround
266 \end{lstlisting}
267 $\implies$
268 \nprouddigits{2}%
269 \begin{tabular}{l | l}
270 \textbf{Description} & \textbf{Price} \\ \hline
271 \fortablerow{table example}{formatrow}
272 \end{tabular}
273
274 \section*{Payload File}
275 \lstinputlisting[language=YAML,numbers=left,xleftmargin={15pt},caption={example.yaml},columns=↵
    fullflexible]{example.yaml}
276
277 \section*{Specification File}
278 \lstinputlisting[language=YAML,numbers=left,xleftmargin=15pt,caption={example-specification.yaml↵
    },columns=fullflexible]{example-specification.yaml}
279 \end{document}

```