

emo•ji for all (LaTeX engines)

Robert Grimm

Version v1.0 (2023/99/99)

Abstract

Emo implements the `\emo{<emoji-name>}` command for including color emoji such as `\emo{desert-island}` for 🌴 or `\emo{parrot}` for 🦜 in your documents independent of LaTeX engine. The implementation uses the Noto color emoji font if the engine supports it and includes PDF graphics otherwise. It also supports conversion to HTML with either LaTeXML or TeX4ht. Next, PDF graphics are automatically derived from Noto's SVG sources, so the visual appearance is very similar. The source repository is at <https://github.com/apparebit/emo>. Emo may come in particularly handy when dealing with academic publishers that provide only minimal support for non-Latin scripts (cough, ACM, cough).

Contents

1	Installation	2
2	Usage	3
2.1	One Main Macro	3
2.1.1	Naming Scheme	3
2.1.2	Default Configuration	5
2.2	Two Optional Macros	5
2.3	Conversion to HTML	5
3	Configuration	5
3.1	Running the Configuration Script	5
4	Copyright and Licensing	6
5	Implementation	6
5.1	Package Options	7
5.2	Package Dependencies and Backend Selection	7
5.3	The Emoji Table	8
5.4	Internal Macros	9
5.5	User Macros and Hooks	10
5.5.1	Background: Variability vs Code Duplication	10
5.5.2	A Skeleton of Render Hooks	11
5.5.3	The User Macros	12
5.5.4	Activating the Hooks	12

6	LaTeXML Binding	13
7	Testing and Documenting Emo	15
7.1	Emo's Support Package	15
7.2	Engine Test Runner and Report	18
7.3	Simple HTML Test Document	19

1 Installation

The emo package is available through its [source repository](#) or through [CTAN](#). Installation is fairly straightforward, though it does involve a lot more files than usual.

1. Start by extracting this package's files from `emo.dtx` by running:

```
$ pdftex emo.dtx
```

Do *not* use `tex`; it mangles the embedded `README.md`. `pdflatex` also extracts the files and then builds the documentation. Embedded files are `build.sh`, `emo.ins`, `emo.sty`, `emo.sty.ltxml`, `emo-support.sty`, `canary.tex`, `demo.tex`, and `README.md`. Extraction will overwrite existing files with the same name without asking.

2. Test the package and build its documentation by making `build.sh` executable and then running it:

```
$ chmod +x build.sh; ./build.sh
```

The shell script does three things: First, it tests `emo` by compiling `canary.tex` with `pdfTeX`, `XeTeX`, as well as `LuaTeX` and generating `canary.pdf` with the results from the three tests. Second, it tests `emo` by compiling `demo.tex` with `LaTeXML` as well as `TeX4ht` and generating `demo-latexml.html` as well as `demo-tex4ht.html`, which are framed together by `demo.html`. Third, it builds the package documentation in `emo.pdf` with all bells and indices.

3. Get started reconfiguring supported emoji by running:

```
$ python config/emo.py -h
```

For more detailed instructions, see §3 below.

4. Put the following files somewhere LaTeX can find them. In a pinch, your current project's directory will do. However, `emo`'s installation potentially comprises thousands of files. So, you probably want to use a dedicated directory and add that to the search path for LaTeX, e.g., by setting the `TEXINPUTS` environment variable.

- (a) `emo.sty` with the package implementation;
- (b) `emo.sty.ltxml` with the binding for [LaTeXML](#);
- (c) `emo.def` with the emoji table;
- (d) `emo-lingchi.ttf` with the two glyphs for `\lingchi`;
- (e) `emo-graphics` with the fallback PDF graphics.

TeX Live requires that each package's files have unique names. For that reason, the PDF graphics in the `emo-graphics` directory start with the `emo-` prefix as well.

When running on the LuaLaTeX engine, the emo package also uses the Noto color emoji (NotoColorEmoji.ttf) and Linux Libertine (LinLibertine_R.otf) fonts, with the latter used for rendering \YHWH only. Neither file is included with emo’s distribution, since both of them are distributed with major TeX distributions already. If they are not included with your LaTeX distribution, you can find them on CTAN. The emo-lingchi.ttf font distributed with emo is a two glyph subset of NotoSerifTC-Regular.otf, i.e., the traditional Chinese version of Noto serif.



2 Usage

As usual, you declare your document’s dependency on emo with \usepackage{emo}. In addition to the unadorned form, emo takes up to two options:

extra Also define the \lingchi and \YHWH macros, which produce 凌遲 and יהודה.

index Create an emoji index tagged emo with the .edx extension for the raw index and the .end extension for the processed index. This option relies on the index package, generates the raw .edx file, but does not build or use the processed index.

2.1 One Main Macro

\emo An \emo{<emoji-name>} invocation expands to the named emoji. For LuaLaTeX, it uses the Noto color emoji font. For all other engines, it uses PDF graphics. That way, \emo{desert-island} results in  and \emo{parrot} results in .

Since LaTeX tends to produce a lot of command line noise about underfull boxes and loaded fonts, it’s a easy to miss meaningful warnings. For that reason, \emo expands to an attention-seeking error message upon undefined emoji names. For example, \emo{boo} produces Bad \emo{boo}.

2.1.1 Naming Scheme

With a few exceptions, emo’s names for emoji are automatically derived from their Unicode names, with letters converted to lowercase, punctuation such as commas, colons, quotes, and parentheses stripped, and interword spaces replaced by dashes. Furthermore, instead of the rather verbose dark-skin-tone, medium-dark-skin-tone, etc modifiers, emo uses the more succinct darkest, darker, medium, lighter, and lightest.

For some names, emo goes further by hard-coding shorter names. Those names are listed in Table 1.

Emo’s emo.def contains the names and codepoints of all currently supported emoji in Unicode display order. Emo’s distribution also includes the emoji-test.txt file, which is part of [Unicode TR-51](#) and contains the names and codepoints of all *potentially* supported emoji, i.e., all emoji, also in Unicode display order. It further organizes emoji into groups and subgroups, with the current (sub)group being the one named on the closest line above the emoji that starts with # (sub)group:. As described in the next section, the group and subgroup names can be used during configuration for concisely naming a large number of emoji.

Table 1: Exceptional emoji names

Transformed Unicode Name	Emo Replacement Name
a-button-blood-type	a-button
ab-button-blood-type	ab-button
b-button-blood-type	b-button
o-button-blood-type	o-button
bust-in-silhouette	bust
busts-in-silhouette	busts
flag-european-union	eu
globe-showing-america	globe-america
globe-showing-asia-australia	globe-asia-australia
globe-showing-europe-africa	globe-africa-europe
hear-no-evil-monkey	hear-no-evil
index-pointing-at-the-viewer	index-pointing-at-viewer
index-pointing-at-the-viewer-darkest	index-pointing-at-viewer-darkest
index-pointing-at-the-viewer-darker	index-pointing-at-viewer-darker
index-pointing-at-the-viewer-medium	index-pointing-at-viewer-medium
index-pointing-at-the-viewer-lighter	index-pointing-at-viewer-lighter
index-pointing-at-the-viewer-lightest	index-pointing-at-viewer-lightest
keycap-*	keycap-star
keycap-#	keycap-hash
keycap-0	keycap-zero
keycap-1	keycap-one
keycap-2	keycap-two
keycap-3	keycap-three
keycap-4	keycap-four
keycap-5	keycap-five
keycap-6	keycap-six
keycap-7	keycap-seven
keycap-8	keycap-eight
keycap-9	keycap-nine
keycap-10	keycap-ten
magnifying-glass-tilted-left	loupe-left
magnifying-glass-tilted-right	loupe-right
palm-down-hand	palm-down
palm-down-hand-darkest	palm-down-darkest
palm-down-hand-darker	palm-down-darker
palm-down-hand-medium	palm-down-medium
palm-down-hand-lighter	palm-down-lighter
palm-down-hand-lightest	palm-down-lightest
palm-up-hand	palm-up
palm-up-hand-darkest	palm-up-darkest
palm-up-hand-darker	palm-up-darker
palm-up-hand-medium	palm-up-medium
palm-up-hand-lighter	palm-up-lighter
palm-up-hand-lightest	palm-up-lightest
rolling-on-the-floor-laughing	rofl
see-no-evil-monkey	see-no-evil
speak-no-evil-monkey	speak-no-evil

2.1.2 Default Configuration

Emo’s default configuration includes the following emoji, sorted in Unicode display order and arranged into the corresponding groups and subgroups. consistently with Unicode groups and display order. **TODO**

2.2 Two Optional Macros

`\lingchi` The `\lingchi` and `\YHWH` macros take no arguments and produce 凌遲 and יהוה. They are only available if `emo` is used with the `extra` option. The former renders the Chinese term for “death by a thousand cuts.” While originally an execution method, the term applies to surprisingly many software systems as well. The latter produces the Tetragrammaton, the Hebrew name for God. Observant Jews never utter what’s written, not even in their thoughts, substituting *Adonai* (“My Lord”), *Elohim* (“God”), or *HaShem* (“The Name”) instead. In my mind, that nicely mirrors the very incomprehensibility of יהוה. Both macros preserve a subsequent space as space, no backslash needed.

2.3 Conversion to HTML

Emo supports conversion to HTML with either [LaTeXML](#) or [TeX4ht](#). LaTeXML support is implemented by a separate “binding” against LaTeXML’s Perl API. I chronicled my exploration of suitable options leading to that less than ideal choice in a [GitHub issue](#). TeX4ht support is implemented by the `emo` package itself. It requires processing with LuaLaTeX e.g., by passing `-l` or `--lua` to the `make4ht` tool.

3 Configuration

Emo’s implementation is actually split over two files: `emo.sty` is extracted from `emo.dtx` and defines the substance of the package, its options, its helper macros, and the user-visible `\emo`, `\lingchi`, and `\YHWH` macros. Currently supported emoji are defined by the emoji table in the second file, `emo.def`. For every supported emoji, the file contains a command `\emo@emoji@⟨emoji-name⟩` with the emoji’s codepoints as value.

Configuration automates the regeneration of the emoji table for arbitrary numbers of emoji. `config/emo.py` is the script and `config/emoji-test.txt` is the list of all emoji from the Unicode standard.

3.1 Running the Configuration Script

To update emo’s configuration, invoke the `config/emo.py` script:

```
$ python3 config/emo.py <selector> <selector> ...
```

Each selector may be:

- The literal `ALL` (case-sensitive) for *all* emoji.
- Name of a group in `emoji-test.txt` lowercased and with spaces replaced by dashes and ampersand `&` replaced by an `and`; e.g., `travel-and-places`.
- Name of a group, a double colon `::`, and name of a subgroup, again lowercased and with spaces replaced by dashes and `&` by an `and`; e.g., `travel-and-places::placegeographic`.

- The name of an emoji; e.g., `desert-island`.

For conjunctive group names, such as “Smileys & Emotion” (`emoji-test.txt`) or “smileys-and-emotion” (`emo.py`), the configuration script also accepts either of the two nouns as a shortcut, e.g., “smileys” or “emotion.”

For data safety, `emo.py` does not overwrite PDF graphics and hence can only *add* emoji to the configuration. To remove emoji, simply remove their PDF graphics from `emo-graphics` and then run `emo.py` without selector arguments, which updates the emoji table accordingly.

`emo.py` effectively treats `emoji-test.txt` as registry of all emoji and the filenames of PDF graphics in `emo-graphics` as `emo`’s current inventory. For all emoji named by selector arguments but not in the inventory, `emo.py` converts the SVG source graphic from the Noto color emoji sources to a PDF file and deletes the `/Page /Group` object from the the PDF again, since that object trips up `pdfLaTeX`. And yeah, `emo.py` automatically downloads the Noto color emoji sources if necessary.

4 Copyright and Licensing

Since `emo`’s distribution includes not only LaTeX code but also a substantial Python script, Unicode data about emoji, as well as graphics and fonts derived from Google’s Noto project, a number of different licenses apply. All of them are [OSI approved](#) and non-copyleft:

- This package’s LaTeX and also Perl code extracted from `emo.dtx` is © Copyright 2023 by Robert Grimm and has been released under the [LPPL v1.3c](#) or later.
- The `config/emo.py` script also is © Copyright 2023 by Robert Grimm but has been released under the [Apache 2.0 license](#).
- The `[config/emoji-test.txt]` configuration file is a data file from [Unicode TR-51](#) and hence subject to the [Unicode License](#).
- The `emo-lingchi.ttf` font is a two-glyph subset of the traditional Chinese version of Google’s [Noto serif](#) and hence subject to the [SIL Open Font License v1.1](#).
- The PDF graphics in the `emo-graphics` directory are derived from the sources for [Noto’s color emoji](#) and hence subject to the Apache 2.0 license.

5 Implementation

Let’s get started on `emo`’s implementation:

```
1 \<package>
```

Except, that implementation started near the top of this `emo.dtx` file, well before the documentation’s preamble. For completeness, here is the package declaration again:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{emo}
[2023/99/99 v1.0 emo•ji for all (LaTeX engines)]
```

Unfortunately, emo’s version number appears in triplicate in this file, which makes cutting a release a little more tricky than it should be. Alas, the above, faux package declaration is not one of those repeat offenders. Really.

5.1 Package Options

`\ifEmojiExtra` Define a conditional flag for each package option.

```
\ifemo@index
\ifemo@debug 2 \newif\ifEmojiExtra
              3 \newif\ifemo@index
              4 \newif\ifemo@debug
```

Wire each conditional to the package option and then process them all.

```
5 \DeclareOption{extra}{\EmojiExtratrue}
6 \DeclareOption{index}{\emo@indextrue}
7 \DeclareOption{debug}{\emo@debugtrue}
8 \ProcessOptions\relax
```

5.2 Package Dependencies and Backend Selection

Require `inputenc` to declare this file’s character encoding as UTF-8. XeTeX and LuaTeX already use that encoding by default and hence this line is redundant. But pdfTeX originates from darker, pre-Unicode times and thus needs to be told. Though thankfully, it does support the encoding by now.

```
9 \RequirePackage[utf8]{inputenc}
```

`\ifemo@use@unicode` Define a conditional flag for each major backend feature. They are *not* orthogonal: `\ifemo@use@unicode` and `\ifemo@use@pdf` are mutually exclusive and determine whether emo generates Unicode text or PDF graphics. For `\ifemo@use@font` to be enabled, `\ifemo@use@unicode` must already be enabled as well.

```
10 \newif\ifemo@use@unicode
11 \newif\ifemo@use@font
12 \newif\ifemo@use@pdf
```

Inspect the runtime environment and accordingly set the just defined backend flags. `\HCode` is defined by TeX4ht. Since correctly converting to HTML with TeX4ht requires LuaTeX, print a helpful error message for other engines.

```
13 \RequirePackage{iftex}
14 \ifdefined\HCode
15 \ifluatex\else
16 \PackageError{emo}{%
17   LuaTeX required for TeX4ht converting LaTeX to HTML.\MessageBreak
18   You may also see ``Missing character'' messages or \MessageBreak
19   ``Unicode character ... not set up'' errors.\MessageBreak
20   To fix, please pass the -l or --lua option to make4ht tool}{%
21   Run TeX4ht with LuaTeX by passing -l or --lua option to make4ht}
22 \fi
23 \emo@use@unicodetrue
```

```

24 \else
25 \ifluatex
26 \emo@use@unicodetrue
27 \emo@use@fonttrue
28 \else
29 \emo@use@pdftrue
30 \fi
31 \fi

```

We need the backend flags to decide whether to load fontspec or not. We do that before requiring any other packages because that way the packages related to encoding and fonts are loaded first (together with iftex).

```

32 \ifemo@use@font
33 \RequirePackage{fontspec}
34 \fi

```

Require xcolor for formatting highly visible error messages within the generated document. Always including another package that is only used when there are errors is not ideal. But I couldn't get on-demand package loading to work. So we have to eagerly require the package.

```

35 \RequirePackage{xcolor}

```

The two remaining requirements depends on a package option and on the backend, respectively.

```

36 \ifemo@index
37 \RequirePackage{index}
38 \fi
39 \ifemo@use@pdf
40 \RequirePackage{graphicx}
41 \fi

```

5.3 The Emoji Table

The emoji table contains an entry for every emoji supported in the current configuration. Each such entry defines a macro without arguments, which has the emoji's name prepended with `emo@emoji@` as its name and expands to the emoji's fully qualified Unicode codepoints. The macro for keycap-hash is an exception: Its first codepoint is the plain hash # character, which trips up TeX's parser. Hence the macro for keycap-hash expands to the `\char`'ed codepoints. If the extra option is active, the emoji table also contains entries for `\lingchi` and `\YHWH`.

Before version 1.0, `emo.def` contained the macro definitions for the emoji table, with definitions ordered by emojis' codepoints. The lack of meaningful hierarchy and order made `emo.def` unusable for generating documentation, notably an inventory of currently supported emoji. To make `emo.def` more useful, version 1.0 introduced a higher-level file format based on the five macros described next.

<code>\EmojiBeginGroup</code>	The file format for <code>emo.def</code> relies on <code>\DefineEmoji</code> to associate emoji names with their
<code>\EmojiBeginSubgroup</code>	Unicode codepoints, <code>\EmojiBeginGroup</code> and <code>\EmojiEndGroup</code> to organize emoji into
<code>\DefineEmoji</code>	
<code>\EmojiEndSubgroup</code>	
<code>\EmojiEndGroup</code>	

groups, and on `\EmojiBeginSubgroup` and `\EmojiEndSubgroup` to organize emoji into subgroups. All definitions with `\DefineEmoji` are listed in Unicode display order and grouped in their Unicode group and subgroup. Subgroups are properly nested inside groups. The group and subgroup for lingchi and YHWH are both called `extra`.

`\DefineEmoji{⟨name⟩}{⟨codepoints⟩}` takes an emoji’s name and its Unicode codepoints as arguments—except for keycap-hash, which uses has the `\char`’ed codepoints as value because TeX does not handle hash characters in the input.

`\EmojiBeginGroup{⟨group⟩}` and `\EmojiEndGroup{⟨group⟩}` take a group name as their only argument. In contrast, `\EmojiBeginSubgroup{⟨group⟩}{⟨subgroup⟩}` and `\EmojiEndSubgroup{⟨group⟩}{⟨subgroup⟩}` take both the group and subgroup names as their two arguments.

If the `extra` option is enabled, `\lingchi` and `\YHWH` are treated just like emoji, and their names and Unicode codepoints (sans any group, font selection, or text direction) are included with the emoji table.

The conversion from the abstract file format for `emo.def` to the concrete emoji table is simple enough, especially since four macros remain no-ops.

```
42 \def\EmojiBeginGroup#1{}
43 \def\EmojiBeginSubgroup#1#2{}
44 \def\DefineEmoji#1#2{%
45     \expandafter\def\csname emo@emoji@#1\endcsname{#2}}
46 \def\EmojiEndSubgroup#1#2{}
47 \def\EmojiEndGroup#1{}
```

With that, we are ready to read `emo.def`:

```
48 \input{emo.def}
```

The emoji table is automatically generated with the `config/emo.py` Python script. When invoked without arguments, that script simply recreates the emoji table based on the emoji present in the `emo-graphics` directory. If invoked with arguments naming Unicode emoji groups, groups and subgroups, or individual emoji, the script converts SVG graphics from the Noto emoji font sources to PDF graphics compatible with LaTeX and then rebuilds the inventory in `emo.def`. If necessary, the Python code downloads the Noto emoji font sources.

5.4 Internal Macros

`\emo@error@fg` Define two colors and a macro `\emo@error{⟨emoji-name⟩}` function that uses the two colors for formatting an attention-grabbing error message. If you use an invalid emoji name and overlook the warning in the console, you *will* notice the error message in the document thusly formatted.

```
49 \definecolor{emo@error@fg}{rgb}{1,1,1}
50 \definecolor{emo@error@bg}{rgb}{.6824,.0863,.0863}
51 \def\emo@error#1{%
52     \colorbox{emo@error@bg}{%
53         \textcolor{emo@error@fg}{%
54             \textsf{Bad} \texttt{\char`\emo\{#1\}}}}}
```

`\emo@ifdef` The `\emo@ifdef{⟨name⟩}{⟨code⟩}` macro validate the emoji name given as first argument. If the name is invalid, it expands to an error message. If the name is valid, it executes the code given as second argument. The implementation critically relies on the emoji table being accurate.

```

55 \def\emo@ifdef#1#2{%
56   \ifcsname emo@emoji@#1\endcsname#2\else%
57     \PackageWarning{emo}{Unknown emoji name in ``string\emo{#1}''}%
58     \emo@error{#1}%
59   \fi}

```

5.5 User Macros and Hooks

5.5.1 Background: Variability vs Code Duplication

Emo’s implementation changed considerably between versions 0.1 and 1.0:

For **versions 0.1 and 0.2**, I focused on getting emo to work across pdfTeX, XeTeX, and LuaTeX as well as to adhere to the (naming) conventions of the TeX ecosystem. My initial strategy for managing the variability of package options and backends was to define the same internal and user macros several times each to account for just that variability. Hence, emo included two versions for the internal `\emo@index` macro, namely one version to emit an index entry and one version to do nothing. It also included two versions each for `\emo`, `\lingchi`, and `\YHWH`, namely one version to emit a group with a command setting the font (and another command setting the text direction for `\YHWH`) followed by some Unicode codepoints and another version to include a PDF graphic. It worked. But especially the two definitions for each of the three user macros suffered from noticeable code duplication.

Version 0.2 also added support for conversion to HTML with LaTeXML. But it did so through a separate binding and hence did not require changes to emo’s implementation.

In contrast, for **version 0.3**, I integrated support for TeX4ht into the implementation and treated rendering Unicode codepoints *without* preceding font selection as a *third* backend. That increased variability and also the corresponding complexity of definitions. Notably, the internal `\emo@content` macro for rendering either Unicode codepoints, Unicode codepoints with font selection, or PDF graphic was defined thrice.

At the same time, the implementation also eliminated some variability by including entries for `\lingchi` and `\YHWH` in the emoji table. That made it possible for both macros to delegate to `\emo`—except when rendering Unicode codepoints *with font selection*. In the latter case, each macro renders glyphs in a different “language” and possibly writing system and hence also needs to select a different font.

For **version 0.4**, I introduced the debug package option, which instructs emo to make the bounding boxes for its output visible. To implement the option, I added code that wraps the result of a macro in an fbox three times. Clearly, that’s two times too many.

That same version also added unit tests and, as a result of writing and debugging the unit tests, my TeX-fu improved considerably. So after releasing that version, I started looking for a more ergonomic approach to emo’s implementation, discovered [LaTeX hooks](#), and switched to them for **version 1.0**.

5.5.2 A Skeleton of Render Hooks

emo/render/before	To determine emo's hooks, we consider the functional requirements of emo's three
emo/render/emoji	macros and pay close attention to how those requirements converge and diverge:
emo/render/chinese	1. All three macros visually render one or more glyphs. Let's introduce the
emo/render/hebrew	emo/render/content hook for that.
emo/render/content	2. Each macro generates glyphs in a different "language" and writing system and
emo/render/after	thus may need to set the font, text direction, and so on. Let's introduce the
	emo/render/emoji, emo/render/chinese, and emo/render/hebrew hooks for
	that.
	3. Just as in aspect-oriented programming, we sometimes need to track macro in-
	ocations (e.g., for the index package option) or modify macro results (e.g., for
	the debug package option). Let's introduce the emo/render/before and emo/-
	render/after hooks for that.

That more than suffices for declaring emo's render hooks:

```

60 \NewHook{emo/render/before}
61 \NewHook{emo/render/emoji}
62 \ifEmojiExtra
63 \NewHook{emo/render/chinese}
64 \NewHook{emo/render/hebrew}
65 \fi
66 \NewHook{emo/render/content}
67 \NewReversedHook{emo/render/after}

```

\emo@makecommand With the hooks declared, we are ready to define the internal \emo@makecommand macro, which we'll use to define the user-facing macros. Since those three macros differ in control sequence and font selection hook, \emo@makecommand{<command>}{<hook-call>} takes the command to be defined and the call to the appropriate font selection hook as its arguments. The resulting user macro takes one argument, the name of the entity to render. Since hooks have no arguments, we need a way for determining the currently expanding command, its key argument, and its rendered value. The implementation uses \emo@command, \emo@key, and \emo@value for just that purpose.

The rest is straight-forward: After defining \emo@command and \emo@key, the implementation tests whether that key is valid. If so, it invokes the emo/render/before hook, renders the content by invoking the font selection and content hooks inside a group, binds the result to \emo@value, invokes the emo/render/after hook, and yields the value. The group ensures that any sticky changes to font etc are contained within the macro invocation.

Admittedly, it took some trial and error to arrive at this version of \emo@makecommand. I started out with \emo@makecommand only assembling the body of each user macro. That required using \edef for the user macro and prefixing almost every token in \emo@makecommand with \noexpand. I converged on the current, simpler implementation once I realized that I could pass the control sequence to be defined as an argument.

```

68 \def\emo@makecommand#1#2{
69     \newcommand*{#1}[1]{%
70         \def\emo@command{#1}%

```

```

71      \def\emo@key{##1}%
72      \emo@ifdef{##1}{%
73          \UseHook{emo/render/before}%
74          \def\emo@value{%
75              \begingroup%
76              #2%
77              \UseHook{emo/render/content}%
78              \endgroup}%
79          \UseHook{emo/render/after}%
80          \emo@value}}

```

5.5.3 The User Macros

`\emo` The `\emo{<name>}` macro renders the named emoji either as Unicode or as a PDF graphic. To define it, all we need to do is instantiate the skeleton with the font selection hook for emoji.

```

81 \emo@makecommand\emo{\UseHook{emo/render/emoji}}

```

`\lingchi` Since `\lingchi` and `\YHWH` do not take arguments, they require a helper macro that provides the necessary argument to the macro generated by `\emo@makecommand`. Originally, both macros included a trailing `\xspace` in their expansion. While it would be possible to implement that with hooks based on `\emo@command`, it seems like an unnecessary complication, so I dropped that feature in version 1.0.

```

82 \ifEmojiExtra
83 \emo@makecommand\emo@lingchi{\UseHook{emo/render/chinese}}
84 \emo@makecommand\emo@YHWH{\UseHook{emo/render/hebrew}}
85 \newcommand*{\lingchi}{\emo@lingchi{lingchi}}
86 \newcommand*{\YHWH}{\emo@YHWH{YHWH}}
87 \fi

```

5.5.4 Activating the Hooks

So far, the three user-facing macros do nothing besides creating empty groups. We change that by activating the hooks as required by emoji's package options and the current engine. Conveniently, we only need to add to a hook if we need it to do something. In the following, we activate hooks in invocation order, from `emo/render/before` to font selection hooks to `emo/render/content` to `emo/render/after`.

If the `index` option is enabled, create the necessary index and emit entries through the `emo/render/before` hook. The hook uses the `\emo@emit@index` helper macro because stepping through `[emo]` requires three more `\expandafter` invocations.

```

88 \ifemo@index
89 \newindex{emo}{edx}{end}{Emoji Index}
90 \def\emo@emit@index#1{\index[emo]{#1}}
91 \AddToHook{emo/render/before}{%
92     \expandafter\emo@emit@index\expandafter{\emo@key}}
93 \fi

```

Next, if the backend uses fonts, set up font selection in the corresponding hooks. We only activate the `emo/render/chinese` and `emo/render/hebrew` hooks if the extra package option is enabled. Note that since Hebrew is written right-to-left, the font selection hook also set the text direction. This is safe to do because font selection and content are always enclosed in a group.

```

94 \ifemo@use@font
95 \newfontface\emo@font@emoji[Renderer=Harfbuzz]{NotoColorEmoji.ttf}
96 \AddToHook{emo/render/emoji}{\emo@font@emoji}
97 \ifEmojiExtra
98 \newfontface\emo@font@chinese{emo-lingchi.ttf}
99 \newfontface\emo@font@hebrew{LinLibertine_R.otf}
100 \AddToHook{emo/render/chinese}{\emo@font@chinese}
101 \AddToHook{emo/render/hebrew}{\emo@font@hebrew\textdir TRT}
102 \fi
103 \fi

```

Next, we activate the `emo/render/content` hook to either emit Unicode codepoints or PDF graphics.

```

104 \ifemo@use@unicode
105 \AddToHook{emo/render/content}{\csname emo@emoji@\emo@key\endcsname}
106 \fi
107 \ifemo@use@pdf
108 \AddToHook{emo/render/content}{%
109     \raisebox{-0.2ex}{%
110         \includegraphics[height=1em]{emo-graphics/emo-\emo@key}}}
111 \fi

```

Finally, if the debug package option is enabled, we wrap the rendered content in a frame box.

```

112 \ifemo@debug
113 \AddToHook{emo/render/after}{%
114     \let\emo@realvalue\emo@value%
115     \def\emo@value{\fbox{\emo@realvalue}}
116 \fi

```

Et voilà. That's it!

```

117 \endpackage

```

6 LaTeXML Binding

To support conversion from LaTeX to HTML, `emo` includes a so-called binding for [LaTeXML](#). It effectively is a (much simplified) re-implementation of `emo`'s core functionality, only written in Perl against LaTeXML's API. The binding ignores the `index` option and does not perform error checking on emoji names. If either is important to you, please compile the document with LaTeX first. Furthermore, the binding emits necessary Unicode codepoints only, without font annotations. If you want to specify fonts, please use a CSS fontstack.

Asking package authors to reimplement their packages for LaTeXML seems unreasonable to me. It leads to code duplication and places the maintenance burden on package authors. Yet, right after announcing emo, the question of LaTeXML support came up. LaTeXML includes the latexml.sty package, which defines \iflatexml. I would have used that command to make the three-line change to emo.sty necessary to support LaTeXML, except latexml.sty contains lots of other stuff that isn't needed. Always loading lots of macros only to detect LaTeXML slows down compilation and wastes memory. Since reimplementing \iflatexml would require a binding anyways, I just wrote a minimal binding. As I said, LaTeXML's approach is broken.

With that out of the way, let's get started:

```
1 \langle*latexml-binding\rangle
```

The binding starts with an explicit preamble because docstrip does not allow for a redefinition of the starting characters of a line comment. It is followed by the Perl dependencies.

```
2 ## emo's LaTeXML binding.
3 ## (C) 2023 by Robert Grimm.
4 ## Released under LPPL v1.3c or later.
5 use strict;
6 use warnings;
7 use LaTeXML::Package;
```

`\ifEmojiExtra` Next, we use raw TeX to declare the LaTeX package and the `\ifEmojiExtra` conditional. The binding does not require any other conditionals since it only runs under LaTeXML and does not support options other than extra.

```
8 RawTeX(<<'EOTeX');
9 \ProvidesPackage{emo}
10 [2023/99/99 v1.0 emo.ji for all (LaTeX engines)]
11 \newif\ifEmojiExtra
12 EOTeX
```

Option processing is almost trivial:

```
13 DeclareOption('extra', '\EmojiExtratrue');
14 DeclareOption('index', '');
15 DeclareOption('debug', '');
16 ProcessOptions();
```

`\EmojiBeginGroup` Define the five macros for parsing the emoji table. They are the same as those defined by the package itself.

```
\EmojiBeginSubgroup \DefineEmoji
\EmojiEndSubgroup 17 DefMacro('\EmojiBeginGroup{}', '');
\EmojiEndGroup 18 DefMacro('\EmojiBeginSubgroup{}{}', '');
19 DefMacro('\DefineEmoji{}{}',
20 '\expandafter\def\csname emo@emoji@#1\endcsname{#2}');
21 DefMacro('\EmojiEndSubgroup{}{}', '');
22 DefMacro('\EmojiEndGroup{}', '');
```

Thusly prepared, read in the emoji table.

```
23 InputDefinitions('emo', type => 'def', noltxml => 1);
```

`\emo` Define the simplest possible version of `\emo`. It has no hooks, no error checking, no font selection, and no support for index or debug. It does, however, expand the emoji table entry.

```
24 DefMacro('\emo{', '\csname emo@emoji@#1\endcsname');
```

`\lingchi` If the `emo@extra` conditional is enabled, provide similarly minimal re-definitions of the `\YHWH` `\lingchi` and `\YHWH` macros.

```
25 if (IfCondition(T_CS('ifEmojiExtra')) {
26   DefMacro('\lingchi', "\x{51cc}\x{9072}");
27   DefMacro('\YHWH', "\x{05D9}\x{05D4}\x{05D5}\x{05D4}");
28 }
```

That's it for the binding, too.

```
29 </latexml-binding>
```

7 Testing and Documenting Emo

As `emo`'s tagline so loudly proclaims, this package is intended to enable emoji across all major LaTeX engines. That requires testing across all major LaTeX engines. Macros to make testing and also documenting `emo` easier follow.

Alas, we first start an always false conditional to prevent execution of this code while generating the documentation.

```
30 < *scaffold>
31 \iffalse
32 </scaffold>
```

7.1 Emo's Support Package

```
1 < *support>
```

The `emo-support` package defines high-level macros that make testing and documenting the `emo` package easier. At the same time, no attempt has been made to make `emo-support` reusable. The package only exists for developing `emo`.

```
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesPackage{emo-support}[2023/99/99 v1.0 Test & document emo]
```

`emo-support` builds on standalone so that the resulting PDF has just the size of the output (plus some small margin). Its verbose option results in additional output from `\checkwidth` to help debug failing tests.

```
4 \newif\ifemo@verbose@canary
5 \DeclareOption{verbose}{\emo@verbose@canarytrue}
```

```
6 \ProcessOptions\relax
```

If standalone is the document class, configure it for capturing test output. The lua-visual-debug package may help debug regressions.

We require emo since this package is all about emo, iftex for determining the LaTeX engine, and xcolor for appearances sake. This class reuses several of emo's internal macros, including \ifemo@use@unicode, \ifemo@use@font, and \ifemo@use@pdf for backend selection as well as \emo@font@emoji, \emo@font@chinese, and \emo@font@hebrew for font selection.

```
7 \RequirePackage[extra]{emo}
8 \RequirePackage{iftex}
9 \RequirePackage{xcolor}
```

When requiring Libertinus and Inconsolata, relative order matters and does so depending on LaTeX engine.

```
10 \iftutex
11 \RequirePackage{fontspec}
12 \RequirePackage{libertinus}
13 \setmonofont{inconsolata}
14 \else
15 \RequirePackage{libertinus}
16 \RequirePackage{inconsolata}
17 \fi
```

\enginename I couldn't find an existing macro that provides the desired functionality, so we gotta round up the usual suspects...

```
18 \ifxetex
19 \def\enginename{XeTeX}
20 \else
21 \ifluatex
22 \def\enginename{LuaTeX}
23 \else
24 \ifpdfTEX
25 \def\enginename{pdfTeX}
26 \else
27 \def\enginename{\emph{unknown engine}}
28 \fi
29 \fi
30 \fi
```

emo@canary@frameinner Define frame and background colors for boundary boxes of sample text.

emo@canary@frameouter

```
emo@canary@background 31 \definecolor{emo@canary@frameinner}{HTML}{636366}
32 \definecolor{emo@canary@frameouter}{HTML}{48484A}
33 \definecolor{emo@canary@background}{HTML}{E5E5EA}
```

Adjust settings for \fcolorbox so that it serves as bounding box.


```

34 \setlength{\fboxrule}{0.5pt}
35 \setlength{\fboxsep}{0pt}

```

`\emo@nobox` The sample text may or may not show bounding boxes for words and emoji. It always shows the bounding box for the entire line.

`\emo@linebox`

```

36 \newcommand\emo@nobox[1]{#1}
37 \newcommand\emo@wordbox[1]{%
38   \fcolorbox{emo@canary@frameinner}{white}{#1}}
39 \newcommand\emo@linebox[1]{%
40   \fcolorbox{emo@canary@frameouter}{emo@canary@background}{#1}}

```

`\sampletext` Show a single line of text that makes use of `emo`'s three macros. To help identify incorrect font metrics, spurious whitespace, and other issues, show the line's bounding box and, for the starred version, the bounding boxes for words and emoji, too.

```

41 \def\@sampletext#1{%
42   \emo@linebox{%
43     #1{It's} #1{\lingchi}:
44     #1{Please}, #1{\YHWH}, #1{have} #1{mercy}
45     #1{\emo{pleading-face}}!}%
46   \vspace{1ex}}
47 \newcommand*\sampletext{%
48   \ifstar{\@sampletext\emo@wordbox}{\@sampletext\emo@nobox}}

```

`\emo@canary@actual` Validating `emo`'s macros turned out to be a bit trickier than I had expected. The obvious approach, fully expanding the macros and then comparing the result, doesn't work. TeX does support eager expansion via, for example, `\expandafter` and `\edef`. But it does so only for macros that expand to text but not boxes, graphics, and so on.

Instead, we need to take a sneakier approach: Generate a box with the macro invocation and another box with the expected result and then compare the widths of the two boxes. While that is an incomplete comparison and hence cannot detect all bugs, it *can* detect any bug where the macro's output is shorter or longer than expected. That conveniently includes whitespace, which is one of the bigger dangers for regressions.

We get started on that testing strategy by defining two box registers.

```

49 \newsavebox{\emo@canary@actual}
50 \newsavebox{\emo@canary@expected}

```

`\checkwidth` Before version 1.0, `\lingchi` and `\YHWH` relied on `xspace` to avoid any trailing backslashes. Now that all three user macros share the same code, implementing that feature would require another hook. So I dropped it.

Even though `\checkwidth` only tests three macros and two of them take no arguments, the test macro still needs five arguments to cover all variability:

1. name of macro being tested;
2. macro invocation being tested;
3. name of font variable used in LuaLaTeX's output;

4. Unicode code sequence in LuaLaTeX’s output;
5. file name for fallback PDF graphic without “emo-” prefix.

The third and fourth arguments are separate because the font variable only exists when running under LuaLaTeX.

```

51 \newcommand\checkwidth[5]{%
52   \sbox\emo@canary@actual{#2}%
53   \ifemo@use@font%
54     \sbox\emo@canary@expected{%
55       \begingroup\csname emo@font@#3\endcsname #4\endgroup}%
56   \else%
57   \ifemo@use@unicode%
58     \sbox\emo@canary@expected{\begingroup #4\endgroup}%
59   \else%
60     \sbox\emo@canary@expected{%
61       \raisebox{-0.2ex}{%
62         \includegraphics[height=1em]{emo-graphics/emo-#5}}}%
63   \fi%
64   \fi%
65   \ifemo@verbose@canary%
66     \fbox{\usebox{\emo@canary@actual}}\space%
67     \fbox{\usebox{\emo@canary@expected}}\space%
68   \fi%
69   \def\macroname{\texttt{\char`\'#1}}%
70   \ifdim\wd\emo@canary@actual=\wd\emo@canary@expected%
71     \mbox{\macroname{} \emo{check-mark-button}}%
72   \else%
73     \edef\emo@actual{\the\wd\emo@canary@actual}%
74     \edef\emo@expected{\the\wd\emo@canary@expected}%
75     \mbox{\macroname{} \emo{cross-mark} \emo@actual{} \emo@expected}%
76   \fi}

```

Use a small gap between paragraphs instead of indentation.

```

77 \setlength\parindent{0pt}
78 \setlength{\parskip}{1ex}

```

That’s it for emo’s support package.

```

79 </support>



```

7.2 Engine Test Runner and Report

```

1 <*canary>

```

Each test report identifies the LaTeX engine and then shows the results of the width tests for emo’s three user-visible macros. If a test passes, the output only contains the macro name and a  check mark. If a test fails, the output contains the macro name, the divergent box widths, and a  cross mark. In the latter case, the width test results spill into the next line (at the least).

```

2 \documentclass[border=10pt, varwidth=6in]{standalone}
3 \usepackage{emo-support}
4 \begin{document}
5 \Huge
6 \emo{keycap-hash} \enginename: {\Large Width of
7 \checkwidth{emo}{\emo{robot}}{emoji}{\char"1F916}{robot},
8 \checkwidth{lingchi}{\lingchi}{chinese}{\char"51CC\char"9072}{lingchi},
9 \checkwidth{YHWH}{\YHWH}{hebrew}{%
10 \csname textdir\endcsname TRT\char"5D9\char"5D4\char"5D5\char"5D4%
11 }{YHWH}}
12 \vspace{1ex}\par

```

Next is the sample text, first with and then without boundary boxes for words and emoji.

```

13 \sampletext*\par\sampletext
14 \end{document}

```

That's it for the tests and report.

```

15 \</canary>

```

7.3 Simple HTML Test Document

```

1 \*oneline

```

Not much to see here besides one line of content.s

```

2 \documentclass[border=10pt, varwidth=6in]{standalone}
3 \usepackage{emo-support}
4 \begin{document}
5 \Huge\sampletext
6 \end{document}

```

It's a wrap 🤖

```

7 \</oneline>

```

Change History

0.1	General: Make initial release	1	0.4	General: Automate testing across engines with canary.tex	1
0.2	General: Add LaTeXML binding for conversion to HTML	13		Introduce a simple unit testing framework	15
	Prefix font and graphic files with “emo-”	1		\ifemo@debug: Add debug option for drawing boundary boxes	7
	Support pdftex for extracting emo.dtx	1	1.0	General: Drop \textdir from emoji table entry for YHWH	8
0.3	General: Include \lingchi and \YHWH in emoji table	8		Introduce high-level emoji table format that preserves Unicode grouping and order	8
	Support TeX4ht for conversion to HTML	1		Use LaTeX hooks for handling engines and options	1
	\lingchi: Build on \emo by default . .	12		\lingchi: Drop use of trailing xspace .	12
	\YHWH: Build on \emo by default . . .	12		\YHWH: Drop use of trailing xspace . .	12

Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

Symbols	\documentclass	2, 2
\ifstar	E	
\@sampletext	\edef	73, 74
\\	\else 14, 15, 20, 23, 24, 26, 28, 56, 56, 59, 72	
\{	\emo	6, 7, <u>24</u> , 45, 57, 71, 75, <u>81</u>
\}	\emo/render/after	<u>60</u>
A	\emo/render/before	<u>60</u>
\AddToHook . . 91, 96, 100, 101, 105, 108, 113	\emo/render/chinese	<u>60</u>
B	\emo/render/content	<u>60</u>
\begin	\emo/render/emoji	<u>60</u>
\begingroup	\emo/render/hebrew	<u>60</u>
C	\emo@actual	73, 75
\char	\emo@canary@actual . . . 49, 52, 66, 70, 73	
\checkwidth	\emo@canary@background	<u>31</u>
\colorbox	\emo@canary@expected	<u>49</u> , 54, 58, 60, 67, 70, 74
\csname	\emo@canary@frameinner	<u>31</u>
D	\emo@canary@frameouter	<u>31</u>
\DeclareOption	\emo@command	<u>68</u>
\def 19, 20, 22, 25, 27, 41, 42, 43, 44, 45, 46, 47, 51, 55, 68, 69, 70, 71, 74, 90, 115	\emo@debugtrue	7
\definecolor	\emo@emit@index	90, 92
\DefineEmoji	\emo@error	<u>49</u> , 58
	\emo@error@bg	<u>49</u>
	\emo@error@fg	<u>49</u>

<code>\emo@expected</code>	74, 75	<code>\ifemo@verbose@canary</code>	4, 65
<code>\emo@font@chinese</code>	98, 100	<code>\ifEmojiExtra</code>	<u>2</u> , 8, 25, 62, 82, 97
<code>\emo@font@emoji</code>	95, 96	<code>\iffalse</code>	31
<code>\emo@font@hebrew</code>	99, 101	<code>\ifluatex</code>	15, 21, 25
<code>\emo@ifdef</code>	<u>55</u> , 72	<code>\ifpdxftex</code>	24
<code>\emo@indextrue</code>	6	<code>\iftutex</code>	10
<code>\emo@key</code>	<u>68</u> , 92, 105, 110	<code>\ifxetex</code>	18
<code>\emo@linebox</code>	<u>36</u> , 42	<code>\includegraphics</code>	62, 110
<code>\emo@lingchi</code>	83, 85	<code>\index</code>	90
<code>\emo@makecommand</code>	<u>68</u> , 81, 83, 84	<code>\input</code>	48
<code>\emo@nobox</code>	<u>36</u> , 48		
<code>\emo@realvalue</code>	114, 115		
<code>\emo@use@fonttrue</code>	27		
<code>\emo@use@pdftrue</code>	29		
<code>\emo@use@unicodetrue</code>	23, 26		
<code>\emo@value</code>	<u>68</u> , 114, 115		
<code>\emo@verbose@canarytrue</code>	5		
<code>\emo@wordbox</code>	<u>36</u> , 48		
<code>\emo@YHWH</code>	84, 86		
<code>\EmojiBeginGroup</code>	<u>17</u> , <u>42</u>		
<code>\EmojiBeginSubgroup</code>	<u>17</u> , 43		
<code>\EmojiBeginSubgroup</code>	<u>42</u>		
<code>\EmojiEndGroup</code>	<u>17</u> , <u>42</u>		
<code>\EmojiEndSubgroup</code>	<u>17</u> , <u>42</u>		
<code>\EmojiExtratrue</code>	5, 13		
<code>\emph</code>	27		
<code>\end</code>	6, 14		
<code>\endcsname</code>	10, 20, 24, 45, 55, 56, 105		
<code>\endgroup</code>	55, 58, 78		
<code>\enginename</code>	6, <u>18</u>		
<code>\expandafter</code>	20, 45, 92		
	F		L
<code>\fbox</code>	66, 67, 115	<code>\Large</code>	6
<code>\fboxrule</code>	34	<code>\let</code>	114
<code>\fboxsep</code>	35	<code>\lingchi</code>	8, <u>25</u> , 43, <u>82</u>
<code>\fcolorbox</code>	38, 40		
<code>\fi</code>	17, 22, 28, 29, 30, 30, 31, 34, 38, 41, 59, 63, 64, 65, 68, 76, 87, 93, 102, 103, 106, 111, 116		
	H		M
<code>\HCode</code>	14	<code>\macroname</code>	69, 71, 75
<code>\Huge</code>	5, 5	<code>\mbox</code>	71, 75
		<code>\MessageBreak</code>	17, 18, 19
	I		N
<code>\ifcsname</code>	56	<code>\NeedsTeXFormat</code>	2
<code>\ifdefined</code>	14	<code>\newcommand</code>	36, 37, 39, 47, 51, 69, 85, 86
<code>\ifdim</code>	70	<code>\newfontface</code>	95, 98, 99
<code>\ifemo@debug</code>	<u>2</u> , 112	<code>\NewHook</code>	60, 61, 63, 64, 66
<code>\ifemo@index</code>	<u>2</u> , 36, 88	<code>\newif</code>	2, 3, 4, 4, 10, 11, 11, 12
<code>\ifemo@use@font</code>	<u>10</u> , 32, 53, 94	<code>\newindex</code>	89
<code>\ifemo@use@pdf</code>	<u>10</u> , 39, 107	<code>\NewReversedHook</code>	67
<code>\ifemo@use@unicode</code>	<u>10</u> , 57, 104	<code>\newsavebox</code>	49, 50
			P
		<code>\PackageError</code>	16
		<code>\PackageWarning</code>	57
		<code>\par</code>	12, 13
		<code>\parindent</code>	77
		<code>\parskip</code>	78
		<code>\ProcessOptions</code>	6, 8
		<code>\ProvidesPackage</code>	3, 9
			R
		<code>\raisebox</code>	61, 109
		<code>\relax</code>	6, 8
		<code>\RequirePackage</code>	7, 8, 9, 9, 11, 12, 13, 15, 16, 33, 35, 37, 40
			S
		<code>\sampletext</code>	5, 13, <u>41</u>
		<code>\sbox</code>	52, 54, 58, 60
		<code>\setlength</code>	34, 35, 77, 78
		<code>\setmonofont</code>	13
		<code>\space</code>	66, 67
		<code>\string</code>	57

T		V	
<code>\textcolor</code>	53	<code>\vspace</code>	12, 46
<code>\textdir</code>	101	W	
<code>\textsf</code>	54	<code>\wd</code>	70, 73, 74
<code>\texttt</code>	54, 69	X	
<code>\the</code>	73, 74	<code>\x</code>	26, 27
U		Y	
<code>\usebox</code>	66, 67	<code>\YHWH</code>	9, <u>25</u> , 44, <u>82</u>
<code>\UseHook</code>	73, 77, 79, 81, 83, 84		
<code>\usepackage</code>	3, 3		