`context-lmtx-mode` Package Guide

This package provides an enhanced environment for working with ConTeXt LMTX inside **Emacs**. It makes it easier and faster to write, edit, and compile ConTeXt documents directly from Emacs, without switching between external tools.

1 Package Structure

The package consists of **two main files**:

1.1 `myextraloaders.el`

This file contains additional configurations and quality-of-life improvements for Emacs, such as:

- Automatic closing of parentheses, quotes, and braces.
- Improved visual appearance for a more comfortable editing experience.
- Custom keyboard shortcuts for navigating quickly between windows.
- Minor performance and usability tweaks to make writing smoother.

This file is auxiliary, meaning it boosts productivity and comfort while working inside Emacs.

1.2 `context-lmtx-mode.el`

This is the core of the package. It provides features specifically designed for editing and compiling ConTeXt LMTX documents, including:

- Syntax highlighting for ConTeXt commands.
- Auto-completion for ConTeXt keywords and snippets.
- Dedicated keybindings for quick document compilation.
- Instant preview of generated output (e.g., PDF) without leaving Emacs.
- Management and coordination of related project files.

2 Purpose of the Package

The main goal of `context-lmtx-mode` is to help ConTeXt users:

1. Write faster with less boilerplate.
2. Avoid syntax errors with syntax-aware features.
3. Compile and preview output directly inside Emacs.

3 Folder Structure

```
context-lmtx-mode/

 context-lmtx-mode.el       # Main major mode file
 Optional/                  # Extra optional tools
    myextras-loader.el     # Isolated loader for optional modules
    template-tools.el      # Template management utilities
 README.md                  # Instruction file
```

4 How the Loading Works

The typical loading sequence is:

1. `myextraloaders.el` — Loads first to apply general editing enhancements and UI improvements.
2. `context-lmtx-mode.el` — Loads afterwards, enabling ConTeXt-specific features and compilation helpers.

Core Package: `context-lmtx-mode`

1 Installation and Setup

1.1 Installation

1. Copy the `context-lmtx-mode` folder into your personal Emacs packages directory:
   `~/.emacs.d/Packages/context-lmtx-mode/`

2. Ensure all dependencies are also present in the same directory:

   – `~/.emacs.d/Packages/polymode/`

   – `~/.emacs.d/Packages/company-mode/`

   – `~/.emacs.d/Packages/lua-mode/`

   – `~/.emacs.d/Packages/markdown-mode/`

3. In your `init.el`, add the following code:

```
;; Path to personal packages
(let ((base "~/.emacs.d/Packages/"))
  (dolist (path '("context-lmtx-mode"
                  "polymode"
                  "company-mode"
                  "lua-mode"
                  "markdown-mode"))
    (add-to-list 'load-path (expand-file-name path base))))

;; Load main module
(require 'context-lmtx-mode)

;; Load optional extras (without adding to global load-path)
(load (expand-file-name
       "context-lmtx-mode/Optional/myextras-loader.el"
       "~/.emacs.d/Packages/"))
```

1.2 Usage

- Open any `.ctx` file → `context-lmtx-mode` will start automatically.

- The mode enhances editing with:

  - Syntax highlighting
  - Code completion
  - Optional productivity tools

1.3 Template Management Shortcuts

- `C-c N` → Add or remove a template
- `C-c M` → Copy a template from the saved list into the current directory

2 About `context-lmtx-mode.el`

The file `context-lmtx-mode.el` is the **core module** of the `context-lmtx-mode` package for Emacs. It defines the *major mode* for editing ConTeXt LMTX documents, along with features such as automatic compilation, PDF viewing, error handling, environment code expansion, polymode integration, and auto-completion.
This module forms the foundation for all other optional components, providing essential editing, compilation, and integration functionality.

2.1 Purpose

The main goals of `context-lmtx-mode.el` are:

- Provide a dedicated *major mode* for ConTeXt LMTX documents with proper syntax highlighting.
- Offer a smooth workflow for *editing → compiling → viewing* documents.
- Deliver built-in *error handling* and clear compilation feedback.
- Enable *smart editing features* such as automatic `\start...\stop` environment insertion.
- Support *multi-language blocks* through Polymode integration.
- Integrate with *company-mode* for ConTeXt command auto-completion.

2.2 Key Features

- **Major Mode Definition:**
  - Inherits from `text-mode`.
  - Loads syntax highlighting rules from `context-tex-syntax-highlight.el`.
- **Compilation and Viewing:**
  - `C-c c` — Compile the current file using the ConTeXt `context` command.
  - `C-c v` — Open the compiled PDF with `evince` without creating extra buffers.

- **Error Handling:**
  - Detects compilation errors automatically.
  - Closes the compilation buffer if there are no errors.
  - If errors occur, shows a bottom side-window with the compilation log.
  - Includes a transient error mode to quickly close the window using `q` or `C-c C-q`.
- **Smart Environment Expansion:**
  - `C-c t` — Automatically insert the matching `\stopENV` for a given `\startENV`.
  - The `TAB` key expands environments when appropriate or performs normal indentation otherwise.
- **Polymode Integration:**
  - Allows embedding and editing of `Lua` code blocks using `\startluacode...\stopluacode`.
  - Supports Markdown blocks via `\startmarkdowncode...\stopmarkdowncode`.
  - Associates file extensions `.ctx`, `.mkiv`, and `.mkxl` with this mode.
- **Auto-Completion:**
  - Loads additional modules `context-lmtx-commands` and `context-lmtx-autocomplete`.
  - Provides *company-mode* completion for ConTeXt commands and environments.

2.3 Technical Notes

- Uses Emacs's built-in `compile` command to run ConTeXt.
- Detects errors in compilation output using regular expressions.
- Defines a *host mode* (`poly-context-hostmode`) and two *inner modes* (`poly-context-lua-innermo` and `poly-context-markdown-innermode`) for Polymode.
- Environment expansion feature leverages `thing-at-point` and regex matching to find and complete `\start...\stop` pairs.

3 About the `tools` Directory

The `tools` directory contains auxiliary scripts and generators that provide automatic support for `context-lmtx-mode`. Its purpose is to avoid manual maintenance of command lists by extracting metadata directly from official ConTeXt sources.

3.1 Purpose

The main goals of the `tools` directory are:

- Automate the generation of Emacs Lisp command data used by `context-lmtx-mode`.
- Keep the ConTeXt command list in sync with the official `context-en.xml` metadata file from the ConTeXt distribution.
- Provide accurate auto-completion and syntax-related data without manual duplication.

3.2 File `extract-context-meta.py`

`extract-context-meta.py` is a Python script that:

- Reads the official ConTeXt XML metadata file: `context-en.xml`.
- Extracts command definitions, parameters, and relevant info.
- Generates the Emacs Lisp file `context-lmtx-commands.el` automatically.
- Ensures that the list of known ConTeXt commands for the mode remains up-to-date with the ConTeXt core distribution.

This process reduces human error and eliminates the need for manual editing of large command tables inside Emacs Lisp code.

3.3 Relation to Auto-Completion

The generated file `context-lmtx-commands.el` is then used by the `context-lmtx-autocomplete.el` module, which:

- Integrates with *company-mode* in Emacs.
- Reads the list of available ConTeXt commands.

- Provides live suggestions and completions while editing.
- Updates automatically whenever a new `context-lmtx-commands.el` is generated by the Python tool.

3.4 Technical Notes

- The script `extract-context-meta.py` requires Python 3.
- It parses the XML structure of `context-en.xml` to extract commands in a structured way.
- The generated `context-lmtx-commands.el` contains a Lisp list of commands suitable for direct loading into Emacs.
- Any improvement to auto-completion only requires re-running the Python tool after updating `context-en.xml`.

myextra-loader Modules Documentation

4 About `myextras-loader.el`

The file `myextras-loader.el` is an **optional loader** for additional configurations used by user. Its main role is to load extra features located inside the 'Optional' folder **without** adding that folder to the global `load-path` in Emacs.

4.1 Purpose

The design of `myextras-loader.el` serves three main goals:

- **Prevent namespace collisions** by keeping optional modules isolated from the global `load-path`.
- **Ensure correct file loading** by explicitly locating and loading each optional module from the same directory.
- **Encourage modularity and maintainability** so that each optional feature can be enabled, disabled, or modified independently.

4.2 Usage

You can load this file directly in your `init.el` (or main Emacs configuration):

```
(load (expand-file-name
       "context-lmtx-mode/Optional/myextras-loader.el"
       "~/.emacs.d/Packages/"))
```

This ensures that:

- All optional modules are loaded from the correct local directory.
- The 'Optional' folder remains completely isolated from global Emacs paths.

4.3 How It Works

1. **Identify loader directory** — The variable `myextras-dir` stores the absolute path to the 'Optional' folder.
2. **List optional modules** — The variable `myextras-files` contains the filenames of all optional configuration files to be loaded (in the correct order).
3. **Load files safely** — A `dolist` loop loads each file only if it exists, preventing errors when files are missing.

4.4 Default Optional Modules

By default, the following files are included in `myextras-files`:

- `appearance.el` — UI and visual enhancements for Emacs.
- `keybindings.el` — Custom shortcuts for faster navigation and editing.
- `session-saving.el` — Save and restore your Emacs editing sessions.
- `template-tools.el` — Add, remove, and manage document templates.
- `miscellaneous.el` — Miscellaneous small features and tweaks.

5 About `appearance.el`

The file `appearance.el` is a lightweight configuration module for `context-lmtx-mode` that is responsible for simple *UI tweaks* and optional theme loading in Emacs. It focuses on adjusting common interface settings such as scroll bar width and tab spacing, and optionally loading a preferred color theme.

5.1 Purpose

The main goal of `appearance.el` is to:

- Provide basic interface customization without affecting functionality.
- Centralize all UI-related tweaks in one place for easy maintenance.
- Allow the user to optionally load a theme.

5.2 Configuration Details

5.2.1 Scroll Bar Width

The configuration sets:

```
(setq-default scroll-bar-width 7)
```

This adjusts the width of the scroll bar to **7 pixels** (for a slimmer and less intrusive appearance).

5.2.2 Tab Width

The configuration also sets:

```
(setq-default tab-width 4)
```

This defines the default tab character width in Emacs buffers as **4 spaces**, which is a common convention for code indentation.

5.3 Theme Configuration (Optional)

An optional example theme configuration is included, using `dracula-theme`:

```
;;(use-package dracula-theme
;;  :ensure t
;;  :config
;;  (load-theme 'dracula t))
```

- The lines are commented out by default.
- If uncommented, Emacs will install and activate the Dracula theme, a popular dark color scheme.
- Uses `use-package` for clean package management and configuration.

5.4 Integration

This file ends with:

```
(provide 'appearance)
```

This makes the module available for `require` in other configuration files, keeping the UI tweaks logically separated from functional modules.

6 About `keybindings.el`

The file `keybindings.el` is an **optional module** for `context-lmtx-mode` that defines a small set of custom keybindings. These bindings aim to improve editing efficiency by offering quick window navigation and simple insertion commands without requiring extra packages.

6.1 Purpose

The main goals of `keybindings.el` are:

- Provide **fast navigation between windows** using intuitive shortcuts.
- Allow quick insertion of common characters (e.g., quotation marks) without pressing awkward key combinations.

6.2 Defined Keybindings

- **Window navigation:**
  - `C-c b` — Move to the window **left** (`windmove-left`).
  - `C-c f` — Move to the window **right** (`windmove-right`).
  - `C-c p` — Move to the window **above** (`windmove-up`).
  - `C-c n` — Move to the window **below** (`windmove-down`).
- **Text insertion:**
  - `C-"` — Insert a double quotation mark (`"`) at point.

6.3 Technical Notes

- Uses Emacs built-in `windmove` library functions, no extra dependencies required.
- The `C-"` binding uses an inline anonymous `lambda` function to insert the character without needing a custom function definition.

7 About `template-tools.el`

The file `template-tools.el` is an optional module for `context-lmtx-mode` that provides a complete *template management system* for quickly storing, listing, removing, and copying reusable file or folder templates.
It allows users to:

- Save frequently used document structures or snippets as templates.
- Assign each template an auto-generated number for quick selection.
- Copy templates directly into the current working directory from Emacs.

7.1 Purpose

The main goals of `template-tools.el` are:

- Improve workflow efficiency by providing quick access to reusable structures.
- Make template management simple with a numbered quick-select system.
- Keep template storage local to the `Optional` configuration folder.

7.2 Storage System

1. Templates are stored in the `my/template-list` variable (a list of file/folder paths).
2. This list is automatically loaded from the file `Template-folder.el` located in the Optional folder (`myextras-dir`).
3. The list can be refreshed with `my/load-templates` and saved via `my/save-templates`.
4. The storage file is auto-generated; manual editing is discouraged.

7.3 Core Functions

1. **`my/add-template` — Prompts the user to select a folder or file as a template.**
   - **Avoids duplicates.**

  - **Sorts templates alphabetically.**
  - **Saves updated list automatically.**

2. **`my/remove-template` — Lists all templates with numbers, waits for preview, then removes the selected one.**
3. **`my/copy-template-here` — Lists templates with numbers, waits for 3 seconds, and copies the selected one to the directory of the current buffer.**
   - **Handles both files and directories.**
   - **Prevents overwriting by checking if the destination already exists.**
4. **`my/manage-templates` — Asks whether to add or remove a template and delegates to the relevant function.**

7.4 Numbered Quick-Select System

When listing templates:

- Each template is shown as:

  ```
  1) /path/to/template1
  2) /path/to/template2
  ...
  ```

- The user enters the number (not the path) to perform removal or copying.

This system avoids long directory navigation and enables fast template retrieval.

7.5 Keybindings

- `C-c N` — Add or remove a template (`my/manage-templates`).
- `C-c M` — Copy a template into the current directory (`my/copy-template-here`).

7.6 Technical Notes

- Templates are stored in a plain Emacs Lisp file (`Template-folder.el`) in the Optional folder.
- Uses standard Emacs functions for file and directory manipulation.
- `sit-for 3` is used for preview delay before taking user input.

8 About `session-saving.el`

The file `session-saving.el` is an **optional module** for `context-lmtx-mode` that provides **project-based session saving**. It uses Emacs' built-in `desktop` package to remember open buffers, window configuration, and other session data.
Unlike the default Emacs session saving (which is global), this module saves and loads sessions **per project**, where a project is identified by the presence of a `README.md` file in its root directory.

8.1 Purpose

The main goals of `session-saving.el` are:

- Enable **isolated sessions for each project** so that files from different projects do not mix.
- Allow the user to **resume work** exactly where they left off, including open source files, window layout, and buffer states.
- Integrate the save/load process into a simple keyboard shortcut.

8.2 How It Determines a Project

A project is recognized if the *current buffer's directory* contains a `README.md` file. Internally:

1. `my/in-directory-with-readme-p` — Checks if the current file is inside a folder containing `README.md`.
2. `my/project-dir-from-readme` — Returns the project's root directory if such a `README.md` exists.

8.3 How It Works

8.3.1 Saving a Session

Using `M-x my/save-session` or the shortcut `C-c S`:

- Detect the project folder.
- Create a hidden directory `.emacs-session/` in the project root.

- Use `desktop-save` to store the session data there.
- Show a confirmation message with the save location.

8.3.2 Loading a Session

Using `M-x my/load-session` or the shortcut `C-c L`:

- Detect the project folder.
- Look for an existing `.emacs-session/` directory.
- If found, switch to it with `desktop-change-dir` and restore the session.
- Show either a success or an error message.

8.4 Keybindings

- `C-c S` — Save the current project's session.
- `C-c L` — Load the current project's session.

These keybindings can be invoked from any buffer that is part of a recognized project.

8.5 Technical Notes

- Requires Emacs' built-in `desktop` library: `(require 'desktop)`.
- Session data is stored per project and does not interfere with the global `desktop` session.
- To change the project detection criterion, modify the `my/in-directory-with-readme-p` function.

9 About `miscellaneous.el`

The file `miscellaneous.el` is an optional module for Emacs that contains small, independent tweaks and micro-utilities which improve everyday editing workflow. It can be loaded by `myextras-loader.el` as part of the user's Optional configuration.
This module gathers configurations that do not require their own separate file, including custom deletion functions, editing enhancements, and a few behavior adjustments.

9.1 Purpose

- Provide handy word/backspace deletion utilities for more intuitive text editing.
- Enable minor editing enhancements that are useful in most coding and writing contexts.
- Adjust certain global Emacs behaviors for convenience.

9.2 Backspace and Word Deletion Functions

This section introduces functions to control how words and whitespace are removed.

1. `ryanmarcus/backward-kill-word` **Deletes backward intelligently:**
   - **If the cursor is preceded by whitespace or newlines, it removes them continuously.**
   - **Otherwise, deletes a single word backward, using `backward-kill-word`.**
2. **`delete-backward-word` Deletes *ARG words backward* without saving them to the kill ring.**
3. **`delete-forward-word` Deletes *ARG words forward* without saving them to the kill ring.**

9.3 Keybindings for Deletion

- `C-<backspace>` — Calls `delete-backward-word`.

- `C-<delete>` — Calls `delete-forward-word`.

These bindings make clean word deletion more efficient during text editing.

9.4 Editing Enhancements

- Enables `electric-pair-mode` **globally to automatically insert matching brackets and quotes.**
- **Sets `electric-pair-preserve-balance` to `t` to maintain syntactic balance when inserting or removing pairs.**

9.5 Miscellaneous Behavior Tweaks

- Disables confirmation before saving buffers when running `compile` by setting: `compilation-ask-about-save` to `nil`.