emo·ji for all
(LaTeX engines)

Robert Grimm

Version v0.4 (2023/04/26)

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

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`, `canary.tex`, and `README.md`. Extraction will overwrite existing files with the same name without asking.

2. Build the package documentation with change and symbol indices by running:

   ```
   $ source build.sh
   ```

   The shell script processes the tests in `canary.tex` with `pdflatex`, `xelatex`, and `lualatex`, combining test results into `canary.pdf`. It also processes `emo.dtx` with `pdflatex` and `makeindex` to produce documentation in `emo.pdf`.

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 יהוה.

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 Emoji Names

With some 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 emoji 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. 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. 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-americas`	`globe-americas`
`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`

2.2 Two Extra Macros

`\lingchi`
`\YHWH`

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` 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::place-geographic`.
- 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 with emo’s implementation:

1 ⟨*package⟩

Except, the package implementation started near the top of the `emo.dtx` file, before the documentation preamble. We repeat it here for completeness:

```
\NeedsTeXFormat{LaTeX2e}
```

```
\ProvidesPackage{emo}
    [2023/04/26 v0.4 emo•ji for all (LaTeX engines)]
```

And no, I didn't repeat the version number, date, or package information. Check `emo.dtx`.

5.1 Package Options

`\ifemo@extra`
`\ifemo@index`
`\ifemo@debug`

Emo's `extra` and `index` options are simple flags. So is the only incompletely documented `debug` option. We declare a new conditional for each and, if `\usepackage` includes an option, toggle the conditional's state.

```
\newif\ifemo@extra\emo@extrafalse
\DeclareOption{extra}{\emo@extratrue}
\newif\ifemo@indexing\emo@indexingfalse
\DeclareOption{index}{\emo@indexingtrue}
\newif\ifemo@debug\emo@debugfalse
\DeclareOption{debug}{\emo@debugtrue}
\ProcessOptions\relax
```

5.2 Setup Including Dependencies

The dependency on `inputenc` effectively declares this file's encoding to be UTF-8. The XeTeX and LuaTeX engines already expect files to be encoded that way and hence ignore the declaration. However, pdfTeX supports other (legacy) encodings and needs to be told.

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

`\emo@use@unicode`
`\emo@use@font`
`\emo@use@pdf`
`\emo@backend`

Emo currently supports three different backends for actually rendering emoji, namely the backend named `\emo@use@unicode` emits Unicode codepoints in a group, the one named `\emo@use@font` emits font selection before those same Unicode codepoints in the group, and the one named `\emo@use@pdf` emits PDF graphics. Once we know to name the backends, we can set `\emo@backend` to the currently active backend, determined by interrogating the runtime environment. Alas, we still need to implement the three backends; but `\emo@content` is defined closer to the end of the package implementation.

```
\def\emo@use@unicode{backend:unicode}
\def\emo@use@font{backend:font+unicode}
\def\emo@use@pdf{backend:pdf}
\RequirePackage{iftex}
\ifdefined\HCode
    \let\emo@backend=\emo@use@unicode
\else
\ifluatex
    \let\emo@backend=\emo@use@font
\else
    \let\emo@backend=\emo@use@pdf
\fi
\fi
```

With the backend selected, we can now require backend-specific packages, namely `fontspec` for selecting fonts in the `\emo@use@font` backend and `graphicx` for including PDF graphics in the `\emo@use@pdf` backend. The `\emo@use@unicode` backend has no similar requirements.

```
\ifx\emo@backend\emo@use@font
    \RequirePackage{fontspec}
\fi
\ifx\emo@backend\emo@use@pdf
    \RequirePackage{graphicx}
\fi
```

Next, emo requires `xcolor` for formatting highly visible error messages within the text. Always including another package that is only used when there are errors is not ideal. But when I tried calling `\RequirePackage` for `xcolor` from inside the error macro, it didn't work. Alternatively, I could make in-text errors optional.

```
\RequirePackage{xcolor}
```

Finally, emo's options also have dependencies, with `extra` requiring the `xspace` package and `index` requiring the `index` package:

```
\ifemo@extra
    \RequirePackage{xspace}
\fi
\ifemo@indexing
    \RequirePackage{index}
\fi
```

5.3 The Emoji Table

`\emo@emoji@name`

For each emoji with a PDF graphic in the `emo-graphics` directory and, if enabled, the two extra macros, the corresponding `\emo@emoji@`⟨*emoji-name*⟩ macro expands to its Unicode sequence. With over 3,000 distinct emoji in Unicode 15, emo relies on a Python script for populating the graphics directory and writing the table to the `emo.def` file. Since the package code does not change after installation but the emoji table may very well change, they are kept separate. Alternatively, we could use DocStrip to assemble the package file from three parts, the code from the previous sections, then the contents of the emoji table in `emo.def`, and then all subsequent code. Alas, that seems a bit much for turning two files into one.

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

5.4 Internal Macros

`\emo@error@fg`
`\emo@error@bg`
`\emo@error`

Define two colors and a 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 messsage in the document thusly formatted.

```
\definecolor{emo@error@fg}{rgb}{1,1,1}
\definecolor{emo@error@bg}{rgb}{.6824,.0863,.0863}
\def\emo@error#1{%
    \colorbox{emo@error@bg}{%
        \textcolor{emo@error@fg}{%
            \textsf{Bad} \texttt{\textbackslash emo\{#1\}}%
        }%
    }%
}
```

`\emo@ifdef` Validate the emoji name given as first argument. The macro expands to the second argument if the name is valid and an error message otherwise. Its implementation relies on the `emo@emoji` table.

```
\def\emo@ifdef#1#2{%
    \ifcsname emo@emoji@#1\endcsname#2\else%
        \PackageWarning{emo}{Unknown emoji name in ‘\string\emo{#1}’}%
        \emo@error{#1}%
    \fi%
}
```

`\emo@index` If indexing is enabled, record the use of an emoji. Otherwise, do nothing.

```
\ifemo@indexing
    \newindex{emo}{edx}{end}{Emoji Index}
    \def\emo@index#1{\index[emo]{#1}}
\else
    \def\emo@index#1{}
\fi
```

`\emo@content` Render the emoji content. This macro interfaces with the backend and thus needs to be defined as many times as there are backends: The Unicode backend just expands the entry from the emoji table. The font backend does the same, but sets the font to Noto Color Emoji first. The PDF backend instead includes the corresponding PDF graphic.

```
\ifx\emo@backend\emo@use@unicode
    \def\emo@content#1{%
        \begingroup\csname emo@emoji@#1\endcsname\endgroup%
    }
\else
\ifx\emo@backend\emo@use@font
    \newfontface\emo@font[Renderer=Harfbuzz]{NotoColorEmoji.ttf}
    \def\emo@content#1{%
        \begingroup\emo@font\csname emo@emoji@#1\endcsname\endgroup%
    }
\else
    \def\emo@content#1{%
        \raisebox{-0.2ex}{%
            \includegraphics[height=1em]{emo-graphics/emo-#1}}%
    }
\fi
\fi
```

In debug mode, emo draws a box around the content of `\emo`. That may help when tracking down spurious whitespace.

```
\ifemo@debug
    \let\emo@realcontent=\emo@content
    \def\emo@content#1{\fbox{\emo@realcontent{#1}}}
\fi
```

5.5 User Macros

`\emo` Thanks to carefully defined internal macros, the implementation of the main `\emo` macro is almost trivial. If the emoji name passes muster, emit an index entry and render the emoji content:

```
\newcommand\emo[1]{%
    \emo@ifdef{#1}{%
        \emo@index{#1}%
        \emo@content{#1}%
    }%
}
```

`\lingchi` `\YHWH` Since the emoji table in `emo.def` includes macros with the Unicode codepoints for "lingchi" and "YHWH," the implementation of `\lingchi` and `\YHWH` just delegates to `\emo`.

```
\ifemo@extra
\ifx\emo@backend\emo@use@font\else
    \newcommand\lingchi{\emo{lingchi}\xspace}
    \newcommand\YHWH{\emo{YHWH}\xspace}
\fi
```

Except, as hinted at by the backend test, delegating to `\emo` won't work when using fonts, since `\emo` uses Noto color emoji whereas the two extra macros do not. In that case, we define alternative versions that, similar to `\emo` rely on their own specialized `\emo@content` macros and also wrap them when the `debug` package option is enabled.

```
\ifx\emo@backend\emo@use@font
    \newfontface\emo@chinese{emo-lingchi.ttf}
    \newfontface\emo@hebrew{LinLibertine_R.otf}
   \def\emo@lingchi@content{\begingroup\emo@chinese\emo@emoji@lingchi\endgroup}
   \def\emo@YHWH@content{\begingroup\emo@hebrew\emo@emoji@YHWH\endgroup}

    \ifemo@debug
        \let\emo@lingchi@realcontent=\emo@lingchi@content
        \let\emo@YHWH@content=\emo@YHWH@content
        \def\emo@lingchi@content{\fbox{\emo@lingchi@realcontent}}
```

```
      \def\emo@YHWH@content{\fbox{\emo@YHWH@realcontent}}
   \fi

  \newcommand\lingchi{\emo@index{lingchi}\emo@lingchi@content\xspace}
   \newcommand\YHWH{\emo@index{YHWH}\emo@YHWH@content\xspace}
\fi
\fi
```

Et voilà. That's it!

```
⟨/package⟩
```

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:

```
⟨*latexml-binding⟩
```

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

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

`\ifemo@extra` Next, we use raw TeX to declare the LaTeX package and define the `emo@extra` conditional. There is no need to define the `emo@indexing` conditional, since it corresponds to the unsupported `index` option.

```
RawTeX(<<'EOTeX');
\ProvidesPackage{emo}
    [2023/04/26 v0.4 emo•ji for all (LaTeX engines)]
\newif\ifemo@extra\emo@extrafalse
EOTeX
```

Option prcessing is almost trivial:

```
DeclareOption('extra', '\emo@extratrue');
DeclareOption('index', '');
DeclareOption('debug', '');
ProcessOptions();
```

`\emo@emoji@name`
`\emo`

Just like the actual package implementation, the LaTeXML binding reads the emoji table from `emo.def`. Similar to the actual implementation of the `\emo` macro when running under LuaLaTeX, the binding expands the named entry from the emoji table, producing the emoji's Unicode codepoints.

```
InputDefinitions('emo', type => 'def', noltxml => 1);
DefMacro('\emo{}', '\csname emo@emoji@#1\endcsname');
```

`\lingchi`
`\YHWH`

If the `emo@extra` conditional is enabled, require the `xspace` package and then provide minimal re-definitions of the `\lingchi` and `\YHWH` macros. Both simply expand to the necessary Unicode codepoints.

```
if (IfCondition(T_CS('\ifemo@extra'))) {
    RequirePackage('xspace');
    DefMacro('\lingchi', "\x{51cc}\x{9072}\\xspace");
    DefMacro('\YHWH', "\x{05D9}\x{05D4}\x{05D5}\x{05D4}\\xspace");
}
```

That's it for the binding, too.

⟨/latexml-binding⟩

7 Emo's Test Document

As emo's tagline so loudly pronounces, the package is intended to enable emoji across all major LaTeX engines. This document provides the basis for testing that this is indeed the case. It results in a concise card, or "canary," that (1) identifies the engine, (2) lists the macro names `\emo`, `\lingchi`, and `\YHWH` adorned with ✅ or ❌ to indicate whether the macro produced output of the expected width, and (3) shows the same line of text using the three macros, with visible bounding boxes for line and words as well as line only.

7.1 Test Preamble

As usual, we start the test document by declaring its class and requiring necessary packages. Notably, `standalone` and `varwidth` help generate a PDF card that is sized to the test output. `iftex` is necessary for dynamically detecting the LaTeX engine, and `emo` is about to be tested. The rest are mostly indulgences for appearance. We also wrap the entire test document in an `\iffalse` `\fi` block so that it doesn't interfere with documentation generation.

```
⟨*scaffold⟩
\iffalse
⟨/scaffold⟩
⟨*canary⟩
\documentclass[border=10pt]{standalone}
\usepackage[extra]{emo}
\usepackage{iftex}
\usepackage{xcolor}
\usepackage{varwidth}

\iftutex
\usepackage{fontspec}
\usepackage{libertinus}
\setmonofont{inconsolata}
\else
\usepackage{libertinus}
\usepackage{inconsolata}
\fi
```

`\enginename` I couldn't find an existing macro that provides the engine name. Hence, we gotta round up the usual suspects:

```
\ifxetex
\def\enginename{XeTeX}
\else
\ifluatex
\def\enginename{LuaTeX}
\else
\ifpdftex
\def\enginename{pdfTeX}
\else
\def\enginename{unknown engine}
\fi
\fi
\fi
```

`\nobx` `\wbx` `\lbx` To display text with bounding boxes, we define some (very neutral) colors and boxes. In particular, `\nobx` and `\wbx`, respectively, do not or do draw a box around a word or emoji. `\lbx` draws a box around the line of text.

```
\definecolor{wordboxframe}{HTML}{636366}
\definecolor{lineboxframe}{HTML}{48484A}
```

```
\definecolor{lineboxbg}{HTML}{E5E5EA}

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

\newcommand\nobx[1]{#1}
\newcommand\wbx[1]{\fcolorbox{wordboxframe}{white}{#1}}
\newcommand\lbx[1]{\fcolorbox{lineboxframe}{lineboxbg}{#1}}
```

`\boundedtext` Drawing bounding boxes around "building blocks," i.e., emoji and words appearing in a line of text, can help identify buggy font metrics, spurious spaces, and so on. But it also is an awfully cluttered presentation. Hence we display the same line a second time, this time without the word-level boxes. That seems like a good use case for LaTeX's star-form of a command.

```
\makeatletter
\def\@boundedtext#1{%
    \lbx{%
        #1{It's} #1{\lingchi}:
        #1{Please}, #1{\YHWH}, #1{have} #1{mercy}
        #1{\emo{pleading-face}}!%
    }%
}
\newcommand*\boundedtext{%
    \@ifstar{\@boundedtext{\wbx}}{\@boundedtext{\nobx}}%
}
\makeatother
```

`\checkwidth` Validating the output of emo's macros turned out to be a bit trickier than I had expected. The obvious approach, expanding macros to primitives and then comparing to expected results, simply doesn't work—even though LaTeX does have builtin support for eager expansion via, for example, `\expandafter` and `\edef`. The reason is that unlike C, where the preprocessor runs strictly before the rest of the compiler, macro expansion in TeX and LaTeX is ubiquitous, delayed, repeated, interleaved, and so on.

Instead, we need to take a sneakier approach: Generate a box with the macro application and another box with the expected result and then compare the widths of the two boxes. While that cannot detect all bugs, it can detect one critical class of bugs: spurious whitespace! Alas, there is an additional complication: `\lingchi` and `\YHWH` use `xspace` to avoid ugly trailing backslash characters. But that also makes their results context-sensitive, which might become a problem in a unit test without context. It is for just that reason that the text arguments to `\sbox\actual` or `\sbox\expected` below end with a period.

Even though `\checkwidth` only tests three macros and two of them take no arguments, it still requires 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.

```
\newsavebox{\actual}
\newsavebox{\expected}

\newcommand\checkwidth[5]{%
    \sbox\actual{#2.}%
    \ifluatex%
        \sbox\expected{%
            \begingroup\csname#3\endcsname #4\endgroup.}%
    \else%
        \sbox\expected{%
            \raisebox{-0.2ex}{%
                \includegraphics[height=1em]{emo-graphics/emo-#5}}.}%
    \fi%
    \def\macroname{\texttt{\char‘\\#1}}%
    \ifdim\wd\actual=\wd\expected%
        \mbox{\macroname{} \emo{check-mark-button}}%
    \else%
        \edef\actualwidth{\the\wd\actual}%
        \edef\expectedwidth{\the\wd\expected}%
     \mbox{\macroname{} \emo{cross-mark} \actualwidth{} \expectedwidth}%
    \fi%
}
```

7.2 Test Body

All test macros have been defined and we can finally get to actually testing emo. We put all output in a `varwidth` environment to ensure that the PDF is content-sized. Since the output includes text with bounding boxes, we use a really large font size.

```
\begin{document}
\begin{varwidth}{6in}
\Huge

```

At the top of the card are the engine name and the width tests, one for each macro.

```
\noindent\enginename:
\Large Width of
\checkwidth{emo}{\emo{robot}}{emo@font}{\char”1F916}{robot},
\checkwidth{lingchi}{\lingchi}{emo@chinese}{\char”51CC\char”9072}{lingchi},
\checkwidth{YHWH}{\YHWH}{emo@hebrew}{\char”5D9\char”5D4\char”5D5\char”5D4}{YHWH}
```

```
\vspace{1ex}\Huge

```

At the bottom of the card are the two lines of text with bounding boxes, *with* word-level boxes and then without.

```
\boundedtext*{}
\vspace{1ex}\newline
\boundedtext{}

```

That's it as far as tests are concerned. Clearly, writing the test macros was more involved than using them. That effort also is the reason I decided to include the annotated test document in `emo.dtx`. All that's left at this point is wrapping up the test document, wrapping up the enclosing conditional, and wrapping up emo's documentation. It's a wrap 🤪

```
\end{varwidth}
\end{document}
⟨/canary⟩
⟨*scaffold⟩
\fi
⟨/scaffold⟩
```

Change History

0.1
- General: Make initial release 1

0.2
- General: Add LaTeXML binding for conversion to HTML 11
  - Prefix font and graphic files with "`emo-`" 1
  - Support `pdftex` for extracting `emo.dtx` 1

0.3
- General: Support TeX4ht for conversion to HTML 1
- `\emo@backend`: Make backend support scalable through symbolic names 7
- `\emo@emoji@name`: Include `\lingchi` and `\YWHW` as necessary 8
- `\lingchi`: Build on `\emo` by default 10
- `\YHWH`: Build on `\emo` by default . . 10

0.4
- General: Automate testing across engines with `canary.tex` 1
  - Introduce a simple unit testing framework 12
- `\emo@content`: Remove extra space from PDF renderer 9
- `\ifemo@debug`: Add `debug` option for drawing boundary boxes . . . 7

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.