TECO-64 Internals

Overview

This document is a work in progress. It provides a description of the design of TECO-64 for the benefit of those who maintain or enhance it as well as those who are curious about how it is structured.

History

TECO (Text Editor and Corrector) has been a popular text editor on DEC (Digital Equipment Corporation) computers since the early 60's, and it remains in use today. TECO-64 is an open-source version written in C, intended to be portable to multiple operating environments. It was inspired by TECO C, created by Pete Siemsen, and developed by many others, most recently Tom Almy and Blake McBride. However, TECO-64 was written from scratch to take full advantage of modern features of the C language and run-time library.

Primary development was on Linux (Ubuntu), using *gcc* and other GNU tools, and it has been successfully run on Linux as well as Windows 10 and MacOS. It has also been successfully compiled on VMS, but requires pending run-time library support to be fully executable.

Development

The general guidelines used during development were:

- Familiarity – Use names that are either familiar to the reader (e.g., with a Linux-like directory structure), or which are otherwise descriptive of their purpose (e.g., *er_cmd.c* for a file which processes the ER command).
- Consistency – Use the same conventions in one place that are used in another (e.g., all functions which process specific commands having names of the form *exec_*()*).
- Readability – Use names that can be pronounced (as opposed to an alphabet soup of acronyms). And as much as possible, use names that include verbs (e.g., *reset_term()* or *clear_win()*).
- Simplicity – Avoid long functions. Also avoid large files containing numerous unrelated functions, and don't duplicate code.
- Modularity – Hide details of functions and modules as much as possible. This has made it possible to have more than one method for handling data in the text buffer, and more than one method for handling paging.

Testing & Debugging

- *assert()* statements have been extensively used to verify assumptions at run-time, especially regarding function parameters.
- *PC-lint* was used to verify correctness of code and detect possible bugs.
- *Valgrind* was used to detect memory leaks, as was some added debug code.
- *strace* was used to profile system calls.
- *uftrace* was used (in conjunction with *gprof)* to profile function and library calls.
- *gdb* was used for debugging.
- The use of *assert()* statements, *PC*-lint, *uftrace,* and *gdb* are facilitated by *makefile* options.

Documentation

- *Doxygen* comments were included to provide in-line documentation.
- XML and XSL files are used to create documentation as well as code defining TECO command-line options.

Directory Names

`./`	Contains the *makefile* used to build TECO, and the file used by Doxygen to generate program documentation.
`./bin`	Contains the TECO-64 executable, and also several Perl scripts used to build or test TECO code.
`./doc`	Contains TECO-64 documentation (other than what is generated by Doxygen).
`./etc`	Contains miscellaneous source files for testing TECO, as well as the XML and XSL files used to define command-line options.
`./html`	Contains documentation files generated by Doxygen.
`./include`	Contains .h header files used for building TECO, including one file, _options.h, that is generated during the build process.
`./lib`	Contains files used to test TECO.
`./obj`	Contains generated object files created during the build process.
`./src`	Contains the .c source files used to build TECO, and a configuration file used by PC-Lint.

File Names

`cmd_*.c`	Files that perform general processing of TECO commands.

`*_cmd.c` Files that perform a specific TECO command (e.g., `er_cmd.c` contains code for processing ER commands), or perform multiple related TECO commands (e.g., `yank_cmd.c` contains code for processing commands that yank data into memory).

`*buf.c` Files that provide an interface for inserting, deleting, reading, and writing text in the edit buffer. Only one of the following is used in any specific build.

`gap_buf.c` – Implements a gap buffer.

`rope_buf.c` – Implements a rope buffer (TBD).

`page_*.c` Files that provide an interface for paging forward (and possibly backward) through a file. Only one of the following is used in any specific build.

`page_file.c` – Writes pages to output file, uses a temporary file to allow for backwards paging (TBD).

`page_std.c` – Writes pages to output file; no backwards paging implemented (classic TECO paging method).

`page_vm.c` – Writes pages to output file only when file is closed; virtual memory is used to store pages, which allows for backwards paging (TBD).

`term_*.c` Files that process terminal input and output.

`*_sys.c` Files that provide interfaces to system-dependent features. Code in all other files should be system-independent, but this is subject to change during further testing.

Function Names

`cmd_*()` Functions that perform general command processing.

`exec_*()` Functions that execute individual commands.

`*_ebuf()` Functions that provide an interface to the edit (or text) buffer.

`*_tbuf()` Functions that manipulate the terminal buffer.

`*_win()` Functions that manipulate the editing window.