E Minor v1.0 Language Definition Manual

Official Specification — Version 1.0

Revision Date: 2025-08-13

1. Introduction

E Minor is a high-performance, natively executable, direct-hex opcode mapped programming language designed for:

- Raw Speed: Direct machine-executable output, zero translation layer.
- Optimized Safety: Capsule-based memory handling with Star Code validation.
- Ultra Machine-Friendly Syntax: Dual grammar—shortcode (machine-close) and long-form superlative (human-readable).
- Dual Compile Modes: Automatic switching between AOT and JIT compilation.

2. Language Goals

1. Zero-Ambiguity Execution – Every construct has a single deterministic opcode mapping.
2. Unified Parsing – Lexer, parser, and AST are one fused stage.
3. Native Optimization – Peephole, PGO, inlining, and constant folding built-in.
4. Portable Footprint – Minimal runtime, self-contained execution.
5. Readable + Writable – Long-form for clarity, shortcode for speed.

3. Grammar Definition (EBNF)

```
program = entry_block , { statement } ;
entry_block = "@" , ( "main" | "entry_point" ) , block ;
block = "{" , { statement } , "}" ;
statement = init_stmt | load_stmt | call_stmt | exit_stmt | assign_stmt | invoke_stmt | terminate_stmt | control_stmt ;
init_stmt = ( "#init" , capsule_id ) | ( "initialize" , "capsule" , capsule_id ) ;
load_stmt = ( "#load" , capsule_id , value ) | ( "assign" , "value" , value , "to" , "capsule" , capsule_id ) ;
call_stmt = ( "#call" , func_id , [ "," , capsule_id ] ) | ( "invoke" , "function" , func_id , "with" , capsule_id ) ;
exit_stmt = "#exit" | "terminate" , "execution" ;
assign_stmt = "assign" , value , "to" , capsule_id ;
invoke_stmt = "invoke" , "function" , func_id , [ "with" , capsule_id ] ;
terminate_stmt = "terminate" , "execution" ;
control_stmt = loop_stmt | branch_stmt ;
loop_stmt = "loop" , condition , block ;
branch_stmt = "if" , condition , block , [ "else" , block ] ;
capsule_id = "$" , identifier ;
func_id = "$" , identifier ;
identifier = letter , { letter | digit | "_" } ;
value = hex_literal | int_literal ;
condition = expr ;
```

```
expr = term , { ( "==" | "!=" | "<" | ">" | "<=" | ">=" ) , term } ;
term = identifier | value | capsule_id ;
hex_literal = "0x" , hex_digit , { hex_digit } ;
int_literal = digit , { digit } ;
letter = "A".."Z" | "a".."z" ;
digit = "0".."9" ;
hex_digit = digit | "A".."F" | "a".."f" ;
```

Superlative Keyword	**Shortcode**	**Hex Opcode**
initialize capsule <cap>	#init <cap>	0x01
assign value <val> to capsule <cap>	#load <cap>,<v>	0x02
invoke function <func> with capsule <cap>	#call <func>,<c>	0x03
terminate execution	#exit	0xFF
loop <cond> … end	loop	0x20
if <cond> … end	if	0x21
else … end	else	0x22
compare equal	==	0x30
compare not equal	!=	0x31
compare less than	<	0x32
compare greater than	>	0x33
compare less or equal	<=	0x34
compare greater or equal	>=	0x35