Using Schutz.

Rodney Bates

1. Quick Start

Start with the executable for Schutz on your path as `Lbe`. See Using Schutz For instructions on doing this. At the command line, `cd` into a directory where you wish to develop code in Modula-3, and type `Lbe`. Keep this command line window open.

Use `File->Open` to open a file named `*.[im][3g]` and containing Modula-3 code. Schutz will parse the code in the file and display it, with the layout regularized according to its language-dependent rules. For the moment, Assume it is already syntactically correct.

Schutz uses the file name suffix to ascertain what language the code in the file is in. So to create a new file, open a non-existent file with your desired file name. For now, use the `delete` key to erase the purple text.

Use the usual cursor-positioning and printable character keys to edit the code on the screen. Several common `emacs` key bindings are available, see below. New code you type will be displayed with an underline, meaning it has yet to be parsed. When you are ready, click the now-red `Unparsed` button, to reparse your edited code. If there are no syntax errors, the botton will turn green and say `Parsed`.

At any time, you can use `File->Save` to save the current code. This will write a file in Schutz's internal format, whose file name has an underscore appended to the name of the text file you opened. Later, you can reopen this file, which will leave you in the same state. You can use `File->Export` to write your code as pure text, which will use the original file name.

If there are syntax errors, the parse button will be yellow and say `Misparsed`. In this case, there will be syntax corrections displayed as token insertions, with blue insertion carets below, and token deletions, with blue strikeout through them. Do not panic. These are only suggestions. Except for possible layout changes, (which are reversible), Schutz has so far not changed your code. Any time you reparse, Schutz will treat the suggestions as not having been taken.

If you export the file (i.e., as text), the resulting file will also be as if the suggestions are not taken. This means any suggested to-be-deleted tokens will be present in the text file, but the suggested insertions will not, thus appearing different in this respect from what is on-screen, but containing the same actual code as Schutz has internally. If you later reopen this text file, the initial parse will recreate the same suggestions, thus reproducing the on-screen appearance.

You can accept a suggested correction by placing the cursor inside it and clicking `Analysis->Accept` `syntax repair`. This will leave the repair as if you had typed it that way. Alternatively, you can make any alterations inside a suggested repair just by typing as usual.

In most cases, if Schutz suffers an assertion failure, it will write a "checkpoint" file, named with two underscores appended to the text file name, and inform you of this with a dialog, which you can dismiss by clicking the `C` box in the corner. This file is almost the same as s Schutz internal file, with slightly more information, including a record of the command or operation that failed. You can later open it, which will reinstate your code to the state before the command or operation that failed. You would then have the option of clicking `Devel-> Replay failing command`, but if you are not debugging Schutz, you probably want to do something else.

Additionally, Schutz will pop a dialog describing the assertion failure, with a button labelled `Back out`. Clicking this will (usually), reset things to the state before the command or operation that failed. You can then try something different, possibly as simple as moving the cursor, to try to circumvent the failure. Or, you can save or close the file, to be later reopened.

2. Using Schutz

Schutz is a multi-language editor. As of 2021-02-01, the supported languages are Ldl0, Ldl1, and Modula-3. The first two are for defining languages to Schutz, and will be relevant only if you are doing development work on Schutz itself. Schutz identifies these by file name suffixes of `.ldl0` or `.ldl1`

File name suffixes of `*.[im][3g]` denote modula-3 as the language. There is currently no mechanism to check or create consistency between the file name and the name or kind of a Modula-3 compilaton unit, so be careful with file names. (This is yet to come.)

The altered-as-suggested code will be syntactically correct. It is what Schutz uses internally for doing layout and any semantic actions. Such actions are performed on the corrected form of code only. Currently, Modula-3 has no semantic actions and Ldl0 and Ldl1 semantic actions are done only by separate executables, in batch mode.

Parsing works on the uncorrected text. It is incremental, only reparsing where edits have changed things and nearby, as necessary.

The syntax repair algorithm, which generates suggested corrections, is quite sophisticated. It can include a number of backtracking actions, looking for a least cost total repair. It still has lots of room for tuning, including better costs. Unfortunately, sophisticated may not necessarily mean effective. You will sometimes be shocked at how far off the suggested repairs appear. But since Schutz is the only tool known to me that visibly displays its syntax repairs, it is probable that you have not had the opportunity to see the changes made by other syntax-repairing parsers. Perhaps they are no better.

You can select text by dragging through with the mouse. `Edit->Cut`, `Edit->Copy`, and `Edit->Paste` work as usual.

Schutz can parse and internally represent a syntactically incomplete program by using certain grammar nonterminals, referred to as *placeholders*. These are displayed as identifiers enclosed in French quotes. The syntax repair process may insert some of these, or you can type them manually, if you know their names and how to type the French quotes on your keyboard. Unlike other insertions, placeholders will be emitted in an exported text file. They will be recognized by the scanning and parsing done when such a file is later opened.

Schutz uses ISO 8859-1, also known as ISO-Latin-1, as the character set of text files. Opening and closing French quotes have code point values 16_AB and 16_BB in this set.

To see the list of possible placeholders for language `< lang>`, look in the Schutz distribution at the generated file `./< lang>/derived/< LangTok> .i3`. Look under the commented headings `(* Abstract Plus Nodes: *)`, `(*Abstract Star Nodes: *)`, and `(* Abstract Fixed Nodes: *)` These are the the AST nodes of the language, thus the placeholders, when enclosed in French quotes.

In the view of a compiler, a placeholder would be a syntactic (or, probably, lexical) error. However, in Schutz, it allows for a weak kind of syntactic correctness and thus meaningful internal parsed representation. Its spelling also will, hopefully, be a helpful suggestion as to what it should be replaced by, in order to reach syntactic completeness. Eventually, placeholders will play a more significant role in partial semantic analysis and maybe syntax-directed editing, when these functions are implemented.

3. Key bindings

Keys that are printable/displayable insert or overlay at the cursor position, depending on overwrite mode. Other keys have functions as follows:

- `Ctl/A` Move cursor to beginning of line.
- `Ctl/C` Cancel current operation.
- `Ctl/D` Delete character (forward).
- `Ctl/E` Move cursor to end of line.
- `Ctl/K` Delete to end of line.
- `Ctl/T` Transpose characters.
- `Backspace` Delete character backward.
- `Delete` Delete character (forward).
- `Linefeed` Insert new line.
- `Return` Insert new line.
- `Tab` Insert blanks to next tab stop.

- `Left` Move cursor left.
- `Right` Move cursor right.
- `Up` Move cursor up.
- `Down` Move cursor down.
- `Begin` Move cursor to beginning of line.
- `End` Move cursor to end of line.
- `Prior` Scroll up one window.
- `Next` Scroll down one window.
- `Home` To beginning of image.
- `Insert` Toggle insert mode.