.

Eredmel Specification

Kavi Gupta

July 23, 2015

Chapter 1

Motivation

The purpose of Eredmel is simple: a programming language for single-use scripts that will *never* be maintained by a group. Thus, ideas such as "readability," "there's only one way to do it"-type-standardization, and type safety are more or less irrelevant to its design.

Eredmel's primary design philosophy is customizability. While some languages like Java will not let you overload operators, and some languages like C++ still cling to the idea that functions need to be in the form `<name>(<args>...)`. In Eredmel, on the other hand, functions and operators are defined through the use of Enhanced Regular Expressions[1]. Additionally, the Eredmel preprocessor and interpreter are designed to be as lightweight as possible, containing very little logic and inbuilt functions. Almost all of the utility of Eredmel comes from basic language features written into the libraries. Programmers can easily modify these features and develop their own "flavors" of Eredmel, which may end up looking nothing like each other.

In other words, Eredmel is a language for an individual who wants to program their way, not someone else's way. As such, it is not recommended for use in large-scale programs with multiple editors.

[1] See the next chapter for more details

Chapter 2

Enhanced Regular Expressions

Enhanced Regular Expressions are basically regexes with two additions. First, matching parentheses can be recognized (with parentheses in quotes not counting). Second, any point in the regex can be asserted to be in or not in a quote.

They are completely reverse-compatible with regular expressions, except tildes need to be escaped as `\~`

2.1 Parenthesis Matching

The syntax of a parenthesis match consists of anchors which have the formats `~(` and `~)`, where `()` are replaced with any combination of parenthesis that are preloaded into the Eredmel engine.

These parentheses enforce matching. If a `~(` is matched by the Enregex matcher, then the enregex match will fail if one of these conditions does not match. (Where all counts refer to generic parentheses, here represented by `()`, that are not in quotes).

- The final match does not have a corresponding `~)`
- The text between the `~(` and the final match's corresponding `~)` contains an uneven number of `(`s and `)`s.
- The text between the `~(` (Point A) and the final match's corresponding `~)` (Point B) contains some Point C such that the number of `(`s between A and C is less than the number of `)`s between A and C.

These anchors are a simple part of the regex, so they can be used in alternations and repetitions

2.2 Quote Matching

The syntax of a quote match consists the formats `~'` and `~^'`, where `'` is replaced by an open– or close–quote symbol. The interpretation is as follows:

- If the quote symbol is an open quote, then it represents a positive match.
- If the quote symbol is a close quote that is not also an open quote, then it represents a negative match.
- A `^` reverses the sign of the match.

Whether something is "in a quote" is defined by the following algorithm.

- The beginning of the string is not in a quote.

- A start quote character begins a quote unless it is escaped and the Enregex Type declares that it can be escaped
- A close quote character *of the current quote type* ends a quote unless it is escaped and the Enregex Type declares that it can be escaped

A few examples are below, in which valid quote pairs are `''` and `!?`[1]

- `a~'` matches `'a'`
- `a~^'` does not match `'a'`
- `a~^'` matches `!a?`
- `a~^?` matches `!a?`

[1] This is a deliberately contrived example. These do not count as valid quotes in the Eredmel Default standard, which only recognizes `''`.

Chapter 3

The Eredmel Preprocessor

3.1 Preprocessor Directives

3.1.1 `include` Statements

```
include <path>
```

3.1.2 `replace` Statements

```
 replace\n\t<enregex>\n\t<replacement>
```

3.2 Preprocessor Steps

3.2.1 The Linker

The first step the preprocessor executes is to run the linker. The linker searches the document and expands `include` statements to the contents of the directory, relative to this location (this is platform specific, but `..` should generally refer to the parent directory, etc.)

Include statements expand to the empty string if their file has already been included in this one. This means that circular references are permissible because they don't actually get included an infinite number of times.

3.2.2 The Static Macro Expansion System

The static macro expansion system works as follows.