Dea/L0 API

Generated 2026-08-28
l0-v2.0.0 (a34372f)

Generated by Doxygen 1.9.8

1 Dea/L0 API Reference 1
1.1 About . . . 1
1.2 Overview . . . 1

2 Namespace Index 3
2.1 Namespace List . . . 3

3 Hierarchical Index 5
3.1 Class Hierarchy . . . 5

4 Class Index 11
4.1 Class List . . . 11

5 File Index 19
5.1 File List . . . 19

6 Namespace Documentation 21
6.1 check_trace_log Namespace Reference . . . 21
6.1.1 Function Documentation . . . 22
6.1.1.1 _build_arg_parser() . . . 22
6.1.1.2 _iter_events() . . . 22
6.1.1.3 _print_report() . . . 22
6.1.1.4 _validate_events() . . . 23
6.1.1.5 main() . . . 23
6.1.2 Variable Documentation . . . 24
6.1.2.1 KV_RE . . . 24
6.1.2.2 TRACE_RE . . . 24
6.2 l0_analysis Namespace Reference . . . 24
6.3 l0_ast Namespace Reference . . . 24
6.4 l0_ast_printer Namespace Reference . . . 26
6.4.1 Function Documentation . . . 27
6.4.1.1 _format_span() . . . 27
6.4.1.2 format_module() . . . 27
6.4.1.3 format_node() . . . 27
6.5 l0_backend Namespace Reference . . . 28
6.6 l0_c_emitter Namespace Reference . . . 28
6.7 l0_compilation Namespace Reference . . . 28
6.8 l0_context Namespace Reference . . . 28
6.9 l0_diagnostics Namespace Reference . . . 29
6.9.1 Function Documentation . . . 29
6.9.1.1 diag_from_node() . . . 29
6.9.1.2 diag_from_token() . . . 29
6.9.2 Variable Documentation . . . 30
6.9.2.1 DIAGNOSTIC_CODE_FAMILIES . . . 30

6.10 l0_driver Namespace Reference . . . 30
6.10.1 Function Documentation . . . 30
6.10.1.1 load_source_utf8() . . . 30
6.11 l0_expr_types Namespace Reference . . . 31
6.12 l0_internal_error Namespace Reference . . . 31
6.13 l0_lexer Namespace Reference . . . 31
6.13.1 Function Documentation . . . 32
6.13.1.1 _is_ascii_ident_part() . . . 32
6.13.1.2 _is_ascii_ident_start() . . . 32
6.13.1.3 _logical_predecessor_ends_expression() . . . 32
6.13.1.4 is_reserved_keyword() . . . 32
6.13.2 Variable Documentation . . . 33
6.13.2.1 _EXPR_ENDING_TOKENS . . . 33
6.13.2.2 HEX_CHARS . . . 33
6.13.2.3 KEYWORDS . . . 33
6.13.2.4 OCT_CHARS . . . 33
6.14 l0_locals Namespace Reference . . . 33
6.15 l0_logger Namespace Reference . . . 34
6.15.1 Function Documentation . . . 34
6.15.1.1 log() . . . 34
6.15.1.2 log_debug() . . . 34
6.15.1.3 log_error() . . . 35
6.15.1.4 log_info() . . . 35
6.15.1.5 log_stage() . . . 35
6.15.1.6 log_warning() . . . 35
6.16 l0_name_resolver Namespace Reference . . . 36
6.17 l0_parser Namespace Reference . . . 36
6.17.1 Function Documentation . . . 36
6.17.1.1 token_len() . . . 36
6.18 l0_paths Namespace Reference . . . 37
6.19 l0_resolve Namespace Reference . . . 37
6.19.1 Function Documentation . . . 37
6.19.1.1 _is_imported() . . . 37
6.19.1.2 resolve_symbol() . . . 37
6.19.1.3 resolve_type_ref() . . . 38
6.20 l0_scope_context Namespace Reference . . . 38
6.21 l0_signatures Namespace Reference . . . 39
6.22 l0_string_escape Namespace Reference . . . 39
6.22.1 Function Documentation . . . 39
6.22.1.1 decode_l0_string_token() . . . 39
6.22.1.2 encode_c_string_bytes() . . . 40
6.22.2 Variable Documentation . . . 40

6.22.2.1 _HEX_CHARS . . . 40
6.22.2.2 _OCT_CHARS . . . 40
6.22.2.3 _SIMPLE_ESCAPES . . . 41
6.22.2.4 _TRIGRAPH_THIRD_BYTES . . . 41
6.23 l0_symbols Namespace Reference . . . 41
6.24 l0_types Namespace Reference . . . 41
6.24.1 Function Documentation . . . 42
6.24.1.1 format_type() . . . 42
6.24.1.2 get_builtin_type() . . . 42
6.24.1.3 get_null_type() . . . 43
6.24.2 Variable Documentation . . . 43
6.24.2.1 _NULL_TYPE . . . 43
6.24.2.2 L0_PRIMITIVE_TYPES . . . 43
6.25 l0c Namespace Reference . . . 43
6.25.1 Function Documentation . . . 45
6.25.1.1 _add_all_modules_arg() . . . 45
6.25.1.2 _add_codegen_arg() . . . 46
6.25.1.3 _add_runtime_args() . . . 46
6.25.1.4 _add_target_args() . . . 46
6.25.1.5 _check_entry_main_for_build() . . . 46
6.25.1.6 _compile_generated_c() . . . 47
6.25.1.7 _compiler_flag_family() . . . 47
6.25.1.8 _default_executable_name() . . . 48
6.25.1.9 _dump_tokens_for_file() . . . 48
6.25.1.10 _emit_diagnostic() . . . 48
6.25.1.11 _emit_temporary_source_cleanup_failure() . . . 48
6.25.1.12 _emit_verbose_compiler_identity() . . . 49
6.25.1.13 _find_cc() . . . 49
6.25.1.14 _format_token_dump_text() . . . 49
6.25.1.15 _get_module_names() . . . 49
6.25.1.16 _get_optimize_flag() . . . 50
6.25.1.17 _init_env_defaults() . . . 50
6.25.1.18 _is_valid_module_name() . . . 50
6.25.1.19 _is_windows_host() . . . 50
6.25.1.20 _load_file_lines() . . . 51
6.25.1.21 _normalize_cli_argv() . . . 51
6.25.1.22 _output_flags() . . . 51
6.25.1.23 _remove_temporary_c_source() . . . 51
6.25.1.24 _run_analysis() . . . 52
6.25.1.25 _runtime_include_flags() . . . 52
6.25.1.26 _runtime_library_flags() . . . 52
6.25.1.27 _scan_cli_presentation_options() . . . 52

6.25.1.28 _split_c_options() . . . 53
6.25.1.29 _split_cli_and_program_args() . . . 53
6.25.1.30 _validate_mode_scoped_flags() . . . 53
6.25.1.31 _validate_reserved_canonical_flags() . . . 54
6.25.1.32 _validate_runtime_library_path() . . . 54
6.25.1.33 _validated_temporary_directory() . . . 54
6.25.1.34 _write_temporary_c_source() . . . 55
6.25.1.35 build_compilation_context() . . . 55
6.25.1.36 build_search_paths() . . . 55
6.25.1.37 cmd_ast() . . . 56
6.25.1.38 cmd_build() . . . 56
6.25.1.39 cmd_check() . . . 56
6.25.1.40 cmd_codegen() . . . 57
6.25.1.41 cmd_compile() . . . 57
6.25.1.42 cmd_run() . . . 57
6.25.1.43 cmd_sym() . . . 58
6.25.1.44 cmd_tok() . . . 58
6.25.1.45 cmd_type() . . . 58
6.25.1.46 compiler_identity_text() . . . 59
6.25.1.47 main() . . . 59
6.25.1.48 print_diagnostic_list() . . . 59
6.25.1.49 print_diagnostic_with_snippet() . . . 60
6.25.1.50 print_diagnostics() . . . 60
6.25.2 Variable Documentation . . . 60
6.25.2.1 _CLI_CANONICAL_SHORT_VALUE_OPTIONS . . . 60
6.25.2.2 _CLI_LONG_VALUE_OPTIONS . . . 60
6.25.2.3 _CLI_NAMESPACED_SHORT_VALUE_OPTIONS . . . 61
6.25.2.4 _CLI_OTHER_SHORT_VALUE_OPTIONS . . . 61
6.25.2.5 _CLI_VALUE_OPTIONS . . . 61

7 Class Documentation **63**

7.1 _l0_base_opt Struct Reference . . . 63
7.1.1 Detailed Description . . . 63
7.1.2 Member Data Documentation . . . 63
7.1.2.1 has_value . . . 63
7.2 _l0_h_string Struct Reference . . . 63
7.2.1 Detailed Description . . . 64
7.2.2 Member Data Documentation . . . 64
7.2.2.1 bytes . . . 64
7.2.2.2 len . . . 64
7.2.2.3 refcount . . . 64
7.3 check_trace_log._MessageCollector Class Reference . . . 64

7.3.1 Detailed Description . . . 65
7.3.2 Constructor & Destructor Documentation . . . 65
7.3.2.1 __init__() . . . 65
7.3.3 Member Function Documentation . . . 65
7.3.3.1 append() . . . 65
7.3.4 Member Data Documentation . . . 65
7.3.4.1 limit . . . 65
7.3.4.2 messages . . . 65
7.3.4.3 total . . . 66
7.4 l0_parser._ParseSyncException Class Reference . . . 66
7.4.1 Detailed Description . . . 66
7.5 _rt_alloc_record Struct Reference . . . 67
7.5.1 Detailed Description . . . 67
7.5.2 Member Data Documentation . . . 67
7.5.2.1 base . . . 67
7.5.2.2 cold_index . . . 67
7.5.2.3 generation . . . 68
7.5.2.4 hot_pad . . . 68
7.5.2.5 mem_kind . . . 68
7.5.2.6 q_next . . . 68
7.5.2.7 read_only . . . 68
7.5.2.8 size . . . 68
7.5.2.9 state . . . 68
7.5.2.10 tree_left . . . 68
7.5.2.11 tree_prio . . . 68
7.5.2.12 tree_right . . . 68
7.6 _rt_alloc_record_cold Struct Reference . . . 69
7.6.1 Detailed Description . . . 69
7.6.2 Member Data Documentation . . . 69
7.6.2.1 align . . . 69
7.6.2.2 alloc_file . . . 69
7.6.2.3 alloc_line . . . 69
7.6.2.4 drop_file . . . 69
7.6.2.5 drop_line . . . 69
7.6.2.6 reserved . . . 70
7.6.2.7 type_id . . . 70
7.7 _rt_ptr_site Struct Reference . . . 70
7.7.1 Detailed Description . . . 70
7.7.2 Member Data Documentation . . . 71
7.7.2.1 generation . . . 71
7.7.2.2 owner . . . 71
7.8 l0c._TemporarySourceWriteError Class Reference . . . 71

7.8.1 Detailed Description . . . 72
7.8.2 Constructor & Destructor Documentation . . . 72
7.8.2.1 __init__() . . . 72
7.8.3 Member Data Documentation . . . 72
7.8.3.1 retained_path . . . 72
7.9 AnalysisResult Struct Reference . . . 72
7.9.1 Detailed Description . . . 73
7.9.2 Member Data Documentation . . . 73
7.9.2.1 diags . . . 73
7.9.2.2 driver_state . . . 73
7.9.2.3 expr_types . . . 73
7.9.2.4 intrinsic_targets . . . 74
7.9.2.5 locals_result . . . 74
7.9.2.6 names . . . 74
7.9.2.7 signatures . . . 74
7.9.2.8 var_ref_resolution . . . 74
7.10 l0_analysis.AnalysisResult Class Reference . . . 74
7.10.1 Detailed Description . . . 75
7.10.2 Member Function Documentation . . . 75
7.10.2.1 has_arc_data() . . . 75
7.10.2.2 has_errors() . . . 76
7.10.2.3 has_warnings() . . . 76
7.10.2.4 is_arc_type() . . . 76
7.10.3 Member Data Documentation . . . 77
7.10.3.1 context . . . 77
7.10.3.2 cu . . . 77
7.10.3.3 diagnostics . . . 77
7.10.3.4 enum_infos . . . 77
7.10.3.5 expr_types . . . 77
7.10.3.6 func_envs . . . 77
7.10.3.7 func_types . . . 77
7.10.3.8 intrinsic_targets . . . 77
7.10.3.9 let_types . . . 77
7.10.3.10 module_envs . . . 77
7.10.3.11 struct_infos . . . 78
7.10.3.12 var_ref_resolution . . . 78
7.11 ArrayBase Struct Reference . . . 78
7.11.1 Detailed Description . . . 78
7.11.2 Member Data Documentation . . . 78
7.11.2.1 capacity . . . 78
7.11.2.2 data . . . 78
7.11.2.3 element_size . . . 79

7.12 l0_ast.AssignStmt Class Reference . . . 79
7.12.1 Detailed Description . . . 80
7.12.2 Member Data Documentation . . . 80
7.12.2.1 target . . . 80
7.12.2.2 value . . . 81
7.13 Backend Struct Reference . . . 81
7.13.1 Detailed Description . . . 82
7.13.2 Member Data Documentation . . . 82
7.13.2.1 cfg . . . 82
7.13.2.2 current_env . . . 82
7.13.2.3 current_func_result . . . 82
7.13.2.4 current_loop_cleanup_scopes . . . 82
7.13.2.5 current_loop_labels . . . 82
7.13.2.6 current_module . . . 82
7.13.2.7 current_scope . . . 82
7.13.2.8 current_unit . . . 82
7.13.2.9 emitter . . . 82
7.13.2.10 label_counter . . . 83
7.13.2.11 next_stmt_unreachable . . . 83
7.13.2.12 opts . . . 83
7.13.2.13 result . . . 83
7.13.2.14 scope_counter . . . 83
7.13.2.15 scope_stack . . . 83
7.14 l0_backend.Backend Class Reference . . . 83
7.14.1 Detailed Description . . . 88
7.14.2 Member Function Documentation . . . 89
7.14.2.1 __post_init__() . . . 89
7.14.2.2 _alignof_expr_for_type() . . . 89
7.14.2.3 _build_type_dependency_graph() . . . 89
7.14.2.4 _collect_reassigned_arc_params() . . . 89
7.14.2.5 _convert_expr_with_expected_type() . . . 90
7.14.2.6 _emit_bare_variant_static_initializer() . . . 90
7.14.2.7 _emit_binary_op() . . . 90
7.14.2.8 _emit_block() . . . 91
7.14.2.9 _emit_block_sequence() . . . 91
7.14.2.10 _emit_case() . . . 91
7.14.2.11 _emit_case_literal() . . . 92
7.14.2.12 _emit_checked_pointer_expr() . . . 92
7.14.2.13 _emit_cleanup_at_scope_exit() . . . 92
7.14.2.14 _emit_cleanup_for_loop_exit() . . . 93
7.14.2.15 _emit_cleanup_for_return() . . . 93
7.14.2.16 _emit_condition_branch() . . . 93

7.14.2.17 _emit_condition_expr() . . . 94
7.14.2.18 _emit_condition_value() . . . 94
7.14.2.19 _emit_const_constructor() . . . 94
7.14.2.20 _emit_const_struct_constructor() . . . 95
7.14.2.21 _emit_const_variant_constructor() . . . 95
7.14.2.22 _emit_copy_expr_with_retains() . . . 96
7.14.2.23 _emit_drop() . . . 96
7.14.2.24 _emit_enum_cleanup() . . . 96
7.14.2.25 _emit_expr() . . . 97
7.14.2.26 _emit_expr_with_expected_type() . . . 97
7.14.2.27 _emit_for() . . . 98
7.14.2.28 _emit_function_declaration() . . . 98
7.14.2.29 _emit_function_declarations() . . . 98
7.14.2.30 _emit_function_definition() . . . 99
7.14.2.31 _emit_function_definitions() . . . 99
7.14.2.32 _emit_if_else() . . . 99
7.14.2.33 _emit_inline_with_header_item() . . . 99
7.14.2.34 _emit_let() . . . 100
7.14.2.35 _emit_let_declaration() . . . 100
7.14.2.36 _emit_let_declarations() . . . 100
7.14.2.37 _emit_let_initializer() . . . 100
7.14.2.38 _emit_line_directive() . . . 101
7.14.2.39 _emit_lvalue_with_caching() . . . 101
7.14.2.40 _emit_main_wrapper_if_needed() . . . 102
7.14.2.41 _emit_match() . . . 102
7.14.2.42 _emit_new_expr() . . . 102
7.14.2.43 _emit_ord_intrinsic() . . . 103
7.14.2.44 _emit_owned_expr_with_expected_type() . . . 103
7.14.2.45 _emit_pattern_bindings() . . . 103
7.14.2.46 _emit_pointer_index_lvalue() . . . 104
7.14.2.47 _emit_reassignment() . . . 104
7.14.2.48 _emit_retain_for_copied_value() . . . 104
7.14.2.49 _emit_return() . . . 105
7.14.2.50 _emit_sizeof_intrinsic() . . . 105
7.14.2.51 _emit_stmt() . . . 106
7.14.2.52 _emit_struct_cleanup() . . . 106
7.14.2.53 _emit_struct_constructor() . . . 106
7.14.2.54 _emit_unwrap() . . . 107
7.14.2.55 _emit_value_cleanup() . . . 107
7.14.2.56 _emit_variant_constructor() . . . 108
7.14.2.57 _emit_while() . . . 108
7.14.2.58 _emit_with() . . . 108

7.14.2.59 _emit_with_cleanup_from_scope() . . . 109
7.14.2.60 _emit_with_cleanup_header_let_assign() . . . 109
7.14.2.61 _emit_with_cleanup_header_let_predecl() . . . 109
7.14.2.62 _expect_expr_type() . . . 110
7.14.2.63 _extract_value_type_dependencies() . . . 110
7.14.2.64 _find_cycle_details() . . . 111
7.14.2.65 _find_enum_decl() . . . 111
7.14.2.66 _find_struct_decl() . . . 112
7.14.2.67 _fresh_label() . . . 112
7.14.2.68 _gen_if_else_branch() . . . 112
7.14.2.69 _has_side_effects() . . . 113
7.14.2.70 _int_type_size() . . . 113
7.14.2.71 _is_binary_op_enabled() . . . 114
7.14.2.72 _is_extern_function() . . . 114
7.14.2.73 _is_int_assignable() . . . 114
7.14.2.74 _is_place_expr() . . . 115
7.14.2.75 _is_unwrap_cast_from_place() . . . 115
7.14.2.76 _iter_body_stmts() . . . 115
7.14.2.77 _lookup_local_var_type() . . . 116
7.14.2.78 _lookup_owned_local_name() . . . 116
7.14.2.79 _lookup_symbol() . . . 116
7.14.2.80 _materialize_arc_temp() . . . 117
7.14.2.81 _needs_arc_temp() . . . 117
7.14.2.82 _pointer_type_or_none() . . . 118
7.14.2.83 _pop_scope() . . . 118
7.14.2.84 _push_scope() . . . 118
7.14.2.85 _register_inline_with_cleanup() . . . 118
7.14.2.86 _resolve_let_type() . . . 118
7.14.2.87 _resolve_type_ref() . . . 119
7.14.2.88 _scope_chain_has_cleanup() . . . 119
7.14.2.89 _should_materialize_arc_temp() . . . 119
7.14.2.90 _sizeof_expr_for_type() . . . 120
7.14.2.91 _topological_sort() . . . 120
7.14.2.92 _try_emit_constructor() . . . 120
7.14.2.93 _try_emit_intrinsic() . . . 121
7.14.2.94 _types_equal() . . . 121
7.14.2.95 find_variant_decl() . . . 121
7.14.2.96 generate() . . . 122
7.14.2.97 ice() . . . 122
7.14.3 Member Data Documentation . . . 123
7.14.3.1 _current_func_result [1/2] . . . 123
7.14.3.2 _current_func_result [2/2] . . . 123

7.14.3.3 _current_scope [1/2] . . . 123
7.14.3.4 _current_scope [2/2] . . . 123
7.14.3.5 _emit_let_initializer . . . 123
7.14.3.6 _label_counter . . . 123
7.14.3.7 _loop_cleanup_scope_stack . . . 123
7.14.3.8 _loop_label_stack . . . 123
7.14.3.9 _next_stmt_unreachable [1/2] . . . 124
7.14.3.10 _next_stmt_unreachable [2/2] . . . 124
7.14.3.11 _switch_depth . . . 124
7.14.3.12 analysis [1/2] . . . 124
7.14.3.13 analysis [2/2] . . . 124
7.14.3.14 current_module [1/2] . . . 124
7.14.3.15 current_module [2/2] . . . 124
7.14.3.16 emitter . . . 124
7.15 l0_ast.BinaryOp Class Reference . . . 125
7.15.1 Detailed Description . . . 126
7.15.2 Member Data Documentation . . . 126
7.15.2.1 left . . . 126
7.15.2.2 op . . . 126
7.15.2.3 right . . . 126
7.16 l0_ast.Block Class Reference . . . 127
7.16.1 Detailed Description . . . 128
7.16.2 Member Data Documentation . . . 128
7.16.2.1 stmts . . . 128
7.17 l0_ast.BoolLiteral Class Reference . . . 128
7.17.1 Detailed Description . . . 129
7.17.2 Member Data Documentation . . . 129
7.17.2.1 value . . . 129
7.18 l0_ast.BreakStmt Class Reference . . . 130
7.18.1 Detailed Description . . . 131
7.19 BuildPreparedInput Struct Reference . . . 131
7.19.1 Detailed Description . . . 131
7.19.2 Member Data Documentation . . . 131
7.19.2.1 entry_module . . . 131
7.19.2.2 search_paths . . . 132
7.20 l0_types.BuiltinType Class Reference . . . 132
7.20.1 Detailed Description . . . 133
7.20.2 Member Data Documentation . . . 133
7.20.2.1 name . . . 133
7.21 ByteArray Struct Reference . . . 133
7.21.1 Detailed Description . . . 133
7.21.2 Member Data Documentation . . . 134

7.21.2.1 storage . . . 134
7.22 l0_ast.ByteLiteral Class Reference . . . 134
7.22.1 Detailed Description . . . 135
7.22.2 Member Data Documentation . . . 135
7.22.2.1 value . . . 135
7.23 l0_ast.CallExpr Class Reference . . . 136
7.23.1 Detailed Description . . . 137
7.23.2 Member Data Documentation . . . 137
7.23.2.1 args . . . 137
7.23.2.2 callee . . . 137
7.24 CaseArm Struct Reference . . . 137
7.24.1 Detailed Description . . . 138
7.24.2 Member Data Documentation . . . 138
7.24.2.1 body . . . 138
7.24.2.2 literal . . . 138
7.24.2.3 span . . . 138
7.25 l0_ast.CaseArm Class Reference . . . 138
7.25.1 Detailed Description . . . 139
7.25.2 Member Data Documentation . . . 139
7.25.2.1 body . . . 139
7.25.2.2 literal . . . 140
7.26 CaseElse Struct Reference . . . 140
7.26.1 Detailed Description . . . 140
7.26.2 Member Data Documentation . . . 140
7.26.2.1 body . . . 140
7.26.2.2 span . . . 141
7.27 l0_ast.CaseElse Class Reference . . . 141
7.27.1 Detailed Description . . . 142
7.27.2 Member Data Documentation . . . 142
7.27.2.1 body . . . 142
7.28 l0_ast.CaseStmt Class Reference . . . 142
7.28.1 Detailed Description . . . 143
7.28.2 Member Data Documentation . . . 143
7.28.2.1 arms . . . 143
7.28.2.2 else_arm . . . 144
7.28.2.3 expr . . . 144
7.29 l0_ast.CastExpr Class Reference . . . 144
7.29.1 Detailed Description . . . 145
7.29.2 Member Data Documentation . . . 145
7.29.2.1 expr . . . 145
7.29.2.2 target_type . . . 146
7.30 CCodeBuilder Struct Reference . . . 146

7.30.1 Detailed Description . . . 146
7.30.2 Member Data Documentation . . . 146
7.30.2.1 cached_level . . . 146
7.30.2.2 cached_prefix . . . 146
7.30.2.3 indent_level . . . 146
7.30.2.4 indent_str . . . 146
7.30.2.5 lines . . . 147
7.31 l0_c_emitter.CCodeBuilder Class Reference . . . 147
7.31.1 Detailed Description . . . 148
7.31.2 Member Function Documentation . . . 148
7.31.2.1 dedent() . . . 148
7.31.2.2 emit() . . . 148
7.31.2.3 emit_raw() . . . 148
7.31.2.4 indent() . . . 149
7.31.2.5 to_string() . . . 149
7.31.3 Member Data Documentation . . . 149
7.31.3.1 indent_level . . . 149
7.31.3.2 indent_str . . . 149
7.31.3.3 lines [1/2] . . . 149
7.31.3.4 lines [2/2] . . . 149
7.32 CEmitter Struct Reference . . . 150
7.32.1 Detailed Description . . . 150
7.32.2 Member Data Documentation . . . 150
7.32.2.1 active_line_directive . . . 150
7.32.2.2 analysis . . . 150
7.32.2.3 current_module . . . 151
7.32.2.4 opt_emitted . . . 151
7.32.2.5 opt_wrappers . . . 151
7.32.2.6 out . . . 151
7.32.2.7 tmp_counter . . . 151
7.33 l0_c_emitter.CEmitter Class Reference . . . 151
7.33.1 Detailed Description . . . 158
7.33.2 Member Function Documentation . . . 159
7.33.2.1 _collect_opt_wrappers_from_type() . . . 159
7.33.2.2 _emit_cleanup_by_type() . . . 159
7.33.2.3 _emit_enum_cleanup_switch() . . . 159
7.33.2.4 _emit_enum_value_cleanup() . . . 160
7.33.2.5 _emit_field_cleanup() . . . 160
7.33.2.6 _emit_optional_wrapper() . . . 160
7.33.2.7 _enum_has_arc_data() . . . 160
7.33.2.8 _get_enum_info() . . . 161
7.33.2.9 _get_struct_info() . . . 161

7.33.2.10 _is_early_inner() . . . 162
7.33.2.11 _is_niche_nullable() . . . 162
7.33.2.12 _iter_variant_cleanup_fields() . . . 162
7.33.2.13 _opt_key_for_type() . . . 163
7.33.2.14 _opt_wrapper_name_for_inner() . . . 163
7.33.2.15 _string_token_to_c_bytes_and_len() . . . 163
7.33.2.16 emit_alloc_obj() . . . 163
7.33.2.17 emit_assignment() . . . 163
7.33.2.18 emit_binary_op() . . . 164
7.33.2.19 emit_block_end() . . . 164
7.33.2.20 emit_block_start() . . . 164
7.33.2.21 emit_bool_literal() . . . 164
7.33.2.22 emit_byte_literal() . . . 165
7.33.2.23 emit_case_label() . . . 165
7.33.2.24 emit_cast() . . . 165
7.33.2.25 emit_checked_int_add() . . . 166
7.33.2.26 emit_checked_int_div() . . . 166
7.33.2.27 emit_checked_int_mod() . . . 166
7.33.2.28 emit_checked_int_mul() . . . 166
7.33.2.29 emit_checked_int_sub() . . . 167
7.33.2.30 emit_checked_narrow_cast() . . . 167
7.33.2.31 emit_checked_pointer_assignment() . . . 167
7.33.2.32 emit_checked_ptr_access() . . . 167
7.33.2.33 emit_checked_ptr_access_for_base() . . . 167
7.33.2.34 emit_checked_ptr_index_access() . . . 168
7.33.2.35 emit_comment() . . . 168
7.33.2.36 emit_condition_binary_op() . . . 168
7.33.2.37 emit_condition_pointer_null_check() . . . 168
7.33.2.38 emit_condition_string_compare_call() . . . 168
7.33.2.39 emit_const_bool_literal() . . . 169
7.33.2.40 emit_const_string_literal() . . . 169
7.33.2.41 emit_default_label() . . . 169
7.33.2.42 emit_deref_lvalue() . . . 169
7.33.2.43 emit_discard_expr() . . . 170
7.33.2.44 emit_drop_begin_expr() . . . 170
7.33.2.45 emit_drop_finish_call() . . . 170
7.33.2.46 emit_else() . . . 170
7.33.2.47 emit_enum() . . . 170
7.33.2.48 emit_enum_cleanup() . . . 171
7.33.2.49 emit_enum_payload_field_access() . . . 171
7.33.2.50 emit_enum_tag() . . . 171
7.33.2.51 emit_enum_tag_access() . . . 172

7.33.2.52 emit_enum_variant_init() . . . 172
7.33.2.53 emit_exit_switch() . . . 172
7.33.2.54 emit_expr_stmt() . . . 172
7.33.2.55 emit_field_access() . . . 172
7.33.2.56 emit_field_lvalue() . . . 173
7.33.2.57 emit_for_loop_end() . . . 173
7.33.2.58 emit_for_loop_start() . . . 173
7.33.2.59 emit_forward_decls() . . . 173
7.33.2.60 emit_function_call() . . . 173
7.33.2.61 emit_function_declaration() . . . 174
7.33.2.62 emit_function_definition_footer() . . . 174
7.33.2.63 emit_function_definition_header() . . . 174
7.33.2.64 emit_goto() . . . 175
7.33.2.65 emit_header() . . . 175
7.33.2.66 emit_if_header() . . . 175
7.33.2.67 emit_index_lvalue() . . . 175
7.33.2.68 emit_int_literal() . . . 175
7.33.2.69 emit_label() . . . 176
7.33.2.70 emit_let_decl() . . . 176
7.33.2.71 emit_let_declaration() . . . 176
7.33.2.72 emit_line_directive() . . . 177
7.33.2.73 emit_main_wrapper() . . . 177
7.33.2.74 emit_match_scrutinee_decl() . . . 177
7.33.2.75 emit_match_switch_start() . . . 178
7.33.2.76 emit_module_comment() . . . 178
7.33.2.77 emit_module_separator() . . . 178
7.33.2.78 emit_negated_condition() . . . 178
7.33.2.79 emit_none_value_for_nullable() . . . 178
7.33.2.80 emit_null_assignment() . . . 179
7.33.2.81 emit_null_check_eq() . . . 179
7.33.2.82 emit_null_check_ne() . . . 179
7.33.2.83 emit_null_literal() . . . 179
7.33.2.84 emit_optional_has_value() . . . 180
7.33.2.85 emit_optional_value() . . . 180
7.33.2.86 emit_optional_wrapper_for_defined_type() . . . 180
7.33.2.87 emit_optional_wrappers() . . . 180
7.33.2.88 emit_ord() . . . 181
7.33.2.89 emit_paren_expr() . . . 181
7.33.2.90 emit_pattern_binding_init() . . . 181
7.33.2.91 emit_pointer_assignment() . . . 182
7.33.2.92 emit_pointer_null_check() . . . 182
7.33.2.93 emit_pointer_type() . . . 182

7.33.2.94 emit_return_stmt() . . . 183
7.33.2.95 emit_section_comment() . . . 183
7.33.2.96 emit_sizeof_type() . . . 183
7.33.2.97 emit_some_value_for_nullable() . . . 184
7.33.2.98 emit_string_compare_call() . . . 184
7.33.2.99 emit_string_concat_call() . . . 184
7.33.2.100 emit_string_equals_call() . . . 184
7.33.2.101 emit_string_literal() . . . 185
7.33.2.102 emit_string_release() . . . 185
7.33.2.103 emit_string_retain() . . . 185
7.33.2.104 emit_struct() . . . 185
7.33.2.105 emit_struct_cleanup() . . . 187
7.33.2.106 emit_struct_constructor() . . . 187
7.33.2.107 emit_struct_constructor_for_type() . . . 187
7.33.2.108 emit_struct_init() . . . 188
7.33.2.109 emit_struct_init_from_fields() . . . 188
7.33.2.110 emit_struct_static_initializer() . . . 188
7.33.2.111 emit_struct_static_initializer_for_type() . . . 188
7.33.2.112 emit_switch_end() . . . 189
7.33.2.113 emit_switch_start() . . . 189
7.33.2.114 emit_temp_decl() . . . 189
7.33.2.115 emit_top_level_let_cleanup() . . . 189
7.33.2.116 emit_try_check_niche() . . . 189
7.33.2.117 emit_try_check_value() . . . 190
7.33.2.118 emit_try_extract_value() . . . 190
7.33.2.119 emit_type() . . . 190
7.33.2.120 emit_unary_op() . . . 191
7.33.2.121 emit_unreachable_comment() . . . 191
7.33.2.122 emit_unreachable_marker() . . . 191
7.33.2.123 emit_unwrap_opt() . . . 192
7.33.2.124 emit_unwrap_ptr() . . . 192
7.33.2.125 emit_value_cleanup() . . . 192
7.33.2.126 emit_var_ref() . . . 192
7.33.2.127 emit_variant_constructor() . . . 193
7.33.2.128 emit_variant_constructor_for_type() . . . 193
7.33.2.129 emit_variant_static_initializer() . . . 193
7.33.2.130 emit_variant_static_initializer_for_type() . . . 193
7.33.2.131 emit_while_header() . . . 194
7.33.2.132 emit_widen_int() . . . 194
7.33.2.133 emit_zero_init() . . . 194
7.33.2.134 find_variant_decl() . . . 195
7.33.2.135 fresh_tmp() . . . 195

7.33.2.136 get_output() . . . 195
7.33.2.137 ice() . . . 196
7.33.2.138 is_niche_nullable() . . . 196
7.33.2.139 mangle_enum_name() . . . 196
7.33.2.140 mangle_function_name() . . . 197
7.33.2.141 mangle_identifier() . . . 197
7.33.2.142 mangle_let_name() . . . 197
7.33.2.143 mangle_struct_name() . . . 198
7.33.2.144 prepare_optional_wrappers() . . . 198
7.33.2.145 restore_active_line_directive() . . . 198
7.33.2.146 set_analysis() . . . 198
7.33.3 Member Data Documentation . . . 199
7.33.3.1 _active_line_directive [1/2] . . . 199
7.33.3.2 _active_line_directive [2/2] . . . 199
7.33.3.3 _opt_emitted . . . 199
7.33.3.4 _opt_wrappers . . . 199
7.33.3.5 _tmp_counter . . . 199
7.33.3.6 analysis [1/2] . . . 199
7.33.3.7 analysis [2/2] . . . 199
7.33.3.8 C_KEYWORDS . . . 199
7.33.3.9 current_module . . . 200
7.33.3.10 out . . . 200
7.34 CharBuffer Struct Reference . . . 200
7.34.1 Detailed Description . . . 200
7.34.2 Member Data Documentation . . . 201
7.34.2.1 chars . . . 201
7.35 CivilDate Struct Reference . . . 201
7.35.1 Detailed Description . . . 201
7.35.2 Member Data Documentation . . . 201
7.35.2.1 day . . . 201
7.35.2.2 month . . . 201
7.35.2.3 weekday . . . 201
7.35.2.4 year . . . 201
7.35.2.5 yearday . . . 202
7.36 CleanupHeaderGuard Struct Reference . . . 202
7.36.1 Detailed Description . . . 202
7.36.2 Member Data Documentation . . . 202
7.36.2.1 guarded_decl_ids . . . 202
7.36.2.2 parent . . . 203
7.37 CliOptions Struct Reference . . . 203
7.37.1 Detailed Description . . . 203
7.37.2 Member Data Documentation . . . 203

7.37.2.1 all_modules . . . 203
7.37.2.2 c_compiler . . . 204
7.37.2.3 c_options . . . 204
7.37.2.4 c_sources . . . 204
7.37.2.5 check_basic . . . 204
7.37.2.6 entry_target . . . 204
7.37.2.7 extra_target_count . . . 204
7.37.2.8 include_eof . . . 204
7.37.2.9 interface_paths . . . 204
7.37.2.10 keep_c . . . 204
7.37.2.11 log . . . 204
7.37.2.12 mode . . . 205
7.37.2.13 mode_explicit . . . 205
7.37.2.14 no_line_directives . . . 205
7.37.2.15 output . . . 205
7.37.2.16 program_args . . . 205
7.37.2.17 project_roots . . . 205
7.37.2.18 runtime_include . . . 205
7.37.2.19 runtime_lib . . . 205
7.37.2.20 sys_roots . . . 205
7.37.2.21 trace_arc . . . 205
7.37.2.22 trace_memory . . . 206
7.37.2.23 unchecked . . . 206
7.37.2.24 verbosity . . . 206
7.38 CliParseResult Struct Reference . . . 206
7.38.1 Detailed Description . . . 207
7.38.2 Member Data Documentation . . . 207
7.38.2.1 diags . . . 207
7.38.2.2 help_requested . . . 207
7.38.2.3 options . . . 207
7.38.2.4 rich_log . . . 207
7.38.2.5 verbosity . . . 207
7.38.2.6 version_requested . . . 207
7.39 CodegenOptions Struct Reference . . . 207
7.39.1 Detailed Description . . . 208
7.39.2 Member Data Documentation . . . 208
7.39.2.1 emit_line_directives . . . 208
7.39.2.2 rt_check_basic . . . 208
7.39.2.3 rt_unchecked . . . 208
7.39.2.4 trace_arc . . . 208
7.39.2.5 trace_memory . . . 208
7.40 l0_context.CompilationContext Class Reference . . . 209

7.40.1 Detailed Description . . . 209
7.40.2 Member Function Documentation . . . 210
7.40.2.1 default() . . . 210
7.40.3 Member Data Documentation . . . 210
7.40.3.1 emit_line_directives . . . 210
7.40.3.2 log_level . . . 210
7.40.3.3 log_rich_format . . . 210
7.40.3.4 rt_check_basic . . . 210
7.40.3.5 rt_unchecked . . . 210
7.40.3.6 trace_arc . . . 210
7.40.3.7 trace_memory . . . 210
7.41 l0_compilation.CompilationUnit Class Reference . . . 211
7.41.1 Detailed Description . . . 211
7.41.2 Member Function Documentation . . . 212
7.41.2.1 __contains__() . . . 212
7.41.2.2 entry_name() . . . 212
7.41.3 Member Data Documentation . . . 212
7.41.3.1 entry_module . . . 212
7.41.3.2 modules . . . 212
7.42 CompilerWorkspace Struct Reference . . . 212
7.42.1 Detailed Description . . . 212
7.42.2 Member Data Documentation . . . 213
7.42.2.1 children . . . 213
7.42.2.2 root . . . 213
7.43 l0_ast.ContinueStmt Class Reference . . . 213
7.43.1 Detailed Description . . . 214
7.44 DateTime Struct Reference . . . 214
7.44.1 Detailed Description . . . 215
7.44.2 Member Data Documentation . . . 215
7.44.2.1 day . . . 215
7.44.2.2 hour . . . 215
7.44.2.3 is_dst . . . 215
7.44.2.4 minute . . . 215
7.44.2.5 month . . . 215
7.44.2.6 nanosecond . . . 215
7.44.2.7 second . . . 216
7.44.2.8 utc_offset_sec . . . 216
7.44.2.9 weekday . . . 216
7.44.2.10 year . . . 216
7.44.2.11 yearday . . . 216
7.45 DiagCollector Struct Reference . . . 216
7.45.1 Detailed Description . . . 216

7.45.2 Member Data Documentation . . . 216
7.45.2.1 error_count . . . 216
7.45.2.2 items . . . 217
7.46 Diagnostic Struct Reference . . . 217
7.46.1 Detailed Description . . . 217
7.46.2 Member Data Documentation . . . 217
7.46.2.1 code . . . 217
7.46.2.2 column . . . 217
7.46.2.3 column_end . . . 217
7.46.2.4 filename . . . 218
7.46.2.5 line . . . 218
7.46.2.6 line_end . . . 218
7.46.2.7 message . . . 218
7.46.2.8 phase . . . 218
7.46.2.9 severity . . . 218
7.47 l0_diagnostics.Diagnostic Class Reference . . . 218
7.47.1 Detailed Description . . . 219
7.47.2 Member Function Documentation . . . 219
7.47.2.1 format() . . . 219
7.47.3 Member Data Documentation . . . 219
7.47.3.1 column . . . 219
7.47.3.2 end_column . . . 220
7.47.3.3 end_line . . . 220
7.47.3.4 filename . . . 220
7.47.3.5 kind . . . 220
7.47.3.6 line . . . 220
7.47.3.7 message . . . 220
7.47.3.8 module_name . . . 220
7.48 DriverState Struct Reference . . . 221
7.48.1 Detailed Description . . . 221
7.48.2 Member Data Documentation . . . 221
7.48.2.1 diags . . . 221
7.48.2.2 entry_module . . . 221
7.48.2.3 loaded . . . 222
7.48.2.4 source_names . . . 222
7.48.2.5 source_texts . . . 222
7.48.2.6 units . . . 222
7.49 DriverUnit Struct Reference . . . 222
7.49.1 Detailed Description . . . 223
7.49.2 Member Data Documentation . . . 223
7.49.2.1 module_name . . . 223
7.49.2.2 parse_result . . . 223

7.49.2.3 path . . . 223
7.50 l0_ast.DropStmt Class Reference . . . 223
7.50.1 Detailed Description . . . 224
7.50.2 Member Data Documentation . . . 224
7.50.2.1 name . . . 224
7.51 Duration Struct Reference . . . 225
7.51.1 Detailed Description . . . 225
7.51.2 Member Data Documentation . . . 225
7.51.2.1 nsec . . . 225
7.51.2.2 sec . . . 225
7.52 EnumDecl Struct Reference . . . 225
7.52.1 Detailed Description . . . 226
7.52.2 Member Data Documentation . . . 226
7.52.2.1 name . . . 226
7.52.2.2 span . . . 226
7.52.2.3 variants . . . 226
7.53 l0_ast.EnumDecl Class Reference . . . 226
7.53.1 Detailed Description . . . 227
7.53.2 Member Data Documentation . . . 227
7.53.2.1 name . . . 227
7.53.2.2 variants . . . 228
7.54 EnumInfo Struct Reference . . . 228
7.54.1 Detailed Description . . . 229
7.54.2 Member Data Documentation . . . 229
7.54.2.1 enum_type . . . 229
7.54.2.2 filename . . . 229
7.54.2.3 key . . . 229
7.54.2.4 module_name . . . 229
7.54.2.5 name . . . 229
7.54.2.6 span . . . 229
7.54.2.7 variants . . . 229
7.55 l0_signatures.EnumInfo Class Reference . . . 230
7.55.1 Detailed Description . . . 230
7.55.2 Member Data Documentation . . . 230
7.55.2.1 enum_type . . . 230
7.55.2.2 variants . . . 230
7.56 l0_types.EnumType Class Reference . . . 231
7.56.1 Detailed Description . . . 231
7.56.2 Member Data Documentation . . . 232
7.56.2.1 module . . . 232
7.56.2.2 name . . . 232
7.57 EnumVariant Struct Reference . . . 232

7.57.1 Detailed Description . . . 232
7.57.2 Member Data Documentation . . . 233
7.57.2.1 fields . . . 233
7.57.2.2 name . . . 233
7.58 l0_ast.EnumVariant Class Reference . . . 233
7.58.1 Detailed Description . . . 234
7.58.2 Member Data Documentation . . . 234
7.58.2.1 fields . . . 234
7.58.2.2 name . . . 234
7.59 EnumVariantInfo Struct Reference . . . 234
7.59.1 Detailed Description . . . 235
7.59.2 Member Data Documentation . . . 235
7.59.2.1 field_types . . . 235
7.59.2.2 name . . . 235
7.60 l0_signatures.EnumVariantInfo Class Reference . . . 235
7.60.1 Detailed Description . . . 236
7.60.2 Member Data Documentation . . . 236
7.60.2.1 field_types . . . 236
7.60.2.2 name . . . 236
7.61 EscapedChar Struct Reference . . . 236
7.61.1 Detailed Description . . . 236
7.61.2 Member Data Documentation . . . 236
7.61.2.1 char_value . . . 236
7.61.2.2 escape_str . . . 237
7.62 l0_string_escape.EscapeDecodeError Class Reference . . . 237
7.62.1 Detailed Description . . . 238
7.62.2 Member Data Documentation . . . 238
7.62.2.1 code . . . 238
7.62.2.2 details . . . 238
7.63 l0_ast.Expr Class Reference . . . 238
7.63.1 Detailed Description . . . 240
7.64 ExprArena Struct Reference . . . 240
7.64.1 Detailed Description . . . 241
7.64.2 Member Data Documentation . . . 241
7.64.2.1 nodes . . . 241
7.65 ExpressionTypeChecker Struct Reference . . . 242
7.65.1 Detailed Description . . . 243
7.65.2 Member Data Documentation . . . 243
7.65.2.1 alive_scopes . . . 243
7.65.2.2 cleanup_header_guard_stack . . . 243
7.65.2.3 current_env . . . 243
7.65.2.4 current_func . . . 243

7.65.2.5 current_module . . . 243
7.65.2.6 current_ret_type . . . 243
7.65.2.7 current_scope . . . 243
7.65.2.8 current_unit . . . 243
7.65.2.9 decl_scope_defs . . . 243
7.65.2.10 decl_scopes . . . 244
7.65.2.11 liveness_diagnostics_only . . . 244
7.65.2.12 loop_depth . . . 244
7.65.2.13 loop_flow_capture_stack . . . 244
7.65.2.14 next_stmt_unreachable . . . 244
7.65.2.15 result . . . 244
7.65.2.16 suppress_diagnostics . . . 244
7.65.2.17 suppress_liveness_diagnostics . . . 244
7.66 l0_expr_types.ExpressionTypeChecker Class Reference . . . 245
7.66.1 Detailed Description . . . 249
7.66.2 Member Function Documentation . . . 249
7.66.2.1 __post_init__() . . . 249
7.66.2.2 _binary_equality() . . . 249
7.66.2.3 _binary_expect_both_bool() . . . 250
7.66.2.4 _binary_expect_both_int() . . . 250
7.66.2.5 _can_assign() . . . 250
7.66.2.6 _case_literal_info() . . . 250
7.66.2.7 _check_block() . . . 250
7.66.2.8 _check_dead_liveness_block() . . . 251
7.66.2.9 _check_expr_liveness() . . . 251
7.66.2.10 _check_function() . . . 251
7.66.2.11 _check_loop_iteration() . . . 251
7.66.2.12 _check_return() . . . 252
7.66.2.13 _check_stmt() . . . 252
7.66.2.14 _check_update_from_backedge_states() . . . 252
7.66.2.15 _clone_alive_scopes() . . . 252
7.66.2.16 _declare_local() . . . 252
7.66.2.17 _decode_escaped_bytes() . . . 253
7.66.2.18 _describe_lvalue() . . . 253
7.66.2.19 _diagnostic() . . . 253
7.66.2.20 _error() . . . 253
7.66.2.21 _expr_contains_try() . . . 253
7.66.2.22 _get_const_int_for_explicit_cast() . . . 254
7.66.2.23 _infer_binary() . . . 254
7.66.2.24 _infer_call() . . . 254
7.66.2.25 _infer_cast() . . . 254
7.66.2.26 _infer_expr() . . . 254

7.66.2.27 _infer_field_access() . . . 255
7.66.2.28 _infer_index() . . . 255
7.66.2.29 _infer_new() . . . 255
7.66.2.30 _infer_ord_intrinsic() . . . 255
7.66.2.31 _infer_sizeof_intrinsic() . . . 255
7.66.2.32 _infer_struct_constructor() . . . 256
7.66.2.33 _infer_try() . . . 256
7.66.2.34 _infer_type_expr() . . . 256
7.66.2.35 _infer_unary() . . . 256
7.66.2.36 _infer_var_ref() . . . 256
7.66.2.37 _infer_variant_constructor() . . . 256
7.66.2.38 _is_bool() . . . 257
7.66.2.39 _is_const_null_for_explicit_cast() . . . 257
7.66.2.40 _is_guarded_cleanup_header_ref() . . . 257
7.66.2.41 _is_int_assignable() . . . 257
7.66.2.42 _is_nullable_or_ptr() . . . 257
7.66.2.43 _is_string() . . . 257
7.66.2.44 _is_void() . . . 258
7.66.2.45 _lookup_alive() . . . 258
7.66.2.46 _lookup_local() . . . 258
7.66.2.47 _lookup_local_scope_index() . . . 258
7.66.2.48 _loop_liveness_fixed_point() . . . 258
7.66.2.49 _make_param_alive_scope() . . . 258
7.66.2.50 _make_param_scope() . . . 259
7.66.2.51 _meet_alive_scopes() . . . 259
7.66.2.52 _meet_alive_state() . . . 259
7.66.2.53 _pop_scope() . . . 259
7.66.2.54 _push_scope() . . . 259
7.66.2.55 _reject_name_qualifier() . . . 259
7.66.2.56 _resolve_type_ref() . . . 260
7.66.2.57 _set_alive() . . . 260
7.66.2.58 _stmt_contains_try() . . . 260
7.66.2.59 _store_sizeof_target() . . . 260
7.66.2.60 _store_var_resolution() . . . 260
7.66.2.61 _try_infer_intrinsic() . . . 260
7.66.2.62 _try_resolve_type_name() . . . 261
7.66.2.63 _types_equal() . . . 261
7.66.2.64 _validate_match_variant_pattern() . . . 261
7.66.2.65 _warn() . . . 261
7.66.2.66 check() . . . 261
7.66.3 Member Data Documentation . . . 262
7.66.3.1 _alive_scopes . . . 262

7.66.3.2 _breakable_loop_depth . . . 262
7.66.3.3 _cleanup_header_ref_guard_stack . . . 262
7.66.3.4 _current_func_env . . . 262
7.66.3.5 _current_func_type . . . 262
7.66.3.6 _liveness_diagnostics_only . . . 262
7.66.3.7 _local_scopes . . . 262
7.66.3.8 _loop_flow_capture_stack . . . 262
7.66.3.9 _next_stmt_unreachable . . . 262
7.66.3.10 _return_paths . . . 262
7.66.3.11 _suppress_diagnostics . . . 263
7.66.3.12 _suppress_liveness_diagnostics . . . 263
7.66.3.13 analysis . . . 263
7.66.3.14 bool_type . . . 263
7.66.3.15 byte_type . . . 263
7.66.3.16 diagnostics . . . 263
7.66.3.17 enum_infos . . . 263
7.66.3.18 expr_types . . . 263
7.66.3.19 func_envs . . . 263
7.66.3.20 func_types . . . 263
7.66.3.21 int_type . . . 264
7.66.3.22 module_envs . . . 264
7.66.3.23 string_type . . . 264
7.66.3.24 struct_infos . . . 264
7.66.3.25 void_type . . . 264
7.67 ExprNode Struct Reference . . . 264
7.67.1 Detailed Description . . . 265
7.67.2 Member Data Documentation . . . 265
7.67.2.1 bool_value . . . 265
7.67.2.2 expr_a . . . 265
7.67.2.3 expr_b . . . 265
7.67.2.4 expr_c . . . 265
7.67.2.5 ids . . . 265
7.67.2.6 int_value . . . 265
7.67.2.7 kind . . . 266
7.67.2.8 module_path . . . 266
7.67.2.9 name_qualifier . . . 266
7.67.2.10 span . . . 266
7.67.2.11 text_value . . . 266
7.67.2.12 type_ref . . . 266
7.68 l0_ast.ExprStmt Class Reference . . . 266
7.68.1 Detailed Description . . . 267
7.68.2 Member Data Documentation . . . 267

7.68.2.1 expr . . . 267
7.69 l0_ast.FieldAccessExpr Class Reference . . . 268
7.69.1 Detailed Description . . . 269
7.69.2 Member Data Documentation . . . 269
7.69.2.1 field . . . 269
7.69.2.2 obj . . . 269
7.70 FieldDecl Struct Reference . . . 269
7.70.1 Detailed Description . . . 270
7.70.2 Member Data Documentation . . . 270
7.70.2.1 name . . . 270
7.70.2.2 type_ref . . . 270
7.71 l0_ast.FieldDecl Class Reference . . . 270
7.71.1 Detailed Description . . . 271
7.71.2 Member Data Documentation . . . 271
7.71.2.1 name . . . 271
7.71.2.2 type . . . 272
7.72 FileInfo Struct Reference . . . 272
7.72.1 Detailed Description . . . 272
7.72.2 Member Data Documentation . . . 272
7.72.2.1 exists . . . 272
7.72.2.2 is_dir . . . 272
7.72.2.3 is_file . . . 272
7.72.2.4 mtime_nsec . . . 272
7.72.2.5 mtime_sec . . . 273
7.72.2.6 size . . . 273
7.73 l0_ast.ForStmt Class Reference . . . 273
7.73.1 Detailed Description . . . 274
7.73.2 Member Data Documentation . . . 275
7.73.2.1 body . . . 275
7.73.2.2 cond . . . 275
7.73.2.3 init . . . 275
7.73.2.4 update . . . 275
7.74 FuncDecl Struct Reference . . . 275
7.74.1 Detailed Description . . . 276
7.74.2 Member Data Documentation . . . 276
7.74.2.1 body . . . 276
7.74.2.2 is_extern . . . 276
7.74.2.3 name . . . 276
7.74.2.4 params . . . 276
7.74.2.5 return_type . . . 276
7.74.2.6 span . . . 276
7.75 l0_ast.FuncDecl Class Reference . . . 277

7.75.1 Detailed Description . . . 278
7.75.2 Member Data Documentation . . . 278
7.75.2.1 body . . . 278
7.75.2.2 is_extern . . . 278
7.75.2.3 name . . . 278
7.75.2.4 params . . . 278
7.75.2.5 return_type . . . 278
7.76 FunctionEnv Struct Reference . . . 279
7.76.1 Detailed Description . . . 279
7.76.2 Member Data Documentation . . . 279
7.76.2.1 arm_scopes . . . 279
7.76.2.2 block_scopes . . . 280
7.76.2.3 func_decl . . . 280
7.76.2.4 func_name . . . 280
7.76.2.5 key . . . 280
7.76.2.6 module_name . . . 280
7.76.2.7 pattern_arena . . . 280
7.76.2.8 root_scope . . . 280
7.76.2.9 scopes . . . 280
7.76.2.10 stmt_arena . . . 280
7.77 l0_locals.FunctionEnv Class Reference . . . 281
7.77.1 Detailed Description . . . 281
7.77.2 Member Data Documentation . . . 281
7.77.2.1 func . . . 281
7.77.2.2 module_name . . . 282
7.77.2.3 root_scope . . . 282
7.78 l0_types.FuncType Class Reference . . . 282
7.78.1 Detailed Description . . . 283
7.78.2 Member Data Documentation . . . 283
7.78.2.1 params . . . 283
7.78.2.2 result . . . 283
7.79 l0_internal_error.ICELocation Class Reference . . . 283
7.79.1 Detailed Description . . . 284
7.79.2 Member Data Documentation . . . 284
7.79.2.1 filename . . . 284
7.79.2.2 span . . . 284
7.80 l0_ast.IfStmt Class Reference . . . 284
7.80.1 Detailed Description . . . 285
7.80.2 Member Data Documentation . . . 285
7.80.2.1 cond . . . 285
7.80.2.2 else_stmt . . . 286
7.80.2.3 then_stmt . . . 286

7.81 ImplParseResult Struct Reference . . . 286
7.81.1 Detailed Description . . . 286
7.81.2 Member Data Documentation . . . 287
7.81.2.1 diags . . . 287
7.81.2.2 expr_arena . . . 287
7.81.2.3 pattern_arena . . . 287
7.81.2.4 root_module . . . 287
7.81.2.5 stmt_arena . . . 287
7.82 Import Struct Reference . . . 287
7.82.1 Detailed Description . . . 288
7.82.2 Member Data Documentation . . . 288
7.82.2.1 name . . . 288
7.82.2.2 span . . . 288
7.83 l0_ast.Import Class Reference . . . 288
7.83.1 Detailed Description . . . 289
7.83.2 Member Data Documentation . . . 289
7.83.2.1 name . . . 289
7.84 l0_driver.ImportCycleError Class Reference . . . 290
7.84.1 Detailed Description . . . 290
7.85 l0_ast.IndexExpr Class Reference . . . 291
7.85.1 Detailed Description . . . 292
7.85.2 Member Data Documentation . . . 292
7.85.2.1 array . . . 292
7.85.2.2 index . . . 292
7.86 l0_internal_error.InternalCompilerError Class Reference . . . 292
7.86.1 Detailed Description . . . 293
7.86.2 Constructor & Destructor Documentation . . . 293
7.86.2.1 __init__() . . . 293
7.86.3 Member Function Documentation . . . 294
7.86.3.1 format() . . . 294
7.86.4 Member Data Documentation . . . 294
7.86.4.1 loc . . . 294
7.86.4.2 message . . . 294
7.87 l0_ast.IntLiteral Class Reference . . . 294
7.87.1 Detailed Description . . . 296
7.87.2 Member Data Documentation . . . 296
7.87.2.1 value . . . 296
7.88 IntSet Struct Reference . . . 296
7.88.1 Detailed Description . . . 297
7.88.2 Member Data Documentation . . . 297
7.88.2.1 capacity . . . 297
7.88.2.2 count . . . 297

7.88.2.3 keys . . . 297
7.88.2.4 states . . . 297
7.88.2.5 tomb_count . . . 297
7.89 IntStringLinearMap Struct Reference . . . 298
7.89.1 Detailed Description . . . 298
7.89.2 Member Data Documentation . . . 298
7.89.2.1 base . . . 298
7.90 IntStringLinearMapEntry Struct Reference . . . 299
7.90.1 Detailed Description . . . 299
7.90.2 Member Data Documentation . . . 299
7.90.2.1 key . . . 299
7.90.2.2 value . . . 299
7.91 l0_opt_bool Struct Reference . . . 299
7.91.1 Detailed Description . . . 299
7.91.2 Member Data Documentation . . . 300
7.91.2.1 has_value . . . 300
7.91.2.2 value . . . 300
7.92 l0_opt_byte Struct Reference . . . 300
7.92.1 Detailed Description . . . 300
7.92.2 Member Data Documentation . . . 300
7.92.2.1 has_value . . . 300
7.92.2.2 value . . . 300
7.93 l0_opt_int Struct Reference . . . 301
7.93.1 Detailed Description . . . 301
7.93.2 Member Data Documentation . . . 301
7.93.2.1 has_value . . . 301
7.93.2.2 value . . . 301
7.94 l0_opt_string Struct Reference . . . 301
7.94.1 Detailed Description . . . 302
7.94.2 Member Data Documentation . . . 302
7.94.2.1 has_value . . . 302
7.94.2.2 value . . . 302
7.95 l0_string Struct Reference . . . 302
7.95.1 Detailed Description . . . 303
7.95.2 Member Data Documentation . . . 303
7.95.2.1 bytes . . . 303
7.95.2.2 [union] . . . 303
7.95.2.3 h_str . . . 303
7.95.2.4 int . . . 303
7.95.2.5 kind . . . 304
7.95.2.6 len . . . 304
7.95.2.7 [struct] . . . 304

7.96 l0_sys_rt_RtFileInfo Struct Reference . . . 304
7.96.1 Detailed Description . . . 305
7.96.2 Member Data Documentation . . . 305
7.96.2.1 exists . . . 305
7.96.2.2 is_dir . . . 305
7.96.2.3 is_file . . . 305
7.96.2.4 mtime_nsec . . . 305
7.96.2.5 mtime_sec . . . 305
7.96.2.6 size . . . 305
7.97 l0_sys_rt_RtTimeParts Struct Reference . . . 306
7.97.1 Detailed Description . . . 306
7.97.2 Member Data Documentation . . . 306
7.97.2.1 nsec . . . 306
7.97.2.2 sec . . . 306
7.98 l0_driver.L0Driver Class Reference . . . 306
7.98.1 Detailed Description . . . 307
7.98.2 Constructor & Destructor Documentation . . . 307
7.98.2.1 __init__() . . . 307
7.98.3 Member Function Documentation . . . 308
7.98.3.1 _load_single_file() . . . 308
7.98.3.2 _parse_source() . . . 308
7.98.3.3 analyze() . . . 308
7.98.3.4 build_compilation_unit() . . . 309
7.98.3.5 load_module() . . . 309
7.98.4 Member Data Documentation . . . 310
7.98.4.1 context . . . 310
7.98.4.2 diagnostics . . . 310
7.98.4.3 search_paths . . . 310
7.99 l0_ast.LetDecl Class Reference . . . 310
7.99.1 Detailed Description . . . 311
7.99.2 Member Data Documentation . . . 311
7.99.2.1 name . . . 311
7.99.2.2 type . . . 312
7.99.2.3 value . . . 312
7.100 LetDecl Struct Reference . . . 312
7.100.1 Detailed Description . . . 312
7.100.2 Member Data Documentation . . . 313
7.100.2.1 name . . . 313
7.100.2.2 span . . . 313
7.100.2.3 type_ref . . . 313
7.100.2.4 value . . . 313
7.101 l0_ast.LetStmt Class Reference . . . 313

7.101.1 Detailed Description . . . 314
7.101.2 Member Data Documentation . . . 314
7.101.2.1 name . . . 314
7.101.2.2 type . . . 315
7.101.2.3 value . . . 315
7.102 l0_lexer.Lexer Class Reference . . . 315
7.102.1 Detailed Description . . . 316
7.102.2 Constructor & Destructor Documentation . . . 317
7.102.2.1 __init__() . . . 317
7.102.3 Member Function Documentation . . . 317
7.102.3.1 _advance() . . . 317
7.102.3.2 _at_end() . . . 317
7.102.3.3 _collect_invalid_characters() . . . 317
7.102.3.4 _error() . . . 317
7.102.3.5 _lexer_error_token() . . . 318
7.102.3.6 _next_token() . . . 318
7.102.3.7 _peek() . . . 318
7.102.3.8 _peek_next() . . . 318
7.102.3.9 _read_byte_literal() . . . 318
7.102.3.10 _read_number() . . . 318
7.102.3.11 _read_string_literal() . . . 319
7.102.3.12 _read_valid_char_escape() . . . 319
7.102.3.13 _skip_ws_and_comments() . . . 319
7.102.3.14 _wrap_new_diagnostics() . . . 319
7.102.3.15 from_source() . . . 319
7.102.3.16 tokenize() . . . 320
7.102.4 Member Data Documentation . . . 320
7.102.4.1 _prev_ends_expression . . . 320
7.102.4.2 column . . . 320
7.102.4.3 diagnostics . . . 320
7.102.4.4 filename . . . 320
7.102.4.5 index . . . 320
7.102.4.6 length . . . 320
7.102.4.7 line . . . 320
7.102.4.8 source . . . 321
7.103 LexerState Struct Reference . . . 321
7.103.1 Detailed Description . . . 322
7.103.2 Member Data Documentation . . . 322
7.103.2.1 column . . . 322
7.103.2.2 diags . . . 322
7.103.2.3 filename . . . 322
7.103.2.4 index . . . 322

7.103.2.5 length . . . 322
7.103.2.6 line . . . 322
7.103.2.7 pending_errors . . . 322
7.103.2.8 prev_ends_expression . . . 322
7.103.2.9 queued_index . . . 322
7.103.2.10 queued_tokens . . . 323
7.103.2.11 source . . . 323
7.104 LinearMapBase Struct Reference . . . 323
7.104.1 Detailed Description . . . 323
7.104.2 Member Data Documentation . . . 324
7.104.2.1 entries . . . 324
7.104.2.2 key_size . . . 324
7.104.2.3 value_size . . . 324
7.105 LinearMapEntryBase Struct Reference . . . 324
7.105.1 Detailed Description . . . 324
7.105.2 Member Data Documentation . . . 324
7.105.2.1 hash_key . . . 324
7.105.2.2 hash_value . . . 325
7.106 l0_locals.LocalKind Class Reference . . . 325
7.106.1 Detailed Description . . . 326
7.106.2 Member Data Documentation . . . 326
7.106.2.1 LOCAL . . . 326
7.106.2.2 PARAM . . . 326
7.106.2.3 PATTERN_VAR . . . 326
7.107 l0_locals.LocalScopeResolver Class Reference . . . 326
7.107.1 Detailed Description . . . 327
7.107.2 Constructor & Destructor Documentation . . . 327
7.107.2.1 __init__() . . . 327
7.107.3 Member Function Documentation . . . 328
7.107.3.1 _bind_pattern_vars() . . . 328
7.107.3.2 _build_function_env() . . . 328
7.107.3.3 _declare() . . . 328
7.107.3.4 _visit_block() . . . 328
7.107.3.5 _visit_case_arm() . . . 329
7.107.3.6 _visit_case_else() . . . 329
7.107.3.7 _visit_match_arm() . . . 329
7.107.3.8 _visit_stmt() . . . 329
7.107.3.9 _visit_with_stmt() . . . 329
7.107.3.10 get_block_scope() . . . 329
7.107.3.11 get_match_arm_scope() . . . 330
7.107.3.12 resolve() . . . 330
7.107.4 Member Data Documentation . . . 330

7.107.4.1 modules . . . 330
7.108 LocalsResult Struct Reference . . . 331
7.108.1 Detailed Description . . . 331
7.108.2 Member Data Documentation . . . 331
7.108.2.1 by_name . . . 331
7.108.2.2 envs . . . 331
7.109 l0_locals.LocalSymbol Class Reference . . . 332
7.109.1 Detailed Description . . . 332
7.109.2 Member Data Documentation . . . 332
7.109.2.1 decl . . . 332
7.109.2.2 kind . . . 333
7.109.2.3 name . . . 333
7.109.2.4 type_ref . . . 333
7.110 LocalSymbol Struct Reference . . . 333
7.110.1 Detailed Description . . . 333
7.110.2 Member Data Documentation . . . 333
7.110.2.1 binding_slot . . . 333
7.110.2.2 decl_id . . . 333
7.110.2.3 decl_ptr . . . 334
7.110.2.4 kind . . . 334
7.110.2.5 name . . . 334
7.111 LogConfig Struct Reference . . . 334
7.111.1 Detailed Description . . . 334
7.111.2 Member Data Documentation . . . 334
7.111.2.1 level . . . 334
7.111.2.2 rich . . . 334
7.112 l0_context.LogLevel Class Reference . . . 335
7.112.1 Detailed Description . . . 335
7.112.2 Member Data Documentation . . . 336
7.112.2.1 DEBUG . . . 336
7.112.2.2 ERROR . . . 336
7.112.2.3 INFO . . . 336
7.112.2.4 SILENT . . . 336
7.112.2.5 WARNING . . . 336
7.113 LoopCleanupScopes Struct Reference . . . 336
7.113.1 Detailed Description . . . 337
7.113.2 Member Data Documentation . . . 337
7.113.2.1 break_scope . . . 337
7.113.2.2 break_scope_id . . . 337
7.113.2.3 continue_scope . . . 338
7.113.2.4 continue_scope_id . . . 338
7.113.2.5 parent . . . 338

7.114 LoopFixedPointFlow Struct Reference . . . 338
7.114.1 Detailed Description . . . 339
7.114.2 Member Data Documentation . . . 339
7.114.2.1 backedge_states . . . 339
7.114.2.2 capture . . . 339
7.114.2.3 head . . . 339
7.115 l0_expr_types.LoopFlowCapture Class Reference . . . 339
7.115.1 Detailed Description . . . 340
7.115.2 Member Data Documentation . . . 340
7.115.2.1 break_states . . . 340
7.115.2.2 continue_states . . . 340
7.116 LoopFlowCapture Struct Reference . . . 340
7.116.1 Detailed Description . . . 341
7.116.2 Member Data Documentation . . . 341
7.116.2.1 break_states . . . 341
7.116.2.2 continue_states . . . 341
7.116.2.3 parent . . . 341
7.117 LoopIterationFlow Struct Reference . . . 341
7.117.1 Detailed Description . . . 342
7.117.2 Member Data Documentation . . . 342
7.117.2.1 backedge_states . . . 342
7.117.2.2 body_flow . . . 342
7.117.2.3 capture . . . 343
7.118 LoopLabels Struct Reference . . . 343
7.118.1 Detailed Description . . . 343
7.118.2 Member Data Documentation . . . 343
7.118.2.1 break_label . . . 343
7.118.2.2 continue_label . . . 343
7.118.2.3 parent . . . 344
7.119 l0_ast.MatchArm Class Reference . . . 344
7.119.1 Detailed Description . . . 345
7.119.2 Member Data Documentation . . . 346
7.119.2.1 body . . . 346
7.119.2.2 pattern . . . 346
7.120 MatchArm Struct Reference . . . 346
7.120.1 Detailed Description . . . 346
7.120.2 Member Data Documentation . . . 347
7.120.2.1 body . . . 347
7.120.2.2 pattern . . . 347
7.120.2.3 span . . . 347
7.121 MatchArmScopeEntry Struct Reference . . . 347
7.121.1 Detailed Description . . . 348

7.121.2 Member Data Documentation . . . 348
7.121.2.1 arm . . . 348
7.121.2.2 scope . . . 348
7.122 l0_ast.MatchStmt Class Reference . . . 348
7.122.1 Detailed Description . . . 349
7.122.2 Member Data Documentation . . . 349
7.122.2.1 arms . . . 349
7.122.2.2 expr . . . 350
7.123 l0_ast.Module Class Reference . . . 350
7.123.1 Detailed Description . . . 351
7.123.2 Member Data Documentation . . . 351
7.123.2.1 decls . . . 351
7.123.2.2 filename . . . 351
7.123.2.3 imports . . . 351
7.123.2.4 name . . . 351
7.124 Module Struct Reference . . . 352
7.124.1 Detailed Description . . . 352
7.124.2 Member Data Documentation . . . 352
7.124.2.1 decls . . . 352
7.124.2.2 filename . . . 352
7.124.2.3 imports . . . 353
7.124.2.4 name . . . 353
7.124.2.5 span . . . 353
7.125 l0_symbols.ModuleEnv Class Reference . . . 353
7.125.1 Detailed Description . . . 354
7.125.2 Member Function Documentation . . . 354
7.125.2.1 name() . . . 354
7.125.3 Member Data Documentation . . . 354
7.125.3.1 all . . . 354
7.125.3.2 ambiguous_imports . . . 354
7.125.3.3 diagnostics . . . 354
7.125.3.4 imported . . . 355
7.125.3.5 locals . . . 355
7.125.3.6 module . . . 355
7.126 ModuleEnv Struct Reference . . . 355
7.126.1 Detailed Description . . . 356
7.126.2 Member Data Documentation . . . 356
7.126.2.1 all . . . 356
7.126.2.2 ambiguous_imports . . . 356
7.126.2.3 imported . . . 356
7.126.2.4 imported_modules . . . 356
7.126.2.5 locals . . . 356

7.126.2.6 module_name . . . 356
7.126.2.7 module_ptr . . . 357
7.127 MonotonicTime Struct Reference . . . 357
7.127.1 Detailed Description . . . 357
7.127.2 Member Data Documentation . . . 357
7.127.2.1 nsec . . . 357
7.127.2.2 sec . . . 357
7.128 NameResolution Struct Reference . . . 358
7.128.1 Detailed Description . . . 358
7.128.2 Member Data Documentation . . . 358
7.128.2.1 by_name . . . 358
7.128.2.2 envs . . . 358
7.129 l0_name_resolver.NameResolver Class Reference . . . 359
7.129.1 Detailed Description . . . 359
7.129.2 Constructor & Destructor Documentation . . . 359
7.129.2.1 __init__() . . . 359
7.129.3 Member Function Documentation . . . 360
7.129.3.1 _collect_locals() . . . 360
7.129.3.2 _define_local() . . . 360
7.129.3.3 _extern_signatures_compatible() . . . 360
7.129.3.4 _open_imports() . . . 360
7.129.3.5 resolve() . . . 361
7.129.4 Member Data Documentation . . . 361
7.129.4.1 cu . . . 361
7.130 l0_ast.NewExpr Class Reference . . . 361
7.130.1 Detailed Description . . . 363
7.130.2 Member Data Documentation . . . 363
7.130.2.1 args . . . 363
7.130.2.2 type_ref . . . 363
7.131 l0_ast.Node Class Reference . . . 363
7.131.1 Detailed Description . . . 365
7.131.2 Member Data Documentation . . . 365
7.131.2.1 span . . . 365
7.132 l0_types.NullableType Class Reference . . . 366
7.132.1 Detailed Description . . . 366
7.132.2 Member Data Documentation . . . 367
7.132.2.1 inner . . . 367
7.133 l0_ast.NullLiteral Class Reference . . . 367
7.133.1 Detailed Description . . . 368
7.134 l0_types.NullType Class Reference . . . 368
7.134.1 Detailed Description . . . 369
7.135 l0_ast.Param Class Reference . . . 369

7.135.1 Detailed Description . . . 371
7.135.2 Member Data Documentation . . . 371
7.135.2.1 name . . . 371
7.135.2.2 type . . . 371
7.136 Param Struct Reference . . . 371
7.136.1 Detailed Description . . . 372
7.136.2 Member Data Documentation . . . 372
7.136.2.1 name . . . 372
7.136.2.2 type_ref . . . 372
7.137 l0_ast.ParenExpr Class Reference . . . 372
7.137.1 Detailed Description . . . 373
7.137.2 Member Data Documentation . . . 373
7.137.2.1 inner . . . 373
7.138 l0_parser.Parser Class Reference . . . 374
7.138.1 Detailed Description . . . 377
7.138.2 Constructor & Destructor Documentation . . . 377
7.138.2.1 __init__() . . . 377
7.138.3 Member Function Documentation . . . 378
7.138.3.1 _advance() . . . 378
7.138.3.2 _at_case_arm_start() . . . 378
7.138.3.3 _at_end() . . . 378
7.138.3.4 _check() . . . 378
7.138.3.5 _check_reserved_binary_op() . . . 378
7.138.3.6 _emit_lexer_error() . . . 378
7.138.3.7 _emit_remaining_lexer_errors() . . . 379
7.138.3.8 _error() . . . 379
7.138.3.9 _error_bail() . . . 379
7.138.3.10 _error_unexpected() . . . 379
7.138.3.11 _expect() . . . 380
7.138.3.12 _expect_semicolon() . . . 380
7.138.3.13 _expect_variable_name() . . . 380
7.138.3.14 _extend_span() . . . 380
7.138.3.15 _get_dotted_module_name() . . . 381
7.138.3.16 _is_builtin_type_name() . . . 381
7.138.3.17 _is_unambiguous_type_start() . . . 381
7.138.3.18 _last() . . . 381
7.138.3.19 _logical_token_at_cursor() . . . 381
7.138.3.20 _lookahead_is_type_suffix() . . . 382
7.138.3.21 _match() . . . 382
7.138.3.22 _parse_add_expr() . . . 382
7.138.3.23 _parse_and_expr() . . . 382
7.138.3.24 _parse_block() . . . 382

7.138.3.25 _parse_break_stmt() . . . 382
7.138.3.26 _parse_call_argument() . . . 383
7.138.3.27 _parse_case_literal() . . . 383
7.138.3.28 _parse_case_stmt() . . . 383
7.138.3.29 _parse_cast_expr() . . . 383
7.138.3.30 _parse_continue_stmt() . . . 383
7.138.3.31 _parse_drop_stmt() . . . 383
7.138.3.32 _parse_enum() . . . 384
7.138.3.33 _parse_equality_expr() . . . 384
7.138.3.34 _parse_expr() . . . 384
7.138.3.35 _parse_extern_func() . . . 384
7.138.3.36 _parse_for_stmt() . . . 384
7.138.3.37 _parse_for_update_stmt() . . . 384
7.138.3.38 _parse_function() . . . 384
7.138.3.39 _parse_if_stmt() . . . 385
7.138.3.40 _parse_let_stmt() . . . 385
7.138.3.41 _parse_match_stmt() . . . 385
7.138.3.42 _parse_mul_expr() . . . 385
7.138.3.43 _parse_or_expr() . . . 385
7.138.3.44 _parse_pattern() . . . 385
7.138.3.45 _parse_postfix_expr() . . . 385
7.138.3.46 _parse_primary_expr() . . . 386
7.138.3.47 _parse_rel_expr() . . . 386
7.138.3.48 _parse_return_stmt() . . . 386
7.138.3.49 _parse_simple_stmt() . . . 386
7.138.3.50 _parse_stmt() . . . 386
7.138.3.51 _parse_struct() . . . 386
7.138.3.52 _parse_top_level_decl() . . . 386
7.138.3.53 _parse_top_level_let() . . . 387
7.138.3.54 _parse_type() . . . 387
7.138.3.55 _parse_type_alias() . . . 387
7.138.3.56 _parse_unary_expr() . . . 387
7.138.3.57 _parse_while_stmt() . . . 387
7.138.3.58 _parse_with_item() . . . 387
7.138.3.59 _parse_with_stmt() . . . 387
7.138.3.60 _peek() . . . 388
7.138.3.61 _skip_lexer_errors() . . . 388
7.138.3.62 _span_start() . . . 388
7.138.3.63 _sync_case_invalid_arm() . . . 388
7.138.3.64 _sync_stmt() . . . 388
7.138.3.65 _sync_top_level() . . . 388
7.138.3.66 _try_parse_qualified_name() . . . 389

7.138.3.67 from_source() . . . 389
7.138.3.68 parse_module() . . . 389
7.138.4 Member Data Documentation . . . 389
7.138.4.1 _last_token . . . 389
7.138.4.2 _RESERVED_BINARY_OPS . . . 390
7.138.4.3 diagnostics . . . 390
7.138.4.4 eof_aborted . . . 390
7.138.4.5 filename . . . 390
7.138.4.6 index . . . 390
7.138.4.7 tokens . . . 390
7.139 ParseResult Struct Reference . . . 391
7.139.1 Detailed Description . . . 391
7.139.2 Member Data Documentation . . . 391
7.139.2.1 diags . . . 391
7.139.2.2 expr_arena . . . 391
7.139.2.3 pattern_arena . . . 392
7.139.2.4 root_module . . . 392
7.139.2.5 stmt_arena . . . 392
7.140 ParserState Struct Reference . . . 392
7.140.1 Detailed Description . . . 393
7.140.2 Member Data Documentation . . . 393
7.140.2.1 diags . . . 393
7.140.2.2 emitted_lex_errors . . . 393
7.140.2.3 eof_aborted . . . 393
7.140.2.4 expr_arena . . . 393
7.140.2.5 filename . . . 393
7.140.2.6 index . . . 393
7.140.2.7 last_token . . . 393
7.140.2.8 pattern_arena . . . 393
7.140.2.9 stmt_arena . . . 393
7.140.2.10 tokens . . . 394
7.141 l0_ast.Pattern Class Reference . . . 394
7.141.1 Detailed Description . . . 395
7.142 PatternArena Struct Reference . . . 395
7.142.1 Detailed Description . . . 395
7.142.2 Member Data Documentation . . . 396
7.142.2.1 nodes . . . 396
7.143 PatternNode Struct Reference . . . 396
7.143.1 Detailed Description . . . 396
7.143.2 Member Data Documentation . . . 396
7.143.2.1 kind . . . 396
7.143.2.2 module_path . . . 397

7.143.2.3 name . . . 397
7.143.2.4 name_qualifier . . . 397
7.143.2.5 span . . . 397
7.143.2.6 vars . . . 397
7.144 l0_types.PointerType Class Reference . . . 397
7.144.1 Detailed Description . . . 398
7.144.2 Member Data Documentation . . . 398
7.144.2.1 inner . . . 398
7.145 PreparedInput Struct Reference . . . 399
7.145.1 Detailed Description . . . 399
7.145.2 Member Data Documentation . . . 399
7.145.2.1 entry_module . . . 399
7.145.2.2 search_paths . . . 399
7.146 QualifiedNameResult Struct Reference . . . 400
7.146.1 Detailed Description . . . 400
7.146.2 Member Data Documentation . . . 400
7.146.2.1 module_path . . . 400
7.146.2.2 name_qualifier . . . 400
7.146.2.3 name_tok . . . 400
7.147 l0_resolve.ResolveErrorKind Class Reference . . . 401
7.147.1 Detailed Description . . . 401
7.147.2 Member Data Documentation . . . 401
7.147.2.1 AMBIGUOUS_SYMBOL . . . 401
7.147.2.2 MODULE_NOT_IMPORTED . . . 402
7.147.2.3 UNKNOWN_MODULE . . . 402
7.147.2.4 UNKNOWN_SYMBOL . . . 402
7.148 l0_ast.ReturnStmt Class Reference . . . 402
7.148.1 Detailed Description . . . 403
7.148.2 Member Data Documentation . . . 403
7.148.2.1 value . . . 403
7.149 RtFileInfo Struct Reference . . . 404
7.149.1 Detailed Description . . . 404
7.149.2 Member Data Documentation . . . 404
7.149.2.1 exists . . . 404
7.149.2.2 is_dir . . . 404
7.149.2.3 is_file . . . 404
7.149.2.4 mtime_nsec . . . 404
7.149.2.5 mtime_sec . . . 404
7.149.2.6 size . . . 405
7.150 RtTimeParts Struct Reference . . . 405
7.150.1 Detailed Description . . . 405
7.150.2 Member Data Documentation . . . 405

7.150.2.1 nsec . . . 405
7.150.2.2 sec . . . 405
7.151 l0_locals.Scope Class Reference . . . 406
7.151.1 Detailed Description . . . 406
7.151.2 Member Function Documentation . . . 406
7.151.2.1 lookup() . . . 406
7.151.3 Member Data Documentation . . . 407
7.151.3.1 parent . . . 407
7.151.3.2 symbols . . . 407
7.152 Scope Struct Reference . . . 407
7.152.1 Detailed Description . . . 408
7.152.2 Member Data Documentation . . . 408
7.152.2.1 parent . . . 408
7.152.2.2 symbols . . . 408
7.153 l0_scope_context.ScopeContext Class Reference . . . 408
7.153.1 Detailed Description . . . 409
7.153.2 Member Function Documentation . . . 409
7.153.2.1 add_declared() . . . 409
7.153.2.2 add_owned() . . . 410
7.153.2.3 remove_owned() . . . 410
7.153.3 Member Data Documentation . . . 410
7.153.3.1 declared_vars . . . 410
7.153.3.2 owned_vars [1/2] . . . 410
7.153.3.3 owned_vars [2/2] . . . 410
7.153.3.4 parent . . . 411
7.153.3.5 with_cleanup_block . . . 411
7.153.3.6 with_cleanup_in_progress . . . 411
7.153.3.7 with_cleanup_inline . . . 411
7.154 ScopeContext Struct Reference . . . 411
7.154.1 Detailed Description . . . 412
7.154.2 Member Data Documentation . . . 412
7.154.2.1 declared_vars . . . 412
7.154.2.2 owned_vars . . . 412
7.154.2.3 parent . . . 412
7.154.2.4 scope_id . . . 412
7.154.2.5 with_cleanup_block . . . 412
7.154.2.6 with_cleanup_in_progress . . . 412
7.154.2.7 with_cleanup_inline . . . 413
7.155 ScopeSlot Struct Reference . . . 413
7.155.1 Detailed Description . . . 413
7.155.2 Member Data Documentation . . . 414
7.155.2.1 scope . . . 414

7.156 ScopeVar Struct Reference . . . 414
7.156.1 Detailed Description . . . 414
7.156.2 Member Data Documentation . . . 415
7.156.2.1 name . . . 415
7.156.2.2 var_type . . . 415
7.157 l0_signatures.SignatureResolver Class Reference . . . 415
7.157.1 Detailed Description . . . 416
7.157.2 Constructor & Destructor Documentation . . . 416
7.157.2.1 __init__() . . . 416
7.157.3 Member Function Documentation . . . 417
7.157.3.1 _detect_value_type_cycles() . . . 417
7.157.3.2 _emit() . . . 417
7.157.3.3 _enum_type_for_variant_symbol() . . . 417
7.157.3.4 _extract_value_type_dependencies() . . . 417
7.157.3.5 _infer_literal_type() . . . 417
7.157.3.6 _resolve_enum() . . . 418
7.157.3.7 _resolve_func() . . . 418
7.157.3.8 _resolve_let() . . . 418
7.157.3.9 _resolve_module_signatures() . . . 418
7.157.3.10 _resolve_struct() . . . 418
7.157.3.11 _resolve_type_alias() . . . 418
7.157.3.12 _resolve_type_alias_symbol() . . . 419
7.157.3.13 _resolve_type_ref() . . . 419
7.157.3.14 _symbol_is_zero_arg_enum_variant() . . . 419
7.157.3.15 resolve() . . . 420
7.157.4 Member Data Documentation . . . 420
7.157.4.1 cu . . . 420
7.157.4.2 module_envs . . . 420
7.158 SignatureTables Struct Reference . . . 420
7.158.1 Detailed Description . . . 421
7.158.2 Member Data Documentation . . . 421
7.158.2.1 enum_infos . . . 421
7.158.2.2 func_types . . . 421
7.158.2.3 let_types . . . 421
7.158.2.4 struct_infos . . . 421
7.159 l0_driver.SourceEncodingError Class Reference . . . 422
7.159.1 Detailed Description . . . 422
7.159.2 Constructor & Destructor Documentation . . . 423
7.159.2.1 __init__() . . . 423
7.159.3 Member Data Documentation . . . 423
7.159.3.1 message . . . 423
7.159.3.2 path . . . 423

7.160 l0_paths.SourceSearchPaths Class Reference . . . 423
7.160.1 Detailed Description . . . 424
7.160.2 Member Function Documentation . . . 424
7.160.2.1 add_project_root() . . . 424
7.160.2.2 add_system_root() . . . 424
7.160.2.3 module_relpath() . . . 425
7.160.2.4 resolve() . . . 425
7.160.3 Member Data Documentation . . . 425
7.160.3.1 project_roots . . . 425
7.160.3.2 system_roots . . . 426
7.161 SourceSearchPaths Struct Reference . . . 426
7.161.1 Detailed Description . . . 426
7.161.2 Member Data Documentation . . . 426
7.161.2.1 project_roots . . . 426
7.161.2.2 system_roots . . . 426
7.162 l0_ast.Span Class Reference . . . 426
7.162.1 Detailed Description . . . 427
7.162.2 Member Data Documentation . . . 427
7.162.2.1 end_column . . . 427
7.162.2.2 end_line . . . 427
7.162.2.3 start_column . . . 427
7.162.2.4 start_line . . . 427
7.163 Span Struct Reference . . . 427
7.163.1 Detailed Description . . . 428
7.163.2 Member Data Documentation . . . 428
7.163.2.1 end_column . . . 428
7.163.2.2 end_line . . . 428
7.163.2.3 start_column . . . 428
7.163.2.4 start_line . . . 428
7.164 l0_ast.Stmt Class Reference . . . 429
7.164.1 Detailed Description . . . 430
7.165 StmtArena Struct Reference . . . 430
7.165.1 Detailed Description . . . 431
7.165.2 Member Data Documentation . . . 431
7.165.2.1 nodes . . . 431
7.166 l0_expr_types.StmtFlow Class Reference . . . 432
7.166.1 Detailed Description . . . 432
7.166.2 Member Data Documentation . . . 432
7.166.2.1 FALLTHROUGH . . . 432
7.166.2.2 RETURNS . . . 433
7.166.2.3 STOPS . . . 433
7.167 StmtNode Struct Reference . . . 433

7.167.1 Detailed Description . . . 434
7.167.2 Member Data Documentation . . . 434
7.167.2.1 case_arms . . . 434
7.167.2.2 case_else . . . 434
7.167.2.3 expr_a . . . 434
7.167.2.4 expr_b . . . 434
7.167.2.5 ids . . . 434
7.167.2.6 kind . . . 434
7.167.2.7 match_arms . . . 434
7.167.2.8 name . . . 434
7.167.2.9 span . . . 434
7.167.2.10 stmt_a . . . 435
7.167.2.11 stmt_b . . . 435
7.167.2.12 stmt_c . . . 435
7.167.2.13 type_ref . . . 435
7.167.2.14 with_items . . . 435
7.168 StringBuffer Struct Reference . . . 435
7.168.1 Detailed Description . . . 435
7.168.2 Member Data Documentation . . . 435
7.168.2.1 parts . . . 435
7.168.2.2 size . . . 436
7.169 StringIntMap Struct Reference . . . 436
7.169.1 Detailed Description . . . 436
7.169.2 Member Data Documentation . . . 437
7.169.2.1 capacity . . . 437
7.169.2.2 count . . . 437
7.169.2.3 hashes . . . 437
7.169.2.4 keys . . . 437
7.169.2.5 states . . . 437
7.169.2.6 tomb_count . . . 437
7.169.2.7 values . . . 437
7.170 l0_ast.StringLiteral Class Reference . . . 438
7.170.1 Detailed Description . . . 439
7.170.2 Member Data Documentation . . . 439
7.170.2.1 value . . . 439
7.171 StringPtrMap Struct Reference . . . 439
7.171.1 Detailed Description . . . 440
7.171.2 Member Data Documentation . . . 440
7.171.2.1 capacity . . . 440
7.171.2.2 count . . . 440
7.171.2.3 hashes . . . 440
7.171.2.4 keys . . . 440

7.171.2.5 states . . . 440
7.171.2.6 tomb_count . . . 440
7.171.2.7 values . . . 441
7.172 StringSet Struct Reference . . . 441
7.172.1 Detailed Description . . . 441
7.172.2 Member Data Documentation . . . 441
7.172.2.1 capacity . . . 441
7.172.2.2 count . . . 442
7.172.2.3 hashes . . . 442
7.172.2.4 keys . . . 442
7.172.2.5 states . . . 442
7.172.2.6 tomb_count . . . 442
7.173 StringStringLinearMap Struct Reference . . . 442
7.173.1 Detailed Description . . . 443
7.173.2 Member Data Documentation . . . 443
7.173.2.1 base . . . 443
7.174 StringStringLinearMapEntry Struct Reference . . . 443
7.174.1 Detailed Description . . . 443
7.174.2 Member Data Documentation . . . 443
7.174.2.1 key . . . 443
7.174.2.2 value . . . 443
7.175 l0_ast.StructDecl Class Reference . . . 444
7.175.1 Detailed Description . . . 445
7.175.2 Member Data Documentation . . . 445
7.175.2.1 fields . . . 445
7.175.2.2 name . . . 445
7.176 StructDecl Struct Reference . . . 445
7.176.1 Detailed Description . . . 446
7.176.2 Member Data Documentation . . . 446
7.176.2.1 fields . . . 446
7.176.2.2 name . . . 446
7.176.2.3 span . . . 446
7.177 l0_signatures.StructFieldInfo Class Reference . . . 446
7.177.1 Detailed Description . . . 447
7.177.2 Member Data Documentation . . . 447
7.177.2.1 name . . . 447
7.177.2.2 type . . . 447
7.178 StructFieldInfo Struct Reference . . . 447
7.178.1 Detailed Description . . . 448
7.178.2 Member Data Documentation . . . 448
7.178.2.1 field_type . . . 448
7.178.2.2 name . . . 448

7.179 l0_signatures.StructInfo Class Reference . . . 449
7.179.1 Detailed Description . . . 449
7.179.2 Member Data Documentation . . . 449
7.179.2.1 fields . . . 449
7.179.2.2 struct_type . . . 449
7.180 StructInfo Struct Reference . . . 450
7.180.1 Detailed Description . . . 450
7.180.2 Member Data Documentation . . . 450
7.180.2.1 fields . . . 450
7.180.2.2 filename . . . 451
7.180.2.3 key . . . 451
7.180.2.4 module_name . . . 451
7.180.2.5 name . . . 451
7.180.2.6 span . . . 451
7.180.2.7 struct_type . . . 451
7.181 l0_types.StructType Class Reference . . . 451
7.181.1 Detailed Description . . . 452
7.181.2 Member Data Documentation . . . 452
7.181.2.1 module . . . 452
7.181.2.2 name . . . 452
7.182 l0_symbols.Symbol Class Reference . . . 453
7.182.1 Detailed Description . . . 453
7.182.2 Member Data Documentation . . . 453
7.182.2.1 kind . . . 453
7.182.2.2 module . . . 454
7.182.2.3 name . . . 454
7.182.2.4 node . . . 454
7.182.2.5 type . . . 454
7.183 Symbol Struct Reference . . . 454
7.183.1 Detailed Description . . . 455
7.183.2 Member Data Documentation . . . 455
7.183.2.1 decl_ptr . . . 455
7.183.2.2 kind . . . 455
7.183.2.3 module_name . . . 455
7.183.2.4 name . . . 455
7.183.2.5 owner_name . . . 455
7.183.2.6 resolved_type . . . 455
7.183.2.7 span . . . 456
7.184 l0_symbols.SymbolKind Class Reference . . . 456
7.184.1 Detailed Description . . . 457
7.184.2 Member Data Documentation . . . 457
7.184.2.1 ENUM . . . 457

7.184.2.2 ENUM_VARIANT . . . 457
7.184.2.3 FUNC . . . 457
7.184.2.4 LET . . . 457
7.184.2.5 MODULE . . . 457
7.184.2.6 STRUCT . . . 457
7.184.2.7 TYPE_ALIAS . . . 458
7.185 l0_resolve.SymbolResolution Class Reference . . . 458
7.185.1 Detailed Description . . . 458
7.185.2 Member Data Documentation . . . 459
7.185.2.1 ambiguous_modules . . . 459
7.185.2.2 error . . . 459
7.185.2.3 module_name . . . 459
7.185.2.4 name . . . 459
7.185.2.5 symbol . . . 459
7.186 l0_lexer.Token Class Reference . . . 459
7.186.1 Detailed Description . . . 460
7.186.2 Member Function Documentation . . . 460
7.186.2.1 __repr__() . . . 460
7.186.3 Member Data Documentation . . . 460
7.186.3.1 column . . . 460
7.186.3.2 diagnostic . . . 461
7.186.3.3 diagnostics . . . 461
7.186.3.4 kind . . . 461
7.186.3.5 line . . . 461
7.186.3.6 recovery . . . 461
7.186.3.7 text . . . 461
7.187 Token Struct Reference . . . 461
7.187.1 Detailed Description . . . 461
7.187.2 Member Data Documentation . . . 462
7.187.2.1 column . . . 462
7.187.2.2 index . . . 462
7.187.2.3 line . . . 462
7.187.2.4 token_type . . . 462
7.188 l0_lexer.TokenKind Class Reference . . . 462
7.188.1 Detailed Description . . . 464
7.188.2 Member Data Documentation . . . 466
7.188.2.1 AMP . . . 466
7.188.2.2 ANDAND . . . 466
7.188.2.3 ARROW_FUNC . . . 466
7.188.2.4 ARROW_MATCH . . . 466
7.188.2.5 AS . . . 466
7.188.2.6 BANG . . . 466

7.188.2.7 BREAK . . . 466
7.188.2.8 BYTE . . . 467
7.188.2.9 CARET . . . 467
7.188.2.10 CASE . . . 467
7.188.2.11 CLEANUP . . . 467
7.188.2.12 COLON . . . 467
7.188.2.13 COMMA . . . 467
7.188.2.14 CONTINUE . . . 467
7.188.2.15 DOT . . . 467
7.188.2.16 DOUBLE_COLON . . . 467
7.188.2.17 DROP . . . 467
7.188.2.18 ELSE . . . 468
7.188.2.19 ENUM . . . 468
7.188.2.20 EOF . . . 468
7.188.2.21 EQ . . . 468
7.188.2.22 EQEQ . . . 468
7.188.2.23 EXTERN . . . 468
7.188.2.24 FALSE . . . 468
7.188.2.25 FOR . . . 468
7.188.2.26 FUNC . . . 468
7.188.2.27 FUTURE_EXTENSION . . . 468
7.188.2.28 GE . . . 469
7.188.2.29 GT . . . 469
7.188.2.30 IDENT . . . 469
7.188.2.31 IF . . . 469
7.188.2.32 IMPORT . . . 469
7.188.2.33 INT . . . 469
7.188.2.34 LBRACE . . . 469
7.188.2.35 LBRACKET . . . 469
7.188.2.36 LE . . . 469
7.188.2.37 LET . . . 469
7.188.2.38 LEXER_ERROR . . . 470
7.188.2.39 LPAREN . . . 470
7.188.2.40 LSHIFT . . . 470
7.188.2.41 LT . . . 470
7.188.2.42 MATCH . . . 470
7.188.2.43 MINUS . . . 470
7.188.2.44 MODULE . . . 470
7.188.2.45 MODULO . . . 470
7.188.2.46 NE . . . 470
7.188.2.47 NEW . . . 470
7.188.2.48 NULL . . . 471

7.188.2.49 OROR . . . 471
7.188.2.50 PIPE . . . 471
7.188.2.51 PLUS . . . 471
7.188.2.52 QUESTION . . . 471
7.188.2.53 RBRACE . . . 471
7.188.2.54 RBRACKET . . . 471
7.188.2.55 RETURN . . . 471
7.188.2.56 RPAREN . . . 471
7.188.2.57 RSHIFT . . . 471
7.188.2.58 SEMI . . . 472
7.188.2.59 SLASH . . . 472
7.188.2.60 STAR . . . 472
7.188.2.61 STRING . . . 472
7.188.2.62 STRUCT . . . 472
7.188.2.63 TILDE . . . 472
7.188.2.64 TRUE . . . 472
7.188.2.65 TYPE . . . 472
7.188.2.66 UNDERSCORE . . . 472
7.188.2.67 WHILE . . . 472
7.188.2.68 WITH . . . 473
7.189 TokenVector Struct Reference . . . 473
7.189.1 Detailed Description . . . 473
7.189.2 Member Data Documentation . . . 473
7.189.2.1 vec . . . 473
7.190 l0_ast.TopLevelDecl Class Reference . . . 474
7.190.1 Detailed Description . . . 475
7.191 TopLevelDecl Struct Reference . . . 475
7.191.1 Detailed Description . . . 475
7.191.2 Member Data Documentation . . . 475
7.191.2.1 data . . . 475
7.191.2.2 kind . . . 475
7.192 l0_ast.TryExpr Class Reference . . . 476
7.192.1 Detailed Description . . . 477
7.192.2 Member Data Documentation . . . 477
7.192.2.1 expr . . . 477
7.193 l0_types.Type Class Reference . . . 477
7.193.1 Detailed Description . . . 478
7.194 Type Struct Reference . . . 478
7.194.1 Detailed Description . . . 478
7.194.2 Member Data Documentation . . . 479
7.194.2.1 inner . . . 479
7.194.2.2 kind . . . 479

7.194.2.3 module_name . . . 479
7.194.2.4 name . . . 479
7.194.2.5 params . . . 479
7.194.2.6 result . . . 479
7.195 l0_ast.TypeAliasDecl Class Reference . . . 480
7.195.1 Detailed Description . . . 481
7.195.2 Member Data Documentation . . . 481
7.195.2.1 name . . . 481
7.195.2.2 target . . . 481
7.196 TypeAliasDecl Struct Reference . . . 481
7.196.1 Detailed Description . . . 482
7.196.2 Member Data Documentation . . . 482
7.196.2.1 name . . . 482
7.196.2.2 span . . . 482
7.196.2.3 target . . . 482
7.197 l0_ast.TypeExpr Class Reference . . . 482
7.197.1 Detailed Description . . . 483
7.197.2 Member Data Documentation . . . 483
7.197.2.1 type_ref . . . 483
7.198 l0_ast.TypeRef Class Reference . . . 484
7.198.1 Detailed Description . . . 485
7.198.2 Member Data Documentation . . . 485
7.198.2.1 is_nullable . . . 485
7.198.2.2 module_path . . . 485
7.198.2.3 name . . . 485
7.198.2.4 name_qualifier . . . 485
7.198.2.5 pointer_depth . . . 485
7.199 TypeRef Struct Reference . . . 486
7.199.1 Detailed Description . . . 486
7.199.2 Member Data Documentation . . . 486
7.199.2.1 is_nullable . . . 486
7.199.2.2 module_path . . . 486
7.199.2.3 name . . . 486
7.199.2.4 name_qualifier . . . 487
7.199.2.5 pointer_depth . . . 487
7.199.2.6 span . . . 487
7.200 l0_resolve.TypeResolution Class Reference . . . 487
7.200.1 Detailed Description . . . 488
7.200.2 Member Data Documentation . . . 488
7.200.2.1 ambiguous_modules . . . 488
7.200.2.2 error . . . 488
7.200.2.3 module_name . . . 488

7.200.2.4 name . . . 488
7.200.2.5 symbol . . . 488
7.200.2.6 type . . . 488
7.201 l0_resolve.TypeResolveErrorKind Class Reference . . . 489
7.201.1 Detailed Description . . . 489
7.201.2 Member Data Documentation . . . 490
7.201.2.1 AMBIGUOUS_TYPE . . . 490
7.201.2.2 INVALID_NULLABLE_VOID . . . 490
7.201.2.3 MODULE_NOT_IMPORTED . . . 490
7.201.2.4 NOT_A_TYPE . . . 490
7.201.2.5 UNKNOWN_MODULE . . . 490
7.201.2.6 UNKNOWN_TYPE . . . 490
7.201.2.7 UNRESOLVED_ALIAS . . . 490
7.201.2.8 VARIANT_AS_TYPE . . . 490
7.202 l0_ast.UnaryOp Class Reference . . . 491
7.202.1 Detailed Description . . . 492
7.202.2 Member Data Documentation . . . 492
7.202.2.1 op . . . 492
7.202.2.2 operand . . . 492
7.203 Unit Struct Reference . . . 492
7.203.1 Detailed Description . . . 492
7.204 l0_ast.VariantPattern Class Reference . . . 493
7.204.1 Detailed Description . . . 494
7.204.2 Member Data Documentation . . . 494
7.204.2.1 module_path . . . 494
7.204.2.2 name . . . 494
7.204.2.3 name_qualifier . . . 494
7.204.2.4 vars . . . 494
7.205 l0_ast.VarRef Class Reference . . . 495
7.205.1 Detailed Description . . . 496
7.205.2 Member Data Documentation . . . 496
7.205.2.1 module_path . . . 496
7.205.2.2 name . . . 496
7.205.2.3 name_qualifier . . . 496
7.206 l0_analysis.VarRefResolution Class Reference . . . 497
7.206.1 Detailed Description . . . 497
7.206.2 Member Data Documentation . . . 497
7.206.2.1 LOCAL . . . 497
7.206.2.2 MODULE . . . 498
7.207 VectorBase Struct Reference . . . 498
7.207.1 Detailed Description . . . 498
7.207.2 Member Data Documentation . . . 498

7.207.2.1 arr . . . 498
7.207.2.2 length . . . 499
7.208 WallTime Struct Reference . . . 499
7.208.1 Detailed Description . . . 499
7.208.2 Member Data Documentation . . . 499
7.208.2.1 nsec . . . 499
7.208.2.2 sec . . . 499
7.209 l0_ast.WhileStmt Class Reference . . . 500
7.209.1 Detailed Description . . . 501
7.209.2 Member Data Documentation . . . 501
7.209.2.1 body . . . 501
7.209.2.2 cond . . . 501
7.210 l0_ast.WildcardPattern Class Reference . . . 502
7.210.1 Detailed Description . . . 503
7.211 l0_ast.WithItem Class Reference . . . 503
7.211.1 Detailed Description . . . 504
7.211.2 Member Data Documentation . . . 504
7.211.2.1 cleanup . . . 504
7.211.2.2 init . . . 504
7.212 WithItem Struct Reference . . . 504
7.212.1 Detailed Description . . . 505
7.212.2 Member Data Documentation . . . 505
7.212.2.1 cleanup_stmt . . . 505
7.212.2.2 has_cleanup . . . 505
7.212.2.3 init . . . 505
7.212.2.4 span . . . 505
7.213 l0_ast.WithStmt Class Reference . . . 506
7.213.1 Detailed Description . . . 507
7.213.2 Member Data Documentation . . . 507
7.213.2.1 body . . . 507
7.213.2.2 cleanup_body . . . 507
7.213.2.3 items . . . 507

8 File Documentation **509**

8.1 compiler/shared/l0/stdlib/std/array.l0 File Reference . . . 509
8.1.1 Function Documentation . . . 510
8.1.1.1 arr_check() . . . 510
8.1.1.2 arr_create() . . . 510
8.1.1.3 arr_free() . . . 510
8.1.1.4 arr_get() . . . 511
8.1.1.5 arr_resize() . . . 511
8.1.1.6 arr_zap() . . . 511

8.1.1.7 ba_capacity() . . . 511
8.1.1.8 ba_create() . . . 512
8.1.1.9 ba_free() . . . 512
8.1.1.10 ba_get() . . . 512
8.1.1.11 ba_set() . . . 512
8.1.1.12 ba_zap() . . . 514
8.2 compiler/shared/l0/stdlib/std/assert.l0 File Reference . . . 514
8.2.1 Function Documentation . . . 514
8.2.1.1 assert() . . . 514
8.3 compiler/shared/l0/stdlib/std/fs.l0 File Reference . . . 515
8.3.1 Function Documentation . . . 515
8.3.1.1 _fs_from_rt() . . . 515
8.3.1.2 delete_file() . . . 516
8.3.1.3 exists() . . . 516
8.3.1.4 file_size() . . . 516
8.3.1.5 is_dir() . . . 517
8.3.1.6 is_file() . . . 517
8.3.1.7 mtime_sec() . . . 517
8.3.1.8 read_file() . . . 517
8.3.1.9 stat() . . . 518
8.3.1.10 write_file() . . . 518
8.4 compiler/shared/l0/stdlib/std/hashmap.l0 File Reference . . . 518
8.4.1 Function Documentation . . . 521
8.4.1.1 _hm_slot() . . . 521
8.4.1.2 _sim_alloc() . . . 522
8.4.1.3 _sim_find() . . . 522
8.4.1.4 _sim_find_insert() . . . 522
8.4.1.5 _sim_get_hash() . . . 523
8.4.1.6 _sim_get_key() . . . 523
8.4.1.7 _sim_get_state() . . . 523
8.4.1.8 _sim_get_value() . . . 523
8.4.1.9 _sim_needs_grow() . . . 524
8.4.1.10 _sim_rehash() . . . 524
8.4.1.11 _sim_release_key() . . . 524
8.4.1.12 _sim_set_hash() . . . 525
8.4.1.13 _sim_set_key_retain() . . . 525
8.4.1.14 _sim_set_state() . . . 525
8.4.1.15 _sim_set_value() . . . 525
8.4.1.16 _spm_alloc() . . . 526
8.4.1.17 _spm_find() . . . 526
8.4.1.18 _spm_find_insert() . . . 526
8.4.1.19 _spm_get_hash() . . . 527

8.4.1.20 _spm_get_key() . . . 527
8.4.1.21 _spm_get_state() . . . 527
8.4.1.22 _spm_get_value() . . . 527
8.4.1.23 _spm_needs_grow() . . . 528
8.4.1.24 _spm_rehash() . . . 528
8.4.1.25 _spm_release_key() . . . 528
8.4.1.26 _spm_set_hash() . . . 529
8.4.1.27 _spm_set_key_retain() . . . 529
8.4.1.28 _spm_set_state() . . . 529
8.4.1.29 _spm_set_value() . . . 529
8.4.1.30 sim_capacity() . . . 530
8.4.1.31 sim_clear() . . . 530
8.4.1.32 sim_create() . . . 530
8.4.1.33 sim_create_with_capacity() . . . 530
8.4.1.34 sim_free() . . . 531
8.4.1.35 sim_get() . . . 531
8.4.1.36 sim_has() . . . 531
8.4.1.37 sim_keys() . . . 532
8.4.1.38 sim_put() . . . 532
8.4.1.39 sim_remove() . . . 532
8.4.1.40 sim_size() . . . 533
8.4.1.41 sim_slot_key() . . . 533
8.4.1.42 sim_slot_occupied() . . . 533
8.4.1.43 sim_slot_value() . . . 533
8.4.1.44 spm_capacity() . . . 534
8.4.1.45 spm_clear() . . . 534
8.4.1.46 spm_create() . . . 534
8.4.1.47 spm_create_with_capacity() . . . 535
8.4.1.48 spm_free() . . . 535
8.4.1.49 spm_get() . . . 535
8.4.1.50 spm_has() . . . 535
8.4.1.51 spm_keys() . . . 536
8.4.1.52 spm_put() . . . 536
8.4.1.53 spm_remove() . . . 536
8.4.1.54 spm_size() . . . 537
8.4.1.55 spm_slot_key() . . . 537
8.4.1.56 spm_slot_occupied() . . . 537
8.4.1.57 spm_slot_value() . . . 538
8.4.2 Variable Documentation . . . 538
8.4.2.1 HM_DEFAULT_CAP . . . 538
8.4.2.2 HM_EMPTY . . . 538
8.4.2.3 HM_OCCUPIED . . . 538

8.4.2.4 HM_TOMBSTONE . . . 538
8.5 compiler/shared/l0/stdlib/std/hashset.l0 File Reference . . . 539
8.5.1 Function Documentation . . . 540
8.5.1.1 _hs_slot() . . . 540
8.5.1.2 _ss_alloc() . . . 540
8.5.1.3 _ss_find() . . . 542
8.5.1.4 _ss_find_insert() . . . 542
8.5.1.5 _ss_get_hash() . . . 542
8.5.1.6 _ss_get_key() . . . 543
8.5.1.7 _ss_get_state() . . . 543
8.5.1.8 _ss_needs_grow() . . . 543
8.5.1.9 _ss_rehash() . . . 544
8.5.1.10 _ss_release_key() . . . 544
8.5.1.11 _ss_set_hash() . . . 544
8.5.1.12 _ss_set_key_retain() . . . 544
8.5.1.13 _ss_set_state() . . . 546
8.5.1.14 ss_add() . . . 546
8.5.1.15 ss_capacity() . . . 546
8.5.1.16 ss_clear() . . . 547
8.5.1.17 ss_create() . . . 547
8.5.1.18 ss_create_with_capacity() . . . 547
8.5.1.19 ss_free() . . . 547
8.5.1.20 ss_has() . . . 548
8.5.1.21 ss_remove() . . . 548
8.5.1.22 ss_size() . . . 548
8.5.1.23 ss_slot_key() . . . 549
8.5.1.24 ss_slot_occupied() . . . 549
8.5.1.25 ss_to_vector() . . . 549
8.5.2 Variable Documentation . . . 550
8.5.2.1 HS_DEFAULT_CAP . . . 550
8.5.2.2 HS_EMPTY . . . 550
8.5.2.3 HS_OCCUPIED . . . 550
8.5.2.4 HS_TOMBSTONE . . . 550
8.6 compiler/shared/l0/stdlib/std/integer.l0 File Reference . . . 550
8.6.1 Function Documentation . . . 551
8.6.1.1 abs() . . . 551
8.6.1.2 align_down() . . . 552
8.6.1.3 align_up() . . . 552
8.6.1.4 checked_add_int() . . . 552
8.6.1.5 checked_mul_int() . . . 553
8.6.1.6 checked_sub_int() . . . 553
8.6.1.7 clamp() . . . 553

8.6.1.8 div_ceil() . . . 554
8.6.1.9 div_floor() . . . 554
8.6.1.10 ediv() . . . 554
8.6.1.11 emod() . . . 555
8.6.1.12 gcd() . . . 555
8.6.1.13 int_max() . . . 555
8.6.1.14 int_min() . . . 556
8.6.1.15 is_aligned() . . . 556
8.6.1.16 is_even() . . . 556
8.6.1.17 is_multiple() . . . 556
8.6.1.18 is_odd() . . . 557
8.6.1.19 isqrt() . . . 557
8.6.1.20 lcm() . . . 557
8.6.1.21 max() . . . 558
8.6.1.22 min() . . . 558
8.6.1.23 pow() . . . 558
8.6.1.24 sign() . . . 559
8.7 compiler/shared/l0/stdlib/std/io.l0 File Reference . . . 559
8.7.1 Function Documentation . . . 561
8.7.1.1 _io_check_byte_range() . . . 561
8.7.1.2 err_print_b() . . . 562
8.7.1.3 err_print_bb() . . . 562
8.7.1.4 err_print_bi() . . . 562
8.7.1.5 err_print_bs() . . . 562
8.7.1.6 err_print_i() . . . 563
8.7.1.7 err_print_ib() . . . 563
8.7.1.8 err_print_ii() . . . 563
8.7.1.9 err_print_is() . . . 563
8.7.1.10 err_print_s() . . . 564
8.7.1.11 err_print_sb() . . . 564
8.7.1.12 err_print_si() . . . 564
8.7.1.13 err_print_ss() . . . 564
8.7.1.14 err_printl() . . . 565
8.7.1.15 err_printl_b() . . . 565
8.7.1.16 err_printl_i() . . . 565
8.7.1.17 err_printl_s() . . . 565
8.7.1.18 flush_stderr() . . . 565
8.7.1.19 flush_stdout() . . . 565
8.7.1.20 print_b() . . . 566
8.7.1.21 print_bb() . . . 566
8.7.1.22 print_bi() . . . 566
8.7.1.23 print_bs() . . . 566

8.7.1.24 print_i() . . . 567
8.7.1.25 print_ib() . . . 567
8.7.1.26 print_ii() . . . 567
8.7.1.27 print_is() . . . 567
8.7.1.28 print_s() . . . 568
8.7.1.29 print_sb() . . . 568
8.7.1.30 print_si() . . . 568
8.7.1.31 print_ss() . . . 568
8.7.1.32 printl() . . . 569
8.7.1.33 printl_b() . . . 569
8.7.1.34 printl_bb() . . . 569
8.7.1.35 printl_bi() . . . 569
8.7.1.36 printl_bs() . . . 569
8.7.1.37 printl_i() . . . 570
8.7.1.38 printl_ib() . . . 570
8.7.1.39 printl_ii() . . . 570
8.7.1.40 printl_is() . . . 570
8.7.1.41 printl_s() . . . 571
8.7.1.42 printl_sb() . . . 571
8.7.1.43 printl_si() . . . 571
8.7.1.44 printl_ss() . . . 571
8.7.1.45 read_char() . . . 572
8.7.1.46 read_char_or_eof() . . . 572
8.7.1.47 read_line() . . . 572
8.7.1.48 read_stdin_some() . . . 572
8.7.1.49 write_stderr_all() . . . 573
8.7.1.50 write_stderr_some() . . . 573
8.7.1.51 write_stdout_all() . . . 573
8.7.1.52 write_stdout_some() . . . 574
8.8 compiler/shared/l0/stdlib/std/linear_map.l0 File Reference . . . 574
8.8.1 Function Documentation . . . 576
8.8.1.1 _islm_entry_by_index() . . . 576
8.8.1.2 _islm_find_by_key() . . . 576
8.8.1.3 _lm_entry_by_hk_key() . . . 577
8.8.1.4 _lm_entry_by_hv_value() . . . 577
8.8.1.5 _lm_entry_by_index() . . . 577
8.8.1.6 _lm_grow() . . . 578
8.8.1.7 _lm_key_by_entry() . . . 578
8.8.1.8 _lm_new_entry() . . . 578
8.8.1.9 _lm_set_entry() . . . 579
8.8.1.10 _lm_value_by_entry() . . . 579
8.8.1.11 _sslm_entry_by_index() . . . 579

8.8.1.12 _sslm_find_by_key() . . . 580
8.8.1.13 islm_contains() . . . 580
8.8.1.14 islm_create() . . . 580
8.8.1.15 islm_free() . . . 581
8.8.1.16 islm_get() . . . 581
8.8.1.17 islm_key_at() . . . 581
8.8.1.18 islm_len() . . . 581
8.8.1.19 islm_remove() . . . 582
8.8.1.20 islm_set() . . . 582
8.8.1.21 islm_value_at() . . . 582
8.8.1.22 lm_contains_key() . . . 583
8.8.1.23 lm_contains_value() . . . 583
8.8.1.24 lm_create() . . . 583
8.8.1.25 lm_free() . . . 584
8.8.1.26 lm_get() . . . 584
8.8.1.27 lm_len() . . . 584
8.8.1.28 lm_remove() . . . 584
8.8.1.29 lm_set() . . . 586
8.8.1.30 sslm_contains() . . . 586
8.8.1.31 sslm_create() . . . 586
8.8.1.32 sslm_free() . . . 587
8.8.1.33 sslm_get() . . . 587
8.8.1.34 sslm_key_at() . . . 587
8.8.1.35 sslm_len() . . . 587
8.8.1.36 sslm_remove() . . . 588
8.8.1.37 sslm_set() . . . 588
8.8.1.38 sslm_value_at() . . . 588
8.9 compiler/shared/l0/stdlib/std/optional.l0 File Reference . . . 589
8.9.1 Function Documentation . . . 589
8.9.1.1 expect_b() . . . 589
8.9.1.2 expect_i() . . . 589
8.9.1.3 expect_s() . . . 590
8.9.1.4 unwrap_or_b() . . . 590
8.9.1.5 unwrap_or_i() . . . 590
8.9.1.6 unwrap_or_s() . . . 591
8.10 compiler/shared/l0/stdlib/std/rand.l0 File Reference . . . 591
8.10.1 Function Documentation . . . 592
8.10.1.1 rand_bool() . . . 592
8.10.1.2 rand_dice() . . . 592
8.10.1.3 rand_int() . . . 592
8.10.1.4 rand_int_range() . . . 592
8.10.1.5 rand_seed() . . . 593

8.11 compiler/shared/l0/stdlib/std/string.l0 File Reference . . . 593
8.11.1 Function Documentation . . . 594
8.11.1.1 byte_to_s() . . . 594
8.11.1.2 bytes_to_s() . . . 595
8.11.1.3 char_at_s() . . . 595
8.11.1.4 cmp_s() . . . 595
8.11.1.5 concat_s() . . . 596
8.11.1.6 contains_s() . . . 596
8.11.1.7 digit_to_char() . . . 596
8.11.1.8 ends_with_s() . . . 597
8.11.1.9 eq_s() . . . 597
8.11.1.10 find_from_s() . . . 597
8.11.1.11 find_last_s() . . . 598
8.11.1.12 find_s() . . . 598
8.11.1.13 is_alnum() . . . 598
8.11.1.14 is_alpha() . . . 599
8.11.1.15 is_digit() . . . 599
8.11.1.16 is_digit_base() . . . 599
8.11.1.17 is_empty_s() . . . 600
8.11.1.18 is_space() . . . 600
8.11.1.19 len_s() . . . 600
8.11.1.20 slice_s() . . . 600
8.11.1.21 starts_with_s() . . . 601
8.11.1.22 to_digit() . . . 601
8.11.1.23 to_digit_base() . . . 601
8.11.1.24 to_lower() . . . 602
8.11.1.25 to_upper() . . . 602
8.11.1.26 trim_s() . . . 602
8.12 compiler/shared/l0/stdlib/std/system.l0 File Reference . . . 603
8.12.1 Function Documentation . . . 603
8.12.1.1 abort() . . . 603
8.12.1.2 argc() . . . 603
8.12.1.3 argv() . . . 604
8.12.1.4 env_get() . . . 604
8.12.1.5 errno() . . . 604
8.12.1.6 exit() . . . 604
8.12.1.7 get_pid() . . . 605
8.12.1.8 system() . . . 605
8.13 compiler/shared/l0/stdlib/std/text.l0 File Reference . . . 605
8.13.1 Function Documentation . . . 607
8.13.1.1 _line_strip_trailing_cr() . . . 607
8.13.1.2 bool_to_string() . . . 608

8.13.1.3 byte_to_string() . . . 608
8.13.1.4 byte_to_string_base() . . . 608
8.13.1.5 cb_append() . . . 608
8.13.1.6 cb_append_int() . . . 609
8.13.1.7 cb_append_s() . . . 609
8.13.1.8 cb_append_slice() . . . 609
8.13.1.9 cb_capacity() . . . 609
8.13.1.10 cb_clear() . . . 610
8.13.1.11 cb_create() . . . 610
8.13.1.12 cb_free() . . . 610
8.13.1.13 cb_reserve() . . . 610
8.13.1.14 cb_reverse() . . . 611
8.13.1.15 cb_size() . . . 611
8.13.1.16 cb_to_string() . . . 611
8.13.1.17 int_to_bin_string() . . . 611
8.13.1.18 int_to_hex_string() . . . 612
8.13.1.19 int_to_string() . . . 612
8.13.1.20 int_to_string_base() . . . 612
8.13.1.21 join_s() . . . 613
8.13.1.22 lines_s() . . . 613
8.13.1.23 repeat_s() . . . 613
8.13.1.24 replace_s() . . . 614
8.13.1.25 reverse_s() . . . 614
8.13.1.26 sb_append() . . . 614
8.13.1.27 sb_append_byte() . . . 615
8.13.1.28 sb_append_int() . . . 615
8.13.1.29 sb_create() . . . 615
8.13.1.30 sb_free() . . . 615
8.13.1.31 sb_size() . . . 616
8.13.1.32 sb_to_string() . . . 616
8.13.1.33 split_s() . . . 616
8.13.1.34 string_to_bool() . . . 617
8.13.1.35 string_to_byte() . . . 617
8.13.1.36 string_to_byte_base() . . . 617
8.13.1.37 string_to_int() . . . 618
8.13.1.38 string_to_int_base() . . . 618
8.13.1.39 to_lower_s() . . . 618
8.13.1.40 to_upper_s() . . . 619
8.14 compiler/shared/l0/stdlib/std/time.l0 File Reference . . . 619
8.14.1 Function Documentation . . . 620
8.14.1.1 civil_from_unix_days() . . . 620
8.14.1.2 datetime_from_unix_sec() . . . 620

8.14.1.3 day_of_year() . . . 621
8.14.1.4 days_in_month() . . . 621
8.14.1.5 is_leap_year() . . . 621
8.14.1.6 is_valid_nsec() . . . 622
8.14.1.7 local_now_datetime() . . . 622
8.14.1.8 monotonic_diff() . . . 622
8.14.1.9 monotonic_now() . . . 623
8.14.1.10 monotonic_supported() . . . 623
8.14.1.11 utc_now_datetime() . . . 623
8.14.1.12 wall_now() . . . 623
8.14.1.13 wall_to_local_datetime() . . . 623
8.14.1.14 wall_to_utc_datetime() . . . 624
8.15 compiler/shared/l0/stdlib/std/unit.l0 File Reference . . . 624
8.15.1 Function Documentation . . . 624
8.15.1.1 present() . . . 624
8.15.1.2 unit() . . . 625
8.16 compiler/shared/l0/stdlib/std/vector.l0 File Reference . . . 625
8.16.1 Function Documentation . . . 626
8.16.1.1 _sv_less() . . . 626
8.16.1.2 sv_capacity() . . . 627
8.16.1.3 sv_clear() . . . 627
8.16.1.4 sv_create() . . . 627
8.16.1.5 sv_free() . . . 627
8.16.1.6 sv_get() . . . 628
8.16.1.7 sv_push() . . . 628
8.16.1.8 sv_size() . . . 628
8.16.1.9 sv_sort() . . . 629
8.16.1.10 vec_capacity() . . . 629
8.16.1.11 vec_check() . . . 629
8.16.1.12 vec_clear() . . . 629
8.16.1.13 vec_create() . . . 630
8.16.1.14 vec_free() . . . 630
8.16.1.15 vec_get() . . . 630
8.16.1.16 vec_grow() . . . 630
8.16.1.17 vec_push() . . . 631
8.16.1.18 vec_push_bool() . . . 631
8.16.1.19 vec_push_byte() . . . 631
8.16.1.20 vec_push_bytes() . . . 631
8.16.1.21 vec_push_int() . . . 632
8.16.1.22 vec_push_ptr() . . . 632
8.16.1.23 vec_reserve() . . . 632
8.16.1.24 vec_size() . . . 633

8.16.1.25 vec_zap() . . . 633
8.16.1.26 vi_sort() . . . 633
8.16.2 Variable Documentation . . . 633
8.16.2.1 StringVector . . . 633
8.17 compiler/shared/l0/stdlib/sys/hash.l0 File Reference . . . 633
8.17.1 Function Documentation . . . 634
8.17.1.1 rt_hash_bool() . . . 634
8.17.1.2 rt_hash_byte() . . . 634
8.17.1.3 rt_hash_data() . . . 635
8.17.1.4 rt_hash_int() . . . 635
8.17.1.5 rt_hash_opt_bool() . . . 635
8.17.1.6 rt_hash_opt_byte() . . . 636
8.17.1.7 rt_hash_opt_int() . . . 636
8.17.1.8 rt_hash_opt_ptr() . . . 636
8.17.1.9 rt_hash_opt_string() . . . 637
8.17.1.10 rt_hash_ptr() . . . 637
8.17.1.11 rt_hash_string() . . . 637
8.18 compiler/shared/l0/stdlib/sys/memory.l0 File Reference . . . 638
8.18.1 Function Documentation . . . 638
8.18.1.1 rt_alloc() . . . 638
8.18.1.2 rt_array_element() . . . 639
8.18.1.3 rt_calloc() . . . 639
8.18.1.4 rt_free() . . . 639
8.18.1.5 rt_memcmp() . . . 640
8.18.1.6 rt_memcpy() . . . 640
8.18.1.7 rt_memset() . . . 640
8.18.1.8 rt_realloc() . . . 641
8.18.1.9 rt_register_foreign() . . . 641
8.18.1.10 rt_stderr_write() . . . 641
8.18.1.11 rt_stdin_read() . . . 642
8.18.1.12 rt_stdout_write() . . . 642
8.18.1.13 rt_unregister_foreign() . . . 642
8.19 compiler/shared/l0/stdlib/sys/rt.l0 File Reference . . . 643
8.19.1 Function Documentation . . . 645
8.19.1.1 rt_abort() . . . 645
8.19.1.2 rt_delete_file() . . . 645
8.19.1.3 rt_errno() . . . 645
8.19.1.4 rt_exit() . . . 645
8.19.1.5 rt_file_info() . . . 646
8.19.1.6 rt_flush_stderr() . . . 646
8.19.1.7 rt_flush_stdout() . . . 646
8.19.1.8 rt_get_argc() . . . 646

8.19.1.9 rt_get_argv() . . . 646
8.19.1.10 rt_get_env_var() . . . 647
8.19.1.11 rt_get_pid() . . . 647
8.19.1.12 rt_print() . . . 647
8.19.1.13 rt_print_bool() . . . 647
8.19.1.14 rt_print_bool_stderr() . . . 648
8.19.1.15 rt_print_int() . . . 648
8.19.1.16 rt_print_int_stderr() . . . 648
8.19.1.17 rt_print_stderr() . . . 648
8.19.1.18 rt_println() . . . 649
8.19.1.19 rt_println_stderr() . . . 649
8.19.1.20 rt_rand() . . . 649
8.19.1.21 rt_read_char() . . . 649
8.19.1.22 rt_read_file_all() . . . 649
8.19.1.23 rt_read_line() . . . 650
8.19.1.24 rt_srand() . . . 650
8.19.1.25 rt_string_bytes_ptr() . . . 650
8.19.1.26 rt_string_compare() . . . 650
8.19.1.27 rt_string_concat() . . . 651
8.19.1.28 rt_string_equals() . . . 651
8.19.1.29 rt_string_from_byte() . . . 651
8.19.1.30 rt_string_from_byte_array() . . . 652
8.19.1.31 rt_string_get() . . . 652
8.19.1.32 rt_string_release() . . . 652
8.19.1.33 rt_string_retain() . . . 653
8.19.1.34 rt_string_slice() . . . 653
8.19.1.35 rt_strlen() . . . 653
8.19.1.36 rt_system() . . . 654
8.19.1.37 rt_time_local_is_dst() . . . 654
8.19.1.38 rt_time_local_offset_sec() . . . 654
8.19.1.39 rt_time_monotonic() . . . 654
8.19.1.40 rt_time_monotonic_supported() . . . 655
8.19.1.41 rt_time_unix() . . . 655
8.19.1.42 rt_write_file_all() . . . 655
8.20 compiler/shared/runtime/dea_siphash.h File Reference . . . 656
8.20.1 Detailed Description . . . 657
8.20.2 Function Documentation . . . 657
8.20.2.1 siphash13() . . . 657
8.20.2.2 siphash13_tag8_bf() . . . 657
8.20.2.3 siphash13_tag8_bytes() . . . 658
8.20.2.4 siphash24() . . . 658
8.20.2.5 siphash24_tag8_bf() . . . 659

8.20.2.6 siphash24_tag8_bytes() . . . 659
8.21 dea_siphash.h . . . 660
8.22 compiler/shared/runtime/l0_runtime.h File Reference . . . 662
8.22.1 Detailed Description . . . 671
8.22.2 Macro Definition Documentation . . . 671
8.22.2.1 _L0_TAG_ENUM . . . 671
8.22.2.2 _L0_TAG_OPT . . . 671
8.22.2.3 _L0_TAG_PTR . . . 671
8.22.2.4 _L0_TAG_STRUCT . . . 672
8.22.2.5 _RT_ACCESS_READ . . . 672
8.22.2.6 _RT_ACCESS_UNTRACKED_OK . . . 672
8.22.2.7 _RT_ACCESS_WRITE . . . 672
8.22.2.8 _RT_ALIGNOF . . . 672
8.22.2.9 _RT_ALLOC_INIT_CAP . . . 672
8.22.2.10 _RT_ALLOC_LIVE . . . 672
8.22.2.11 _RT_ALLOC_POOLED . . . 672
8.22.2.12 _RT_ALLOC_QUARANTINED . . . 672
8.22.2.13 _RT_ALLOC_TOMBSTONE . . . 672
8.22.2.14 _RT_MEM_ARC . . . 673
8.22.2.15 _RT_MEM_FOREIGN . . . 673
8.22.2.16 _RT_MEM_NEW . . . 673
8.22.2.17 _RT_MEM_RAW . . . 673
8.22.2.18 _RT_MEM_STATIC . . . 673
8.22.2.19 _RT_QUARANTINE_MAX_BYTES . . . 673
8.22.2.20 _RT_QUARANTINE_MAX_COUNT . . . 673
8.22.2.21 _rt_rec_cold . . . 673
8.22.2.22 _RT_REC_POOL_CHUNK . . . 673
8.22.2.23 _RT_TRACE_ARC . . . 673
8.22.2.24 _RT_TRACE_ARC_LOC . . . 674
8.22.2.25 _RT_TRACE_MEM . . . 674
8.22.2.26 _RT_TRACE_MEM_LOC . . . 674
8.22.2.27 DEA_RUNTIME_FCLOSE . . . 674
8.22.2.28 DEA_RUNTIME_FILE_INFO_STAT . . . 674
8.22.2.29 DEA_RUNTIME_REMOVE . . . 674
8.22.2.30 L0_DEFINED_l0_sys_rt_RtFileInfo . . . 674
8.22.2.31 L0_DEFINED_l0_sys_rt_RtTimeParts . . . 674
8.22.2.32 L0_OPT_BOOL_DEFINED . . . 674
8.22.2.33 L0_OPT_BYTE_DEFINED . . . 675
8.22.2.34 L0_OPT_INT_DEFINED . . . 675
8.22.2.35 L0_OPT_STRING_DEFINED . . . 675
8.22.2.36 L0_STRING_CONST . . . 675
8.22.2.37 L0_STRING_K_HEAP . . . 675

8.22.2.38 L0_STRING_K_STATIC . . . 675
8.22.2.39 L0_UNREACHABLE . . . 675
8.22.3 Typedef Documentation . . . 675
8.22.3.1 _rt_alloc_record . . . 675
8.22.3.2 _rt_alloc_record_cold . . . 675
8.22.3.3 _rt_alloc_record_size_check . . . 676
8.22.3.4 _rt_ptr_site . . . 676
8.22.3.5 _rt_siphash_key_t . . . 676
8.22.3.6 _rt_siphash_tag8_t . . . 676
8.22.3.7 l0_bool . . . 676
8.22.3.8 l0_byte . . . 676
8.22.3.9 l0_double . . . 676
8.22.3.10 l0_float . . . 676
8.22.3.11 l0_int . . . 676
8.22.3.12 l0_long . . . 676
8.22.3.13 l0_short . . . 677
8.22.3.14 l0_tiny . . . 677
8.22.3.15 l0_uint . . . 677
8.22.3.16 l0_ulong . . . 677
8.22.3.17 l0_ushort . . . 677
8.22.4 Function Documentation . . . 677
8.22.4.1 _rt_alloc_hash() . . . 677
8.22.4.2 _rt_alloc_obj() . . . 677
8.22.4.3 _rt_alloc_string() . . . 677
8.22.4.4 _rt_alloc_table_insert() . . . 678
8.22.4.5 _rt_alloc_table_lookup() . . . 678
8.22.4.6 _rt_alloc_table_rehash() . . . 678
8.22.4.7 _rt_alloc_table_remove() . . . 678
8.22.4.8 _rt_alloc_tree_find_containing() . . . 678
8.22.4.9 _rt_alloc_tree_glb() . . . 678
8.22.4.10 _rt_alloc_tree_insert() . . . 679
8.22.4.11 _rt_alloc_tree_lub() . . . 679
8.22.4.12 _rt_alloc_tree_remove() . . . 679
8.22.4.13 _rt_check_index_ptr_site() . . . 679
8.22.4.14 _rt_check_index_ptr_site_slow() . . . 680
8.22.4.15 _rt_check_ptr_align() . . . 680
8.22.4.16 _rt_check_ptr_site() . . . 680
8.22.4.17 _rt_check_ptr_site_slow() . . . 681
8.22.4.18 _rt_check_record_writeable() . . . 681
8.22.4.19 _rt_drop_begin_impl() . . . 681
8.22.4.20 _rt_drop_finish_impl() . . . 681
8.22.4.21 _rt_evict_quarantine() . . . 681

8.22.4.22 _rt_fmix32() . . . 681
8.22.4.23 _rt_fold_u64_to_u32_fmix() . . . 682
8.22.4.24 _rt_free_string() . . . 682
8.22.4.25 _rt_hash_bool() . . . 682
8.22.4.26 _rt_hash_byte() . . . 683
8.22.4.27 _rt_hash_data() . . . 683
8.22.4.28 _rt_hash_int() . . . 683
8.22.4.29 _rt_hash_string() . . . 684
8.22.4.30 _rt_hash_tag8() . . . 684
8.22.4.31 _rt_iadd() . . . 685
8.22.4.32 _rt_idiv() . . . 685
8.22.4.33 _rt_imod() . . . 685
8.22.4.34 _rt_imul() . . . 686
8.22.4.35 _rt_index_access_mode() . . . 686
8.22.4.36 _rt_index_allows_untracked() . . . 686
8.22.4.37 _rt_index_element_size() . . . 686
8.22.4.38 _rt_index_offset() . . . 686
8.22.4.39 _rt_init_args() . . . 687
8.22.4.40 _rt_init_heap_string() . . . 687
8.22.4.41 _rt_isub() . . . 687
8.22.4.42 _rt_l0_string_from_const_literal() . . . 688
8.22.4.43 _rt_narrow_l0_byte() . . . 688
8.22.4.44 _rt_new_l0_string() . . . 688
8.22.4.45 _rt_panic() . . . 689
8.22.4.46 _rt_panic_fmt() . . . 689
8.22.4.47 _rt_panic_invalid_access() . . . 689
8.22.4.48 _rt_panic_invalid_drop() . . . 689
8.22.4.49 _rt_panic_invalid_release() . . . 690
8.22.4.50 _rt_pid_to_l0_int() . . . 690
8.22.4.51 _rt_print() . . . 690
8.22.4.52 _rt_promote_new_alloc() . . . 690
8.22.4.53 _rt_ptr_is_aligned() . . . 690
8.22.4.54 _rt_ptr_site_store() . . . 691
8.22.4.55 _rt_quarantine_alloc_record() . . . 691
8.22.4.56 _rt_range_contains() . . . 691
8.22.4.57 _rt_realloc_string() . . . 691
8.22.4.58 _rt_realloc_tracked_alloc() . . . 691
8.22.4.59 _rt_rec_new() . . . 692
8.22.4.60 _rt_rec_recycle() . . . 692
8.22.4.61 _rt_release_tracked_alloc() . . . 692
8.22.4.62 _rt_required_access_mode() . . . 692
8.22.4.63 _rt_required_align() . . . 692

8.22.4.64 _rt_required_size() . . . 692
8.22.4.65 _rt_stream_write_some() . . . 692
8.22.4.66 _rt_string_bytes() . . . 693
8.22.4.67 _rt_time_to_l0_int_nsec() . . . 693
8.22.4.68 _rt_time_to_l0_int_sec() . . . 693
8.22.4.69 _rt_time_write_parts() . . . 694
8.22.4.70 _rt_trace_invalid_drop() . . . 694
8.22.4.71 _rt_track_alloc_record() . . . 694
8.22.4.72 _rt_track_alloc_record_kind() . . . 694
8.22.4.73 _rt_track_arc_bytes() . . . 694
8.22.4.74 _rt_track_static_bytes() . . . 695
8.22.4.75 _rt_tree_insert_at() . . . 695
8.22.4.76 _rt_tree_merge() . . . 695
8.22.4.77 _rt_tree_prio_for() . . . 695
8.22.4.78 _rt_tree_remove_at() . . . 695
8.22.4.79 _rt_untrack_arc_alloc() . . . 695
8.22.4.80 _rt_validate_derived_ptr() . . . 695
8.22.4.81 _unwrap_opt() . . . 695
8.22.4.82 _unwrap_ptr() . . . 696
8.22.4.83 rt_abort() . . . 696
8.22.4.84 rt_alloc() . . . 696
8.22.4.85 rt_array_element() . . . 697
8.22.4.86 rt_calloc() . . . 697
8.22.4.87 rt_delete_file() . . . 698
8.22.4.88 rt_errno() . . . 698
8.22.4.89 rt_exit() . . . 698
8.22.4.90 rt_file_info() . . . 699
8.22.4.91 rt_flush_stderr() . . . 699
8.22.4.92 rt_flush_stdout() . . . 699
8.22.4.93 rt_free() . . . 699
8.22.4.94 rt_get_argc() . . . 700
8.22.4.95 rt_get_argv() . . . 700
8.22.4.96 rt_get_env_var() . . . 700
8.22.4.97 rt_get_pid() . . . 701
8.22.4.98 rt_hash_bool() . . . 701
8.22.4.99 rt_hash_byte() . . . 701
8.22.4.100 rt_hash_data() . . . 702
8.22.4.101 rt_hash_int() . . . 702
8.22.4.102 rt_hash_opt_bool() . . . 702
8.22.4.103 rt_hash_opt_byte() . . . 703
8.22.4.104 rt_hash_opt_int() . . . 703
8.22.4.105 rt_hash_opt_ptr() . . . 703

8.22.4.106 rt_hash_opt_string() . . . 704
8.22.4.107 rt_hash_ptr() . . . 704
8.22.4.108 rt_hash_string() . . . 705
8.22.4.109 rt_memcmp() . . . 705
8.22.4.110 rt_memcpy() . . . 705
8.22.4.111 rt_memset() . . . 706
8.22.4.112 rt_print() . . . 706
8.22.4.113 rt_print_bool() . . . 707
8.22.4.114 rt_print_bool_stderr() . . . 707
8.22.4.115 rt_print_int() . . . 707
8.22.4.116 rt_print_int_stderr() . . . 708
8.22.4.117 rt_print_stderr() . . . 708
8.22.4.118 rt_println() . . . 708
8.22.4.119 rt_println_stderr() . . . 708
8.22.4.120 rt_rand() . . . 708
8.22.4.121 rt_read_char() . . . 709
8.22.4.122 rt_read_file_all() . . . 709
8.22.4.123 rt_read_line() . . . 709
8.22.4.124 rt_realloc() . . . 710
8.22.4.125 rt_register_foreign() . . . 710
8.22.4.126 rt_srand() . . . 710
8.22.4.127 rt_stderr_write() . . . 711
8.22.4.128 rt_stdin_read() . . . 711
8.22.4.129 rt_stdout_write() . . . 711
8.22.4.130 rt_string_bytes_ptr() . . . 712
8.22.4.131 rt_string_compare() . . . 712
8.22.4.132 rt_string_concat() . . . 713
8.22.4.133 rt_string_equals() . . . 713
8.22.4.134 rt_string_from_byte() . . . 713
8.22.4.135 rt_string_from_byte_array() . . . 714
8.22.4.136 rt_string_get() . . . 714
8.22.4.137 rt_string_release() . . . 715
8.22.4.138 rt_string_retain() . . . 715
8.22.4.139 rt_string_slice() . . . 715
8.22.4.140 rt_strlen() . . . 716
8.22.4.141 rt_system() . . . 716
8.22.4.142 rt_time_local_is_dst() . . . 716
8.22.4.143 rt_time_local_offset_sec() . . . 717
8.22.4.144 rt_time_monotonic() . . . 717
8.22.4.145 rt_time_monotonic_supported() . . . 718
8.22.4.146 rt_time_unix() . . . 718
8.22.4.147 rt_unregister_foreign() . . . 718

8.22.4.148 rt_write_file_all() . . . 718
8.22.5 Variable Documentation . . . 719
8.22.5.1 _l0_sh_tag_bool . . . 719
8.22.5.2 _l0_sh_tag_byte . . . 719
8.22.5.3 _l0_sh_tag_data . . . 719
8.22.5.4 _l0_sh_tag_int . . . 719
8.22.5.5 _l0_sh_tag_string . . . 719
8.22.5.6 _rt_alloc_next_generation . . . 719
8.22.5.7 _rt_alloc_table . . . 719
8.22.5.8 _rt_alloc_table_cap . . . 720
8.22.5.9 _rt_alloc_table_cnt . . . 720
8.22.5.10 _rt_alloc_table_tombstones . . . 720
8.22.5.11 _rt_alloc_tree_root . . . 720
8.22.5.12 _rt_argc . . . 720
8.22.5.13 _rt_argv . . . 720
8.22.5.14 _rt_cold_chunks . . . 720
8.22.5.15 _RT_MEM_SENTINEL . . . 720
8.22.5.16 _rt_quarantine_bytes . . . 720
8.22.5.17 _rt_quarantine_count . . . 720
8.22.5.18 _rt_quarantine_head . . . 721
8.22.5.19 _rt_quarantine_tail . . . 721
8.22.5.20 _rt_rec_free_list . . . 721
8.22.5.21 _rt_rec_pool_chunks . . . 721
8.22.5.22 _rt_sh_key . . . 721
8.22.5.23 L0_OPT_STRING_EMPTY . . . 721
8.22.5.24 L0_OPT_STRING_NULL . . . 721
8.22.5.25 L0_STRING_EMPTY . . . 721
8.23 l0_runtime.h . . . 722
8.24 compiler/stage1_py/l0_analysis.py File Reference . . . 759
8.25 compiler/stage1_py/l0_ast.py File Reference . . . 759
8.26 compiler/stage1_py/l0_ast_printer.py File Reference . . . 762
8.27 compiler/stage1_py/l0_backend.py File Reference . . . 762
8.28 compiler/stage1_py/l0_c_emitter.py File Reference . . . 762
8.29 compiler/stage1_py/l0_compilation.py File Reference . . . 763
8.30 compiler/stage1_py/l0_context.py File Reference . . . 763
8.31 compiler/stage1_py/l0_diagnostics.py File Reference . . . 763
8.32 compiler/stage1_py/l0_driver.py File Reference . . . 764
8.33 compiler/stage1_py/l0_expr_types.py File Reference . . . 764
8.34 compiler/stage1_py/l0_internal_error.py File Reference . . . 764
8.35 compiler/stage1_py/l0_lexer.py File Reference . . . 765
8.36 compiler/stage1_py/l0_locals.py File Reference . . . 765
8.37 compiler/stage1_py/l0_logger.py File Reference . . . 766

8.38 compiler/stage1_py/l0_name_resolver.py File Reference . . . 766
8.39 compiler/stage1_py/l0_parser.py File Reference . . . 766
8.40 compiler/stage1_py/l0_paths.py File Reference . . . 767
8.41 compiler/stage1_py/l0_resolve.py File Reference . . . 767
8.42 compiler/stage1_py/l0_scope_context.py File Reference . . . 768
8.43 compiler/stage1_py/l0_signatures.py File Reference . . . 768
8.44 compiler/stage1_py/l0_string_escape.py File Reference . . . 768
8.45 compiler/stage1_py/l0_symbols.py File Reference . . . 769
8.46 compiler/stage1_py/l0_types.py File Reference . . . 769
8.47 compiler/stage1_py/l0c.py File Reference . . . 770
8.48 compiler/stage2_l0/scripts/check_trace_log.py File Reference . . . 772
8.49 compiler/stage2_l0/src/analysis.l0 File Reference . . . 773
8.49.1 Function Documentation . . . 774
8.49.1.1 analysis_analyze_entry() . . . 774
8.49.1.2 analysis_copy_driver_diags() . . . 774
8.49.1.3 analysis_signature_summary() . . . 774
8.50 compiler/stage2_l0/src/ast.l0 File Reference . . . 774
8.50.1 Enumeration Type Documentation . . . 778
8.50.1.1 ExprKind . . . 778
8.50.1.2 PatternKind . . . 779
8.50.1.3 StmtKind . . . 779
8.50.1.4 TopLevelDeclKind . . . 779
8.50.2 Function Documentation . . . 780
8.50.2.1 case_arm_vec_free() . . . 780
8.50.2.2 enum_decl_free() . . . 780
8.50.2.3 enum_variant_free() . . . 780
8.50.2.4 enum_variant_vec_free() . . . 780
8.50.2.5 expr_arena_add() . . . 780
8.50.2.6 expr_arena_create() . . . 781
8.50.2.7 expr_arena_free() . . . 781
8.50.2.8 expr_arena_get() . . . 781
8.50.2.9 expr_node_create() . . . 782
8.50.2.10 expr_node_free() . . . 782
8.50.2.11 field_decl_free() . . . 782
8.50.2.12 field_decl_vec_free() . . . 782
8.50.2.13 func_decl_free() . . . 783
8.50.2.14 id_vec_create() . . . 783
8.50.2.15 id_vec_free() . . . 783
8.50.2.16 id_vec_get() . . . 783
8.50.2.17 id_vec_push() . . . 784
8.50.2.18 import_vec_free() . . . 784
8.50.2.19 invalid_expr_id() . . . 784

8.50.2.20 invalid_pattern_id() . . . 784
8.50.2.21 invalid_stmt_id() . . . 785
8.50.2.22 let_decl_free() . . . 785
8.50.2.23 match_arm_vec_free() . . . 785
8.50.2.24 module_free() . . . 785
8.50.2.25 param_free() . . . 785
8.50.2.26 param_vec_free() . . . 786
8.50.2.27 pattern_arena_add() . . . 786
8.50.2.28 pattern_arena_create() . . . 786
8.50.2.29 pattern_arena_free() . . . 786
8.50.2.30 pattern_arena_get() . . . 787
8.50.2.31 pattern_node_create() . . . 787
8.50.2.32 pattern_node_free() . . . 787
8.50.2.33 ptr_vec_create() . . . 788
8.50.2.34 ptr_vec_get() . . . 788
8.50.2.35 ptr_vec_push() . . . 788
8.50.2.36 span_here() . . . 788
8.50.2.37 stmt_arena_add() . . . 789
8.50.2.38 stmt_arena_create() . . . 789
8.50.2.39 stmt_arena_free() . . . 789
8.50.2.40 stmt_arena_get() . . . 790
8.50.2.41 stmt_node_create() . . . 790
8.50.2.42 stmt_node_free() . . . 790
8.50.2.43 struct_decl_free() . . . 790
8.50.2.44 sv_free_opt() . . . 791
8.50.2.45 top_level_decl_free() . . . 791
8.50.2.46 top_level_decl_vec_free() . . . 791
8.50.2.47 type_alias_decl_free() . . . 791
8.50.2.48 type_ref_free() . . . 792
8.50.2.49 with_item_vec_free() . . . 792
8.50.3 Variable Documentation . . . 792
8.50.3.1 ExprId . . . 792
8.50.3.2 PatternId . . . 792
8.50.3.3 StmtId . . . 792
8.51 compiler/stage2_l0/src/ast_printer.l0 File Reference . . . 792
8.51.1 Function Documentation . . . 794
8.51.1.1 ap_bool_str() . . . 794
8.51.1.2 ap_full_name() . . . 794
8.51.1.3 ap_indent_str() . . . 794
8.51.1.4 ap_module_prefix() . . . 795
8.51.1.5 ap_print_enum_decl() . . . 795
8.51.1.6 ap_print_enum_variant() . . . 795

8.51.1.7 ap_print_expr() . . . 795
8.51.1.8 ap_print_field_decl() . . . 796
8.51.1.9 ap_print_func_decl() . . . 796
8.51.1.10 ap_print_import() . . . 796
8.51.1.11 ap_print_label() . . . 797
8.51.1.12 ap_print_let_decl() . . . 797
8.51.1.13 ap_print_line() . . . 797
8.51.1.14 ap_print_module() . . . 797
8.51.1.15 ap_print_param() . . . 798
8.51.1.16 ap_print_pattern() . . . 798
8.51.1.17 ap_print_stmt() . . . 798
8.51.1.18 ap_print_struct_decl() . . . 798
8.51.1.19 ap_print_top_level_decl() . . . 799
8.51.1.20 ap_print_type_alias_decl() . . . 799
8.51.1.21 ap_print_type_ref() . . . 799
8.51.1.22 ap_quote_string() . . . 799
8.51.1.23 ap_span_suffix() . . . 800
8.52 compiler/stage2_l0/src/backend.l0 File Reference . . . 800
8.52.1 Function Documentation . . . 805
8.52.1.1 backend_generate() . . . 805
8.52.1.2 be_alignof_expr_for_type() . . . 805
8.52.1.3 be_body_reassigns_param() . . . 806
8.52.1.4 be_collect_type_dependencies() . . . 806
8.52.1.5 be_collect_type_keys() . . . 806
8.52.1.6 be_collect_value_type_dependencies() . . . 807
8.52.1.7 be_convert_expr_with_expected_type() . . . 807
8.52.1.8 be_create() . . . 807
8.52.1.9 be_current_env() . . . 808
8.52.1.10 be_current_expr_arena() . . . 808
8.52.1.11 be_current_stmt_arena() . . . 808
8.52.1.12 be_current_unit() . . . 809
8.52.1.13 be_dependency_sets_free() . . . 809
8.52.1.14 be_emit_bare_variant_static_initializer() . . . 809
8.52.1.15 be_emit_binary_op() . . . 809
8.52.1.16 be_emit_block_sequence() . . . 810
8.52.1.17 be_emit_block_stmt() . . . 810
8.52.1.18 be_emit_call() . . . 810
8.52.1.19 be_emit_case_literal() . . . 811
8.52.1.20 be_emit_case_stmt() . . . 811
8.52.1.21 be_emit_checked_pointer_expr() . . . 811
8.52.1.22 be_emit_cleanup_at_scope_exit() . . . 812
8.52.1.23 be_emit_cleanup_for_loop_exit() . . . 812

8.52.1.24 be_emit_cleanup_for_return() . . . 812
8.52.1.25 be_emit_condition_branch() . . . 812
8.52.1.26 be_emit_condition_expr() . . . 813
8.52.1.27 be_emit_condition_value() . . . 813
8.52.1.28 be_emit_const_constructor() . . . 813
8.52.1.29 be_emit_constructor_call() . . . 814
8.52.1.30 be_emit_copy_expr_with_retains() . . . 814
8.52.1.31 be_emit_drop_stmt() . . . 815
8.52.1.32 be_emit_enum_cleanup() . . . 815
8.52.1.33 be_emit_expr() . . . 815
8.52.1.34 be_emit_expr_with_expected_type() . . . 815
8.52.1.35 be_emit_function_declarations() . . . 816
8.52.1.36 be_emit_function_definitions() . . . 816
8.52.1.37 be_emit_if_branch() . . . 816
8.52.1.38 be_emit_inline_with_header_item() . . . 817
8.52.1.39 be_emit_let_declarations() . . . 817
8.52.1.40 be_emit_let_initializer() . . . 817
8.52.1.41 be_emit_line_directive() . . . 817
8.52.1.42 be_emit_lvalue() . . . 818
8.52.1.43 be_emit_main_wrapper_if_needed() . . . 818
8.52.1.44 be_emit_match_stmt() . . . 818
8.52.1.45 be_emit_new_expr() . . . 818
8.52.1.46 be_emit_owned_expr_with_expected_type() . . . 819
8.52.1.47 be_emit_pattern_bindings() . . . 819
8.52.1.48 be_emit_pointer_index_lvalue() . . . 819
8.52.1.49 be_emit_retain_for_copied_value() . . . 820
8.52.1.50 be_emit_return_stmt_with_inline_cleanup() . . . 820
8.52.1.51 be_emit_static_ord_initializer() . . . 820
8.52.1.52 be_emit_stmt() . . . 821
8.52.1.53 be_emit_struct_cleanup() . . . 821
8.52.1.54 be_emit_type_definitions() . . . 821
8.52.1.55 be_emit_value_cleanup() . . . 822
8.52.1.56 be_emit_with_cleanup_from_scope() . . . 822
8.52.1.57 be_emit_with_cleanup_header_let_assign() . . . 822
8.52.1.58 be_emit_with_cleanup_header_let_predecl() . . . 822
8.52.1.59 be_emit_with_stmt() . . . 823
8.52.1.60 be_expr_type() . . . 823
8.52.1.61 be_find_enum_decl() . . . 823
8.52.1.62 be_find_env() . . . 824
8.52.1.63 be_find_struct_decl() . . . 824
8.52.1.64 be_find_unit() . . . 824
8.52.1.65 be_find_variant_decl() . . . 825

8.52.1.66 be_free() . . . 825
8.52.1.67 be_fresh_label() . . . 825
8.52.1.68 be_has_side_effects() . . . 825
8.52.1.69 be_ice() . . . 826
8.52.1.70 be_ice_string() . . . 826
8.52.1.71 be_int_type_size() . . . 826
8.52.1.72 be_is_int_assignable() . . . 827
8.52.1.73 be_is_place_expr() . . . 827
8.52.1.74 be_is_static_let_initializer() . . . 827
8.52.1.75 be_is_unwrap_cast_from_place() . . . 828
8.52.1.76 be_lookup_local_var_type() . . . 828
8.52.1.77 be_lookup_owned_local_name() . . . 828
8.52.1.78 be_lookup_symbol() . . . 829
8.52.1.79 be_materialize_arc_temp() . . . 829
8.52.1.80 be_needs_arc_temp() . . . 829
8.52.1.81 be_pointer_type_or_null() . . . 830
8.52.1.82 be_pop_loop_cleanup_scopes() . . . 830
8.52.1.83 be_pop_loop_labels() . . . 830
8.52.1.84 be_pop_scope() . . . 830
8.52.1.85 be_push_loop_cleanup_scopes() . . . 831
8.52.1.86 be_push_loop_labels() . . . 831
8.52.1.87 be_push_scope() . . . 831
8.52.1.88 be_register_inline_with_cleanup() . . . 831
8.52.1.89 be_resolve_let_type() . . . 832
8.52.1.90 be_resolve_type_ref() . . . 832
8.52.1.91 be_scope_chain_has_cleanup() . . . 832
8.52.1.92 be_should_materialize_arc_temp() . . . 833
8.52.1.93 be_sizeof_expr_for_type() . . . 833
8.52.1.94 be_static_variant_symbol() . . . 833
8.52.1.95 be_type_key() . . . 834
8.52.1.96 be_type_key_module() . . . 834
8.52.1.97 be_type_key_name() . . . 834
8.52.1.98 be_type_ptr_map_temp_free() . . . 835
8.52.1.99 be_type_ptr_vec_temp_free() . . . 835
8.52.1.100 type_free_opt() . . . 835
8.53 compiler/stage2_l0/src/build_driver.l0 File Reference . . . 835
8.53.1 Function Documentation . . . 838
8.53.1.1 bd_append_c_option_words() . . . 838
8.53.1.2 bd_build_to_exe() . . . 838
8.53.1.3 bd_cmd_build() . . . 838
8.53.1.4 bd_cmd_run() . . . 839
8.53.1.5 bd_collect_c_option_words() . . . 839

8.53.1.6 bd_compiler_flag_family() . . . 839
8.53.1.7 bd_default_exe_name() . . . 840
8.53.1.8 bd_emit_write_error() . . . 840
8.53.1.9 bd_exe_suffix() . . . 840
8.53.1.10 bd_find_c_compiler() . . . 840
8.53.1.11 bd_find_last_sep() . . . 840
8.53.1.12 bd_format_word_list() . . . 841
8.53.1.13 bd_get_optimize_flag() . . . 841
8.53.1.14 bd_has_debug_flag() . . . 841
8.53.1.15 bd_has_explicit_opt_flag() . . . 842
8.53.1.16 bd_is_windows() . . . 842
8.53.1.17 bd_join_display_words() . . . 842
8.53.1.18 bd_join_shell_words() . . . 842
8.53.1.19 bd_log_effective_roots() . . . 843
8.53.1.20 bd_merge_c_option_words() . . . 843
8.53.1.21 bd_null_device() . . . 843
8.53.1.22 bd_prepare_build_result() . . . 843
8.53.1.23 bd_prepare_input() . . . 844
8.53.1.24 bd_prepared_input_free() . . . 844
8.53.1.25 bd_prepared_input_free_opt() . . . 844
8.53.1.26 bd_push_c_compile_command_words() . . . 845
8.53.1.27 bd_push_output_flags() . . . 845
8.53.1.28 bd_push_standard_flags() . . . 845
8.53.1.29 bd_replay_compile_output() . . . 845
8.53.1.30 bd_runtime_include_dir() . . . 846
8.53.1.31 bd_runtime_lib_dir() . . . 846
8.53.1.32 bd_shell_quote() . . . 846
8.53.1.33 bd_shell_quote_posix() . . . 847
8.53.1.34 bd_shell_quote_windows() . . . 847
8.53.1.35 bd_validate_entry_main() . . . 847
8.53.1.36 bd_validate_runtime_lib() . . . 848
8.53.1.37 bd_windows_exec_token() . . . 848
8.53.1.38 bd_windows_token_is_plain() . . . 848
8.53.1.39 bd_with_c_suffix() . . . 849
8.54 compiler/stage2_l0/src/build_info.l0 File Reference . . . 849
8.54.1 Function Documentation . . . 850
8.54.1.1 build_info_build_id() . . . 850
8.54.1.2 build_info_build_time() . . . 850
8.54.1.3 build_info_commit() . . . 850
8.54.1.4 build_info_compiler() . . . 850
8.54.1.5 build_info_has_embedded_version() . . . 851
8.54.1.6 build_info_host() . . . 851

8.54.1.7 build_info_identity_with_version() . . . 851
8.54.1.8 build_info_release_version() . . . 851
8.54.1.9 build_info_version_text() . . . 852
8.55 compiler/stage2_l0/src/c_emitter.l0 File Reference . . . 853
8.55.1 Function Documentation . . . 860
8.55.1.1 ccb_create() . . . 860
8.55.1.2 ccb_dedent() . . . 860
8.55.1.3 ccb_emit() . . . 860
8.55.1.4 ccb_emit_raw() . . . 861
8.55.1.5 ccb_free() . . . 861
8.55.1.6 ccb_indent() . . . 861
8.55.1.7 ccb_prefix() . . . 861
8.55.1.8 ccb_to_string() . . . 862
8.55.1.9 cem_analysis() . . . 862
8.55.1.10 cem_collect_opt_wrappers_from_type() . . . 862
8.55.1.11 cem_create() . . . 862
8.55.1.12 cem_emit_alloc_obj() . . . 863
8.55.1.13 cem_emit_assignment() . . . 863
8.55.1.14 cem_emit_binary_op() . . . 863
8.55.1.15 cem_emit_block_end() . . . 864
8.55.1.16 cem_emit_block_start() . . . 864
8.55.1.17 cem_emit_bool_literal() . . . 864
8.55.1.18 cem_emit_byte_literal() . . . 864
8.55.1.19 cem_emit_case_label() . . . 865
8.55.1.20 cem_emit_cast() . . . 865
8.55.1.21 cem_emit_checked_int_add() . . . 865
8.55.1.22 cem_emit_checked_int_div() . . . 866
8.55.1.23 cem_emit_checked_int_mod() . . . 866
8.55.1.24 cem_emit_checked_int_mul() . . . 866
8.55.1.25 cem_emit_checked_int_sub() . . . 867
8.55.1.26 cem_emit_checked_narrow_cast() . . . 867
8.55.1.27 cem_emit_checked_pointer_assignment() . . . 867
8.55.1.28 cem_emit_checked_ptr_access() . . . 868
8.55.1.29 cem_emit_checked_ptr_access_for_base() . . . 868
8.55.1.30 cem_emit_checked_ptr_index_access() . . . 868
8.55.1.31 cem_emit_comment() . . . 869
8.55.1.32 cem_emit_condition_binary_op() . . . 869
8.55.1.33 cem_emit_condition_pointer_null_check() . . . 869
8.55.1.34 cem_emit_condition_string_compare_call() . . . 870
8.55.1.35 cem_emit_const_bool_literal() . . . 870
8.55.1.36 cem_emit_const_string_literal() . . . 870
8.55.1.37 cem_emit_default_label() . . . 871

8.55.1.38 cem_emit_deref_lvalue() . . . 871
8.55.1.39 cem_emit_discard_expr() . . . 871
8.55.1.40 cem_emit_drop_begin_expr() . . . 872
8.55.1.41 cem_emit_drop_finish_call() . . . 872
8.55.1.42 cem_emit_else() . . . 872
8.55.1.43 cem_emit_enum() . . . 872
8.55.1.44 cem_emit_enum_cleanup() . . . 873
8.55.1.45 cem_emit_enum_payload_field_access() . . . 873
8.55.1.46 cem_emit_enum_tag() . . . 873
8.55.1.47 cem_emit_enum_tag_access() . . . 874
8.55.1.48 cem_emit_enum_variant_init() . . . 874
8.55.1.49 cem_emit_exit_switch() . . . 874
8.55.1.50 cem_emit_expr_stmt() . . . 874
8.55.1.51 cem_emit_field_access() . . . 875
8.55.1.52 cem_emit_field_lvalue() . . . 875
8.55.1.53 cem_emit_for_loop_end() . . . 875
8.55.1.54 cem_emit_for_loop_start() . . . 876
8.55.1.55 cem_emit_forward_decls() . . . 876
8.55.1.56 cem_emit_function_call() . . . 876
8.55.1.57 cem_emit_function_declaration() . . . 876
8.55.1.58 cem_emit_function_definition_footer() . . . 877
8.55.1.59 cem_emit_function_definition_header() . . . 877
8.55.1.60 cem_emit_goto() . . . 877
8.55.1.61 cem_emit_header() . . . 877
8.55.1.62 cem_emit_if_header() . . . 878
8.55.1.63 cem_emit_index_lvalue() . . . 878
8.55.1.64 cem_emit_int_literal() . . . 878
8.55.1.65 cem_emit_label() . . . 879
8.55.1.66 cem_emit_let_decl() . . . 879
8.55.1.67 cem_emit_let_declaration() . . . 879
8.55.1.68 cem_emit_line_directive() . . . 879
8.55.1.69 cem_emit_main_wrapper() . . . 880
8.55.1.70 cem_emit_match_scrutinee_decl() . . . 880
8.55.1.71 cem_emit_match_switch_start() . . . 880
8.55.1.72 cem_emit_module_comment() . . . 880
8.55.1.73 cem_emit_module_separator() . . . 881
8.55.1.74 cem_emit_negated_condition() . . . 881
8.55.1.75 cem_emit_none_value_for_nullable() . . . 881
8.55.1.76 cem_emit_null_assignment() . . . 882
8.55.1.77 cem_emit_null_check_eq() . . . 882
8.55.1.78 cem_emit_null_check_ne() . . . 882
8.55.1.79 cem_emit_null_literal() . . . 882

8.55.1.80 cem_emit_optional_has_value() . . . 883
8.55.1.81 cem_emit_optional_value() . . . 883
8.55.1.82 cem_emit_optional_wrapper() . . . 883
8.55.1.83 cem_emit_optional_wrapper_for_defined_type() . . . 884
8.55.1.84 cem_emit_optional_wrappers() . . . 884
8.55.1.85 cem_emit_ord() . . . 884
8.55.1.86 cem_emit_paren_expr() . . . 884
8.55.1.87 cem_emit_pattern_binding_init() . . . 885
8.55.1.88 cem_emit_pointer_assignment() . . . 885
8.55.1.89 cem_emit_pointer_null_check() . . . 885
8.55.1.90 cem_emit_pointer_type() . . . 886
8.55.1.91 cem_emit_return_stmt() . . . 886
8.55.1.92 cem_emit_return_void_stmt() . . . 886
8.55.1.93 cem_emit_section_comment() . . . 886
8.55.1.94 cem_emit_sizeof_type() . . . 887
8.55.1.95 cem_emit_some_value_for_nullable() . . . 887
8.55.1.96 cem_emit_string_compare_call() . . . 887
8.55.1.97 cem_emit_string_concat_call() . . . 888
8.55.1.98 cem_emit_string_equals_call() . . . 888
8.55.1.99 cem_emit_string_literal() . . . 888
8.55.1.100 cem_emit_string_release() . . . 889
8.55.1.101 cem_emit_string_retain() . . . 889
8.55.1.102 cem_emit_struct() . . . 889
8.55.1.103 cem_emit_struct_cleanup() . . . 889
8.55.1.104 cem_emit_struct_constructor() . . . 890
8.55.1.105 cem_emit_struct_constructor_for_type() . . . 890
8.55.1.106 cem_emit_struct_init() . . . 890
8.55.1.107 cem_emit_struct_init_from_fields() . . . 891
8.55.1.108 cem_emit_struct_static_initializer() . . . 891
8.55.1.109 cem_emit_struct_static_initializer_for_type() . . . 891
8.55.1.110 cem_emit_switch_end() . . . 892
8.55.1.111 cem_emit_switch_start() . . . 892
8.55.1.112 cem_emit_temp_decl() . . . 892
8.55.1.113 cem_emit_top_level_let_cleanup() . . . 892
8.55.1.114 cem_emit_try_check_niche() . . . 893
8.55.1.115 cem_emit_try_check_value() . . . 893
8.55.1.116 cem_emit_try_extract_value() . . . 893
8.55.1.117 cem_emit_type() . . . 893
8.55.1.118 cem_emit_unary_op() . . . 895
8.55.1.119 cem_emit_unreachable_comment() . . . 895
8.55.1.120 cem_emit_unreachable_marker() . . . 895
8.55.1.121 cem_emit_unwrap_opt() . . . 896

8.55.1.122 cem_emit_unwrap_ptr() . . . 896
8.55.1.123 cem_emit_value_cleanup() . . . 896
8.55.1.124 cem_emit_var_ref() . . . 897
8.55.1.125 cem_emit_variant_constructor() . . . 897
8.55.1.126 cem_emit_variant_constructor_for_type() . . . 897
8.55.1.127 cem_emit_variant_static_initializer() . . . 898
8.55.1.128 cem_emit_variant_static_initializer_for_type() . . . 898
8.55.1.129 cem_emit_while_header() . . . 898
8.55.1.130 cem_emit_widen_int() . . . 899
8.55.1.131 cem_emit_zero_init() . . . 899
8.55.1.132 cem_encode_c_byte_literal() . . . 899
8.55.1.133 cem_enum_has_arc_data() . . . 899
8.55.1.134 cem_find_enum_decl() . . . 900
8.55.1.135 cem_find_enum_variant_info() . . . 900
8.55.1.136 cem_free() . . . 900
8.55.1.137 cem_fresh_tmp() . . . 901
8.55.1.138 cem_function_params() . . . 901
8.55.1.139 cem_get_output() . . . 901
8.55.1.140 cem_ice() . . . 902
8.55.1.141 cem_ice_string() . . . 902
8.55.1.142 cem_is_c_keyword() . . . 902
8.55.1.143 cem_is_early_inner() . . . 902
8.55.1.144 cem_is_niche_nullable() . . . 903
8.55.1.145 cem_mangle_enum_name() . . . 903
8.55.1.146 cem_mangle_function_name() . . . 903
8.55.1.147 cem_mangle_identifier() . . . 904
8.55.1.148 cem_mangle_let_name() . . . 904
8.55.1.149 cem_mangle_module() . . . 904
8.55.1.150 cem_mangle_struct_name() . . . 905
8.55.1.151 cem_module_filename() . . . 905
8.55.1.152 cem_note_opt_wrapper() . . . 905
8.55.1.153 cem_opt_key_for_type() . . . 905
8.55.1.154 cem_opt_wrapper_map_free() . . . 906
8.55.1.155 cem_opt_wrapper_name_for_inner() . . . 906
8.55.1.156 cem_prepare_optional_wrappers() . . . 906
8.55.1.157 cem_reset_optional_wrappers() . . . 906
8.55.1.158 cem_restore_active_line_directive() . . . 907
8.55.1.159 cem_set_analysis() . . . 907
8.55.1.160 cem_upper_ascii() . . . 907
8.56 compiler/stage2_l0/src/cli_args.l0 File Reference . . . 907
8.56.1 Enumeration Type Documentation . . . 909
8.56.1.1 CliMode . . . 909

8.56.2 Function Documentation . . . 909
8.56.2.1 cli_assign_target() . . . 909
8.56.2.2 cli_compiler_identity_text() . . . 911
8.56.2.3 cli_diag_free_opt() . . . 911
8.56.2.4 cli_emit_error() . . . 911
8.56.2.5 cli_emit_reserved_option_error() . . . 911
8.56.2.6 cli_has_diagnostic_code() . . . 912
8.56.2.7 cli_has_help_flag() . . . 912
8.56.2.8 cli_has_version_flag() . . . 912
8.56.2.9 cli_mode_flag() . . . 913
8.56.2.10 cli_mode_name() . . . 913
8.56.2.11 cli_options_create() . . . 913
8.56.2.12 cli_options_free() . . . 914
8.56.2.13 cli_options_free_opt() . . . 914
8.56.2.14 cli_parse() . . . 914
8.56.2.15 cli_parse_result_free() . . . 914
8.56.2.16 cli_print_blank_line() . . . 915
8.56.2.17 cli_print_help() . . . 915
8.56.2.18 cli_print_line() . . . 915
8.56.2.19 cli_print_usage() . . . 915
8.56.2.20 cli_print_version() . . . 915
8.56.2.21 cli_require_next_value() . . . 916
8.56.2.22 cli_split_index() . . . 916
8.56.2.23 cli_try_set_mode() . . . 916
8.56.2.24 cli_validate_mode_scoped_flags() . . . 917
8.56.2.25 cli_verbose_cluster_count() . . . 917
8.57 compiler/stage2_l0/src/codegen_options.l0 File Reference . . . 917
8.57.1 Function Documentation . . . 918
8.57.1.1 cg_options_create() . . . 918
8.57.1.2 cg_options_free() . . . 918
8.58 compiler/stage2_l0/src/compiler_filesystem.l0 File Reference . . . 918
8.58.1 Function Documentation . . . 920
8.58.1.1 cfs_apply_cleanup_status() . . . 920
8.58.1.2 cfs_error() . . . 920
8.58.1.3 cfs_join_child() . . . 920
8.58.1.4 cfs_mkdir() . . . 920
8.58.1.5 cfs_path_kind() . . . 920
8.58.1.6 cfs_path_kind_follow() . . . 920
8.58.1.7 cfs_remove_empty_dir() . . . 920
8.58.1.8 cfs_remove_regular_file() . . . 920
8.58.1.9 cfs_resolve_trusted_temp_parent() . . . 920
8.58.1.10 cfs_select_temp_parent() . . . 921

8.58.1.11 cfs_select_temp_parent_from_values() . . . 921
8.58.1.12 cfs_workspace_child() . . . 921
8.58.1.13 cfs_workspace_cleanup() . . . 921
8.58.1.14 cfs_workspace_create() . . . 922
8.58.1.15 cfs_workspace_create_for_identity() . . . 922
8.58.1.16 cfs_workspace_free() . . . 922
8.58.1.17 l0c_fs_join_child() . . . 922
8.58.1.18 l0c_fs_mkdir() . . . 922
8.58.1.19 l0c_fs_path_kind_follow() . . . 922
8.58.1.20 l0c_fs_path_kind_nofollow() . . . 922
8.58.1.21 l0c_fs_remove_empty_dir() . . . 922
8.58.1.22 l0c_fs_remove_regular_file() . . . 923
8.58.1.23 l0c_fs_resolve_trusted_temp_parent() . . . 923
8.58.2 Variable Documentation . . . 923
8.58.2.1 CFS_PATH_ABSENT . . . 923
8.58.2.2 CFS_PATH_DIRECTORY . . . 923
8.58.2.3 CFS_PATH_OTHER . . . 923
8.58.2.4 CFS_PATH_REGULAR . . . 923
8.59 compiler/stage2_l0/src/diag_print.l0 File Reference . . . 923
8.59.1 Function Documentation . . . 924
8.59.1.1 dp_format_header() . . . 924
8.59.1.2 dp_print_collector() . . . 924
8.59.1.3 dp_print_collector_with_sources() . . . 925
8.59.1.4 dp_print_diag() . . . 925
8.59.1.5 dp_print_header() . . . 925
8.59.1.6 dp_print_snippet() . . . 925
8.59.1.7 dp_render_snippet() . . . 926
8.59.1.8 dp_severity_name() . . . 926
8.59.1.9 dp_source_cached() . . . 926
8.60 compiler/stage2_l0/src/driver.l0 File Reference . . . 926
8.60.1 Function Documentation . . . 928
8.60.1.1 dr_copy_parse_diags() . . . 928
8.60.1.2 dr_emit_error() . . . 928
8.60.1.3 dr_load_module() . . . 928
8.60.1.4 dr_mark_loaded() . . . 929
8.60.1.5 dr_parse_source() . . . 929
8.60.1.6 dr_sv_has() . . . 929
8.60.1.7 dr_unit_free() . . . 930
8.60.1.8 dr_units_free() . . . 930
8.60.1.9 dr_visit_closure() . . . 930
8.60.1.10 driver_analyze_entry() . . . 931
8.60.1.11 driver_diag_count() . . . 931

8.60.1.12 driver_diag_get() . . . 931
8.60.1.13 driver_find_unit() . . . 932
8.60.1.14 driver_has_errors() . . . 932
8.60.1.15 driver_module_names_sorted() . . . 932
8.60.1.16 driver_state_free() . . . 932
8.60.1.17 driver_unit_count() . . . 933
8.60.1.18 driver_unit_get() . . . 933
8.61 compiler/stage2_l0/src/expr_types.l0 File Reference . . . 933
8.61.1 Enumeration Type Documentation . . . 938
8.61.1.1 AssignCheckResult . . . 938
8.61.1.2 StmtFlowKind . . . 938
8.61.2 Function Documentation . . . 939
8.61.2.1 etc_alive_declare() . . . 939
8.61.2.2 etc_alive_lookup() . . . 939
8.61.2.3 etc_alive_revive_assignment_target() . . . 939
8.61.2.4 etc_alive_scope_clone() . . . 939
8.61.2.5 etc_alive_scope_current() . . . 940
8.61.2.6 etc_alive_scope_pop() . . . 940
8.61.2.7 etc_alive_scope_push() . . . 940
8.61.2.8 etc_alive_scope_stack_clone() . . . 941
8.61.2.9 etc_alive_scope_stack_equal() . . . 941
8.61.2.10 etc_alive_scope_stack_free() . . . 941
8.61.2.11 etc_alive_scope_stack_meet() . . . 941
8.61.2.12 etc_alive_scope_stack_meet_many() . . . 941
8.61.2.13 etc_alive_scope_stack_restore() . . . 942
8.61.2.14 etc_alive_set() . . . 942
8.61.2.15 etc_alive_state_vec_clone() . . . 942
8.61.2.16 etc_alive_state_vec_free() . . . 942
8.61.2.17 etc_alive_state_vec_push_clone() . . . 942
8.61.2.18 etc_bind_pattern_types() . . . 942
8.61.2.19 etc_can_assign() . . . 943
8.61.2.20 etc_can_cast() . . . 943
8.61.2.21 etc_capture_break_state() . . . 943
8.61.2.22 etc_capture_continue_state() . . . 943
8.61.2.23 etc_case_literal_key() . . . 943
8.61.2.24 etc_cast_const_int() . . . 944
8.61.2.25 etc_cast_is_const_null() . . . 944
8.61.2.26 etc_check_assign_expr() . . . 944
8.61.2.27 etc_check_dead_liveness_stmt() . . . 945
8.61.2.28 etc_check_expr_liveness() . . . 945
8.61.2.29 etc_check_function() . . . 945
8.61.2.30 etc_check_initializer_expr() . . . 946

8.61.2.31 etc_check_local_declaration() . . . 946
8.61.2.32 etc_check_loop_iteration() . . . 946
8.61.2.33 etc_check_match_exhaustiveness() . . . 946
8.61.2.34 etc_check_match_pattern() . . . 947
8.61.2.35 etc_check_stmt() . . . 947
8.61.2.36 etc_check_top_level_initializers() . . . 948
8.61.2.37 etc_check_update_from_backedge_states() . . . 948
8.61.2.38 etc_cleanup_header_guard_stack_free() . . . 948
8.61.2.39 etc_decl_outer_has() . . . 948
8.61.2.40 etc_decl_scope_current() . . . 949
8.61.2.41 etc_decl_scope_pop() . . . 949
8.61.2.42 etc_decl_scope_push() . . . 949
8.61.2.43 etc_diag_error() . . . 949
8.61.2.44 etc_diag_type_ref_failure() . . . 950
8.61.2.45 etc_diag_warn() . . . 950
8.61.2.46 etc_expr_contains_try() . . . 950
8.61.2.47 etc_find_enum_variant_info() . . . 952
8.61.2.48 etc_format_missing_match_variants() . . . 952
8.61.2.49 etc_get_pattern_var_type() . . . 952
8.61.2.50 etc_infer_binary() . . . 953
8.61.2.51 etc_infer_call() . . . 953
8.61.2.52 etc_infer_expr() . . . 953
8.61.2.53 etc_infer_field() . . . 954
8.61.2.54 etc_infer_index() . . . 954
8.61.2.55 etc_infer_ord_intrinsic() . . . 955
8.61.2.56 etc_infer_sizeof_intrinsic() . . . 955
8.61.2.57 etc_infer_unary() . . . 955
8.61.2.58 etc_is_bool() . . . 956
8.61.2.59 etc_is_case_type() . . . 956
8.61.2.60 etc_is_guarded_cleanup_header_local() . . . 956
8.61.2.61 etc_is_int() . . . 957
8.61.2.62 etc_is_string() . . . 957
8.61.2.63 etc_is_void() . . . 957
8.61.2.64 etc_key() . . . 957
8.61.2.65 etc_local_symbol_type() . . . 958
8.61.2.66 etc_lookup_local() . . . 958
8.61.2.67 etc_lookup_symbol() . . . 958
8.61.2.68 etc_lookup_visible_local_symbol() . . . 959
8.61.2.69 etc_loop_flow_capture_free() . . . 959
8.61.2.70 etc_loop_liveness_fixed_point() . . . 959
8.61.2.71 etc_match_variants_fully_covered() . . . 959
8.61.2.72 etc_match_wildcard_is_unreachable() . . . 959

8.61.2.73 etc_nested_pattern_path_message() . . . 959
8.61.2.74 etc_nested_symbol_path_full() . . . 960
8.61.2.75 etc_nested_symbol_path_message() . . . 960
8.61.2.76 etc_nested_symbol_path_simple() . . . 960
8.61.2.77 etc_pattern_symbol_name() . . . 961
8.61.2.78 etc_pattern_var_key() . . . 961
8.61.2.79 etc_scope_chain_push() . . . 961
8.61.2.80 etc_set_expr_type() . . . 962
8.61.2.81 etc_set_intrinsic_target() . . . 962
8.61.2.82 etc_set_pattern_var_type() . . . 962
8.61.2.83 etc_set_var_res() . . . 962
8.61.2.84 etc_stmt_contains_try() . . . 963
8.61.2.85 etc_symbol_as_type() . . . 963
8.61.2.86 etc_try_resolve_type_name() . . . 963
8.61.2.87 etc_type_free_opt() . . . 964
8.61.2.88 etc_types_equal() . . . 964
8.61.2.89 expr_types_check() . . . 964
8.62 compiler/stage2_l0/src/l0c.l0 File Reference . . . 964
8.62.1 Function Documentation . . . 965
8.62.1.1 main() . . . 965
8.63 compiler/stage2_l0/src/l0c_lib.l0 File Reference . . . 965
8.63.1 Function Documentation . . . 966
8.63.1.1 l0c_append_c_option_words() . . . 966
8.63.1.2 l0c_cmd_ast() . . . 967
8.63.1.3 l0c_cmd_check() . . . 967
8.63.1.4 l0c_cmd_gen() . . . 967
8.63.1.5 l0c_cmd_nyi() . . . 968
8.63.1.6 l0c_cmd_sym() . . . 968
8.63.1.7 l0c_cmd_tok() . . . 968
8.63.1.8 l0c_cmd_type() . . . 969
8.63.1.9 l0c_collect_c_option_words() . . . 969
8.63.1.10 l0c_copy_collector() . . . 969
8.63.1.11 l0c_dump_tokens_for_path() . . . 970
8.63.1.12 l0c_log_effective_roots() . . . 970
8.63.1.13 l0c_merge_c_option_words() . . . 970
8.63.1.14 l0c_prepare_input() . . . 971
8.63.1.15 l0c_print_module_token_header() . . . 971
8.63.1.16 l0c_print_token() . . . 971
8.63.1.17 prepared_input_free() . . . 971
8.63.1.18 prepared_input_free_opt() . . . 973
8.63.1.19 run_with_argv() . . . 973
8.64 compiler/stage2_l0/src/lexer.l0 File Reference . . . 973

8.64.1 Function Documentation . . . 975
8.64.1.1 escape_char_value() . . . 975
8.64.1.2 hex_char_to_int() . . . 975
8.64.1.3 is_escape_char() . . . 976
8.64.1.4 is_hex_digit() . . . 976
8.64.1.5 is_ident_part() . . . 976
8.64.1.6 is_ident_start() . . . 977
8.64.1.7 is_octal_digit() . . . 977
8.64.1.8 is_printable_ascii() . . . 977
8.64.1.9 ls_advance() . . . 977
8.64.1.10 ls_at_end() . . . 978
8.64.1.11 ls_clear_pending_error() . . . 978
8.64.1.12 ls_create() . . . 978
8.64.1.13 ls_defer_recoverable_error() . . . 979
8.64.1.14 ls_emit_error() . . . 979
8.64.1.15 ls_free() . . . 979
8.64.1.16 ls_has_errors() . . . 979
8.64.1.17 ls_has_queued_tokens() . . . 980
8.64.1.18 ls_next_token() . . . 980
8.64.1.19 ls_peek() . . . 980
8.64.1.20 ls_peek_next() . . . 980
8.64.1.21 ls_queue_pending_errors() . . . 981
8.64.1.22 ls_queue_pending_recovery() . . . 981
8.64.1.23 ls_queue_terminal_recovery() . . . 981
8.64.1.24 ls_queue_token() . . . 982
8.64.1.25 ls_read_byte_literal() . . . 982
8.64.1.26 ls_read_number() . . . 982
8.64.1.27 ls_read_string_literal() . . . 983
8.64.1.28 ls_read_valid_char_escape() . . . 983
8.64.1.29 ls_skip_invalid_characters() . . . 983
8.64.1.30 ls_skip_whitespace_and_comments() . . . 983
8.64.1.31 ls_take_queued_token() . . . 983
8.64.1.32 tokenize() . . . 984
8.65 compiler/stage2_l0/src/locals.l0 File Reference . . . 984
8.65.1 Enumeration Type Documentation . . . 986
8.65.1.1 LocalKind . . . 986
8.65.2 Function Documentation . . . 986
8.65.2.1 lc_bind_pattern() . . . 986
8.65.2.2 lc_func_env_free() . . . 986
8.65.2.3 lc_func_env_get_block_scope() . . . 986
8.65.2.4 lc_func_env_get_match_arm_scope() . . . 987
8.65.2.5 lc_key() . . . 987

8.65.2.6 lc_make_function_env() . . . 987
8.65.2.7 lc_scope_create() . . . 988
8.65.2.8 lc_scope_define() . . . 988
8.65.2.9 lc_scope_free() . . . 988
8.65.2.10 lc_scope_lookup() . . . 989
8.65.2.11 lc_scope_set_block() . . . 989
8.65.2.12 lc_scope_track() . . . 989
8.65.2.13 lc_visit_block() . . . 990
8.65.2.14 lc_visit_case_arm() . . . 990
8.65.2.15 lc_visit_case_else() . . . 990
8.65.2.16 lc_visit_match_arm() . . . 990
8.65.2.17 lc_visit_scoped_stmt() . . . 991
8.65.2.18 lc_visit_stmt() . . . 991
8.65.2.19 lc_visit_with() . . . 991
8.65.2.20 locals_env_count() . . . 991
8.65.2.21 locals_env_find() . . . 992
8.65.2.22 locals_env_get() . . . 992
8.65.2.23 locals_free() . . . 992
8.65.2.24 locals_resolve() . . . 993
8.66 compiler/stage2_l0/src/name_resolver.l0 File Reference . . . 993
8.66.1 Function Documentation . . . 995
8.66.1.1 nr_build_top_key() . . . 995
8.66.1.2 nr_collect_locals() . . . 995
8.66.1.3 nr_define_local() . . . 995
8.66.1.4 nr_env_count() . . . 996
8.66.1.5 nr_env_find() . . . 996
8.66.1.6 nr_env_get() . . . 996
8.66.1.7 nr_extern_signatures_compatible() . . . 996
8.66.1.8 nr_free() . . . 997
8.66.1.9 nr_make_env() . . . 997
8.66.1.10 nr_map_symbol_get() . . . 997
8.66.1.11 nr_module_env_all() . . . 998
8.66.1.12 nr_module_env_all_keys() . . . 998
8.66.1.13 nr_module_env_free() . . . 998
8.66.1.14 nr_module_env_imported() . . . 998
8.66.1.15 nr_module_env_imported_keys() . . . 999
8.66.1.16 nr_module_env_imports_module() . . . 999
8.66.1.17 nr_module_env_is_ambiguous() . . . 999
8.66.1.18 nr_module_env_local() . . . 1000
8.66.1.19 nr_module_env_local_keys() . . . 1000
8.66.1.20 nr_open_imports() . . . 1000
8.66.1.21 nr_open_imports_one() . . . 1001

8.66.1.22 nr_path_is_same_signature() . . . 1001
8.66.1.23 nr_resolve() . . . 1001
8.66.1.24 nr_span_diag_error() . . . 1002
8.66.1.25 nr_span_diag_warn() . . . 1002
8.66.1.26 nr_symbol_free_map() . . . 1002
8.67 compiler/stage2_l0/src/parser.l0 File Reference . . . 1003
8.67.1 Function Documentation . . . 1003
8.67.1.1 impl_result_to_public() . . . 1003
8.67.1.2 parse_diag_count() . . . 1004
8.67.1.3 parse_diag_get() . . . 1004
8.67.1.4 parse_has_errors() . . . 1004
8.67.1.5 parse_lex_error_result() . . . 1005
8.67.1.6 parse_module_source() . . . 1005
8.67.1.7 parse_module_tokens() . . . 1005
8.67.1.8 parse_module_tokens_owned() . . . 1006
8.67.1.9 parse_result_free() . . . 1006
8.68 compiler/stage2_l0/src/parser/decl.l0 File Reference . . . 1006
8.68.1 Function Documentation . . . 1007
8.68.1.1 impl_parse_module_source() . . . 1007
8.68.1.2 impl_parse_module_tokens() . . . 1007
8.68.1.3 ps_finish_function() . . . 1008
8.68.1.4 ps_lex_error_result() . . . 1008
8.68.1.5 ps_make_empty_block_stmt() . . . 1008
8.68.1.6 ps_parse_enum() . . . 1009
8.68.1.7 ps_parse_extern_func() . . . 1009
8.68.1.8 ps_parse_function() . . . 1009
8.68.1.9 ps_parse_module() . . . 1010
8.68.1.10 ps_parse_struct() . . . 1010
8.68.1.11 ps_parse_top_level_decl() . . . 1010
8.68.1.12 ps_parse_top_level_let() . . . 1011
8.68.1.13 ps_parse_type_alias() . . . 1011
8.68.1.14 ps_tokens_result() . . . 1011
8.69 compiler/stage2_l0/src/parser/expr.l0 File Reference . . . 1012
8.69.1 Function Documentation . . . 1012
8.69.1.1 ps_is_builtin_type_name() . . . 1012
8.69.1.2 ps_is_unambiguous_type_start() . . . 1013
8.69.1.3 ps_parse_add_expr() . . . 1013
8.69.1.4 ps_parse_and_expr() . . . 1013
8.69.1.5 ps_parse_call_argument() . . . 1014
8.69.1.6 ps_parse_cast_expr() . . . 1014
8.69.1.7 ps_parse_equality_expr() . . . 1014
8.69.1.8 ps_parse_expr() . . . 1015

8.69.1.9 ps_parse_mul_expr() . . . 1015
8.69.1.10 ps_parse_or_expr() . . . 1015
8.69.1.11 ps_parse_postfix_expr() . . . 1015
8.69.1.12 ps_parse_primary_expr() . . . 1016
8.69.1.13 ps_parse_rel_expr() . . . 1016
8.69.1.14 ps_parse_unary_expr() . . . 1016
8.70 compiler/stage2_l0/src/parser/shared.l0 File Reference . . . 1017
8.70.1 Function Documentation . . . 1019
8.70.1.1 parse_dotted_name_rest() . . . 1019
8.70.1.2 ps_advance() . . . 1020
8.70.1.3 ps_at_end() . . . 1020
8.70.1.4 ps_at_stmt_start() . . . 1020
8.70.1.5 ps_check() . . . 1021
8.70.1.6 ps_check_byte() . . . 1021
8.70.1.7 ps_check_ident() . . . 1021
8.70.1.8 ps_check_int() . . . 1021
8.70.1.9 ps_check_reserved_binary_op() . . . 1022
8.70.1.10 ps_check_string() . . . 1022
8.70.1.11 ps_create() . . . 1022
8.70.1.12 ps_destroy() . . . 1023
8.70.1.13 ps_emit_error() . . . 1023
8.70.1.14 ps_emit_remaining_lexer_errors() . . . 1023
8.70.1.15 ps_expect() . . . 1023
8.70.1.16 ps_expect_ident() . . . 1024
8.70.1.17 ps_expect_variable_name() . . . 1024
8.70.1.18 ps_extend_span() . . . 1024
8.70.1.19 ps_has_error() . . . 1025
8.70.1.20 ps_is_top_level_start() . . . 1025
8.70.1.21 ps_last() . . . 1025
8.70.1.22 ps_logical_token_at_cursor() . . . 1026
8.70.1.23 ps_match() . . . 1026
8.70.1.24 ps_match_any2() . . . 1026
8.70.1.25 ps_match_any3() . . . 1027
8.70.1.26 ps_match_any4() . . . 1027
8.70.1.27 ps_match_byte() . . . 1027
8.70.1.28 ps_match_ident() . . . 1028
8.70.1.29 ps_match_int() . . . 1028
8.70.1.30 ps_match_string() . . . 1028
8.70.1.31 ps_parse_type() . . . 1029
8.70.1.32 ps_peek() . . . 1029
8.70.1.33 ps_peek_logical_ahead() . . . 1029
8.70.1.34 ps_skip_lexer_errors() . . . 1030

8.70.1.35 ps_span_start() . . . 1030
8.70.1.36 ps_sync_stmt() . . . 1030
8.70.1.37 ps_sync_top_level() . . . 1030
8.70.1.38 ps_try_parse_qualified_name() . . . 1031
8.70.1.39 qn_result_free_all() . . . 1031
8.70.1.40 tok_is() . . . 1031
8.70.1.41 tok_is_byte() . . . 1031
8.70.1.42 tok_is_future_extension() . . . 1032
8.70.1.43 tok_is_ident() . . . 1032
8.70.1.44 tok_is_int() . . . 1032
8.70.1.45 tok_is_string() . . . 1033
8.70.1.46 token_byte_text() . . . 1033
8.70.1.47 token_byte_value() . . . 1033
8.70.1.48 token_ident_text() . . . 1034
8.70.1.49 token_int_text() . . . 1034
8.70.1.50 token_int_value() . . . 1034
8.70.1.51 token_string_text() . . . 1035
8.70.1.52 token_string_value() . . . 1035
8.70.1.53 token_text_for_width() . . . 1035
8.70.1.54 token_width() . . . 1035
8.71 compiler/stage2_l0/src/parser/stmt.l0 File Reference . . . 1036
8.71.1 Function Documentation . . . 1037
8.71.1.1 ps_at_case_arm_start() . . . 1037
8.71.1.2 ps_parse_block() . . . 1037
8.71.1.3 ps_parse_break_stmt() . . . 1037
8.71.1.4 ps_parse_case_literal() . . . 1038
8.71.1.5 ps_parse_case_stmt() . . . 1038
8.71.1.6 ps_parse_continue_stmt() . . . 1038
8.71.1.7 ps_parse_drop_stmt() . . . 1039
8.71.1.8 ps_parse_for_stmt() . . . 1039
8.71.1.9 ps_parse_for_update_stmt() . . . 1039
8.71.1.10 ps_parse_if_stmt() . . . 1040
8.71.1.11 ps_parse_let_stmt() . . . 1040
8.71.1.12 ps_parse_match_stmt() . . . 1040
8.71.1.13 ps_parse_pattern() . . . 1040
8.71.1.14 ps_parse_return_stmt() . . . 1041
8.71.1.15 ps_parse_simple_stmt() . . . 1041
8.71.1.16 ps_parse_stmt() . . . 1041
8.71.1.17 ps_parse_while_stmt() . . . 1042
8.71.1.18 ps_parse_with_item() . . . 1042
8.71.1.19 ps_parse_with_stmt() . . . 1042
8.71.1.20 ps_sync_case_invalid_arm() . . . 1043

8.72 compiler/stage2_l0/src/scope_context.l0 File Reference . . . 1043
8.72.1 Function Documentation . . . 1044
8.72.1.1 sc_add_declared() . . . 1044
8.72.1.2 sc_add_owned() . . . 1044
8.72.1.3 sc_create() . . . 1044
8.72.1.4 sc_free() . . . 1044
8.72.1.5 sc_has_inline_cleanup() . . . 1045
8.72.1.6 sc_inline_cleanup_free_opt() . . . 1045
8.72.1.7 sc_lookup_declared_type() . . . 1045
8.72.1.8 sc_push_inline_cleanup() . . . 1045
8.72.1.9 sc_remove_owned() . . . 1046
8.72.1.10 sc_var_free() . . . 1046
8.73 compiler/stage2_l0/src/sem_context.l0 File Reference . . . 1046
8.73.1 Enumeration Type Documentation . . . 1047
8.73.1.1 VarRefResolution . . . 1047
8.73.2 Function Documentation . . . 1048
8.73.2.1 analysis_diag_count() . . . 1048
8.73.2.2 analysis_diag_get() . . . 1048
8.73.2.3 analysis_expr_key() . . . 1048
8.73.2.4 analysis_get_expr_type() . . . 1049
8.73.2.5 analysis_get_intrinsic_target() . . . 1049
8.73.2.6 analysis_get_var_ref_resolution() . . . 1049
8.73.2.7 analysis_has_arc_data() . . . 1050
8.73.2.8 analysis_has_errors() . . . 1050
8.73.2.9 analysis_is_arc_type() . . . 1050
8.73.2.10 analysis_locals() . . . 1051
8.73.2.11 analysis_name_resolution() . . . 1051
8.73.2.12 analysis_owned_type_map_put() . . . 1051
8.73.2.13 analysis_result_free() . . . 1051
8.73.2.14 analysis_signature_tables() . . . 1053
8.73.2.15 analysis_source_names() . . . 1053
8.73.2.16 analysis_source_texts() . . . 1053
8.74 compiler/stage2_l0/src/signatures.l0 File Reference . . . 1054
8.74.1 Function Documentation . . . 1055
8.74.1.1 sig_cycle_visit() . . . 1055
8.74.1.2 sig_detect_value_type_cycles() . . . 1055
8.74.1.3 sig_diag_error() . . . 1056
8.74.1.4 sig_enum_variant_vec_temp_free() . . . 1056
8.74.1.5 sig_extract_dependencies() . . . 1056
8.74.1.6 sig_infer_literal_type() . . . 1056
8.74.1.7 sig_key() . . . 1057
8.74.1.8 sig_lookup_enum_info() . . . 1057

8.74.1.9 sig_lookup_func_type() . . . 1058
8.74.1.10 sig_lookup_let_type() . . . 1058
8.74.1.11 sig_lookup_struct_info() . . . 1058
8.74.1.12 sig_make_tables() . . . 1059
8.74.1.13 sig_resolve() . . . 1059
8.74.1.14 sig_resolve_enum() . . . 1059
8.74.1.15 sig_resolve_func() . . . 1060
8.74.1.16 sig_resolve_let() . . . 1060
8.74.1.17 sig_resolve_struct() . . . 1060
8.74.1.18 sig_set_free_opt() . . . 1061
8.74.1.19 sig_struct_field_vec_temp_free() . . . 1061
8.74.1.20 sig_symbol_is_zero_field_enum_variant() . . . 1061
8.74.1.21 sig_top_decl_name() . . . 1061
8.74.1.22 sig_top_decl_span() . . . 1062
8.74.1.23 sig_type_vec_free() . . . 1062
8.74.1.24 sig_type_vec_free_opt() . . . 1062
8.75 compiler/stage2_l0/src/source_paths.l0 File Reference . . . 1062
8.75.1 Function Documentation . . . 1064
8.75.1.1 sp_add_project_root() . . . 1064
8.75.1.2 sp_add_system_root() . . . 1064
8.75.1.3 sp_append_env_sys_roots() . . . 1064
8.75.1.4 sp_apply_default_project_roots() . . . 1064
8.75.1.5 sp_apply_default_sys_roots() . . . 1065
8.75.1.6 sp_apply_default_sys_roots_env() . . . 1065
8.75.1.7 sp_build_from_roots() . . . 1065
8.75.1.8 sp_create() . . . 1065
8.75.1.9 sp_format_root_list() . . . 1066
8.75.1.10 sp_free() . . . 1066
8.75.1.11 sp_is_valid_ident_component() . . . 1066
8.75.1.12 sp_is_valid_module_name() . . . 1066
8.75.1.13 sp_module_relpath() . . . 1067
8.75.1.14 sp_normalize_entry_target() . . . 1067
8.75.1.15 sp_project_root_count() . . . 1067
8.75.1.16 sp_project_root_get() . . . 1068
8.75.1.17 sp_resolve() . . . 1068
8.75.1.18 sp_system_root_count() . . . 1068
8.75.1.19 sp_system_root_get() . . . 1069
8.76 compiler/stage2_l0/src/string_escape.l0 File Reference . . . 1069
8.76.1 Function Documentation . . . 1069
8.76.1.1 se_append_octal_digit() . . . 1069
8.76.1.2 se_append_octal_escape() . . . 1070
8.76.1.3 se_encode_c_string_bytes() . . . 1070

8.76.1.4 se_is_direct_c_byte() . . . 1070
8.76.1.5 se_is_trigraph_suffix() . . . 1071
8.77 compiler/stage2_l0/src/symbols.l0 File Reference . . . 1071
8.77.1 Enumeration Type Documentation . . . 1071
8.77.1.1 SymbolKind . . . 1071
8.77.2 Function Documentation . . . 1072
8.77.2.1 symbol_create() . . . 1072
8.77.2.2 symbol_free() . . . 1072
8.77.2.3 symbol_kind_name() . . . 1072
8.77.2.4 symbol_set_type_clone() . . . 1073
8.78 compiler/stage2_l0/src/tokens.l0 File Reference . . . 1073
8.78.1 Enumeration Type Documentation . . . 1075
8.78.1.1 TokenRecovery . . . 1075
8.78.1.2 TokenType . . . 1075
8.78.2 Function Documentation . . . 1075
8.78.2.1 ident_to_token() . . . 1075
8.78.2.2 is_expr_ending_token_type() . . . 1077
8.78.2.3 is_reserved_ident_word() . . . 1078
8.78.2.4 is_reserved_keyword() . . . 1078
8.78.2.5 token_logical_predecessor_ends_expression() . . . 1078
8.78.2.6 token_recovered() . . . 1079
8.78.2.7 token_recovery_is_some() . . . 1079
8.78.2.8 token_recovery_release_payload() . . . 1079
8.78.2.9 token_recovery_to_string() . . . 1080
8.78.2.10 token_release_payload() . . . 1080
8.78.2.11 token_to_string() . . . 1080
8.78.2.12 tv_create() . . . 1080
8.78.2.13 tv_free() . . . 1081
8.78.2.14 tv_get() . . . 1081
8.78.2.15 tv_len() . . . 1081
8.78.2.16 tv_push() . . . 1081
8.79 compiler/stage2_l0/src/type_resolve.l0 File Reference . . . 1082
8.79.1 Function Documentation . . . 1083
8.79.1.1 tr_ambiguous_type_modules() . . . 1083
8.79.1.2 tr_apply_suffix() . . . 1083
8.79.1.3 tr_diag_error() . . . 1083
8.79.1.4 tr_format_ambiguous_hints() . . . 1084
8.79.1.5 tr_format_ambiguous_modules() . . . 1084
8.79.1.6 tr_key() . . . 1084
8.79.1.7 tr_module_from_path() . . . 1084
8.79.1.8 tr_resolve_alias() . . . 1085
8.79.1.9 tr_resolve_type_ref() . . . 1085

8.79.1.10 tr_symbol_lookup() . . . 1086
8.79.1.11 tr_type_ref_full_name() . . . 1086
8.79.1.12 tr_type_ref_simple_name() . . . 1086
8.80 compiler/stage2_l0/src/types.l0 File Reference . . . 1087
8.80.1 Enumeration Type Documentation . . . 1088
8.80.1.1 TypeKind . . . 1088
8.80.2 Function Documentation . . . 1089
8.80.2.1 enum_info_free() . . . 1089
8.80.2.2 sig_tables_free() . . . 1089
8.80.2.3 struct_info_free() . . . 1089
8.80.2.4 type_builtin_bool() . . . 1089
8.80.2.5 type_builtin_byte() . . . 1090
8.80.2.6 type_builtin_int() . . . 1090
8.80.2.7 type_builtin_string() . . . 1090
8.80.2.8 type_builtin_void() . . . 1090
8.80.2.9 type_clone() . . . 1090
8.80.2.10 type_equals() . . . 1091
8.80.2.11 type_format() . . . 1091
8.80.2.12 type_free() . . . 1091
8.80.2.13 type_is_builtin() . . . 1092
8.80.2.14 type_is_builtin_name() . . . 1092
8.80.2.15 type_new() . . . 1092
8.80.2.16 type_new_builtin() . . . 1093
8.80.2.17 type_new_enum() . . . 1093
8.80.2.18 type_new_func() . . . 1093
8.80.2.19 type_new_null() . . . 1094
8.80.2.20 type_new_nullable() . . . 1094
8.80.2.21 type_new_pointer() . . . 1094
8.80.2.22 type_new_struct() . . . 1094
8.80.2.23 type_param_vec_free() . . . 1095
8.81 compiler/stage2_l0/src/util/diag.l0 File Reference . . . 1095
8.81.1 Enumeration Type Documentation . . . 1096
8.81.1.1 DiagnosticSeverity . . . 1096
8.81.2 Function Documentation . . . 1097
8.81.2.1 dgv_create() . . . 1097
8.81.2.2 dgv_free() . . . 1097
8.81.2.3 dgv_get() . . . 1097
8.81.2.4 dgv_length() . . . 1097
8.81.2.5 dgv_push() . . . 1098
8.81.2.6 diag_copy() . . . 1098
8.81.2.7 diag_count() . . . 1098
8.81.2.8 diag_create() . . . 1098

8.81.2.9 diag_error() . . . 1099
8.81.2.10 diag_free() . . . 1099
8.81.2.11 diag_get() . . . 1099
8.81.2.12 diag_has_errors() . . . 1100
8.81.2.13 diag_note() . . . 1100
8.81.2.14 diag_warn() . . . 1100
8.81.3 Variable Documentation . . . 1101
8.81.3.1 DiagnosticVector . . . 1101
8.82 compiler/stage2_l0/src/util/intset.l0 File Reference . . . 1101
8.82.1 Function Documentation . . . 1102
8.82.1.1 _iset_alloc() . . . 1102
8.82.1.2 _iset_find() . . . 1103
8.82.1.3 _iset_find_insert() . . . 1103
8.82.1.4 _iset_get_key() . . . 1103
8.82.1.5 _iset_get_state() . . . 1104
8.82.1.6 _iset_needs_grow() . . . 1104
8.82.1.7 _iset_rehash() . . . 1104
8.82.1.8 _iset_set_key() . . . 1104
8.82.1.9 _iset_set_state() . . . 1105
8.82.1.10 _iset_slot() . . . 1105
8.82.1.11 iset_add() . . . 1105
8.82.1.12 iset_capacity() . . . 1106
8.82.1.13 iset_clear() . . . 1106
8.82.1.14 iset_create() . . . 1106
8.82.1.15 iset_create_with_capacity() . . . 1106
8.82.1.16 iset_free() . . . 1107
8.82.1.17 iset_has() . . . 1107
8.82.1.18 iset_remove() . . . 1107
8.82.1.19 iset_size() . . . 1108
8.82.2 Variable Documentation . . . 1108
8.82.2.1 ISET_DEFAULT_CAP . . . 1108
8.82.2.2 ISET_EMPTY . . . 1108
8.82.2.3 ISET_OCCUPIED . . . 1108
8.82.2.4 ISET_TOMBSTONE . . . 1108
8.83 compiler/stage2_l0/src/util/log.l0 File Reference . . . 1108
8.83.1 Enumeration Type Documentation . . . 1109
8.83.1.1 LogLevel . . . 1109
8.83.2 Function Documentation . . . 1110
8.83.2.1 log_config_create() . . . 1110
8.83.2.2 log_debug() . . . 1110
8.83.2.3 log_error() . . . 1110
8.83.2.4 log_format_stage() . . . 1110

8.83.2.5 log_info() . . . 1111
8.83.2.6 log_level_from_verbosity() . . . 1111
8.83.2.7 log_level_name() . . . 1111
8.83.2.8 log_msg() . . . 1112
8.83.2.9 log_stage() . . . 1112
8.83.2.10 log_warning() . . . 1112
8.84 compiler/shared/l0/stdlib/std/path.l0 File Reference . . . 1112
8.84.1 Function Documentation . . . 1113
8.84.1.1 _path_ext_with_dot() . . . 1113
8.84.1.2 _path_find_last_sep() . . . 1113
8.84.1.3 basename() . . . 1114
8.84.1.4 has_extension() . . . 1114
8.84.1.5 has_parent() . . . 1114
8.84.1.6 is_absolute() . . . 1115
8.84.1.7 is_sep() . . . 1115
8.84.1.8 join() . . . 1115
8.84.1.9 parent() . . . 1115
8.84.1.10 stem() . . . 1116
8.85 compiler/stage2_l0/src/util/path.l0 File Reference . . . 1116
8.85.1 Function Documentation . . . 1117
8.85.1.1 up_basename() . . . 1117
8.85.1.2 up_find_last_sep() . . . 1117
8.85.1.3 up_has_parent() . . . 1117
8.85.1.4 up_is_absolute() . . . 1117
8.85.1.5 up_is_l0_file() . . . 1118
8.85.1.6 up_is_sep() . . . 1118
8.85.1.7 up_join() . . . 1118
8.85.1.8 up_parent() . . . 1119
8.85.1.9 up_stem() . . . 1119
8.86 compiler/stage2_l0/src/util/strings.l0 File Reference . . . 1119
8.86.1 Function Documentation . . . 1120
8.86.1.1 us_is_utf8_continuation() . . . 1120
8.86.1.2 us_is_valid_utf8() . . . 1120
8.86.1.3 us_split_char() . . . 1121
8.86.1.4 us_split_chars2_non_empty() . . . 1121
8.86.1.5 us_split_is_space_non_empty() . . . 1121
8.86.1.6 us_strip_utf8_bom() . . . 1122
8.86.1.7 us_sv_clone() . . . 1122
8.87 docs-mainpage-latex.md File Reference . . . 1122

Index **1123**

Chapter 1

Dea/L0 API Reference

Generated API reference for the Dea/L0 codebase.

This document is built from source comments and docstrings for:

- Stage 1 Python compiler sources under `compiler/stage1_py/`
- Stage 2 L0 compiler sources under `compiler/stage2_l0/src/`
- Shared L0 standard library modules under `compiler/shared/l0/stdlib/`
- Shared runtime headers under `compiler/shared/runtime/`

Design documents and project documentation are stored in the repository's `docs` directory and are not duplicated in this reference manual.

1.1 About

- Project: Dea/L0, a small safe C-family systems language that compiles to C99.
- Source repository: `github.com/googlielmo/dea-lang`.
- Created and maintained by `gwz` (`@googlielmo`).

1.2 Overview

This reference provides indexed access to the API by directory, namespace, class, and file. Detailed documentation for each entity is provided in the corresponding sections.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

check_trace_log . . . 21
l0_analysis . . . 24
l0_ast . . . 24
l0_ast_printer . . . 26
l0_backend . . . 28
l0_c_emitter . . . 28
l0_compilation . . . 28
l0_context . . . 28
l0_diagnostics . . . 29
l0_driver . . . 30
l0_expr_types . . . 31
l0_internal_error . . . 31
l0_lexer . . . 31
l0_locals . . . 33
l0_logger . . . 34
l0_name_resolver . . . 36
l0_parser . . . 36
l0_paths . . . 37
l0_resolve . . . 37
l0_scope_context . . . 38
l0_signatures . . . 39
l0_string_escape . . . 39
l0_symbols . . . 41
l0_types . . . 41
l0c . . . 43

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

_l0_base_opt . . . 63
_l0_h_string . . . 63
check_trace_log._MessageCollector . . . 64
_rt_alloc_record . . . 67
_rt_alloc_record_cold . . . 69
_rt_ptr_site . . . 70
AnalysisResult . . . 72
l0_analysis.AnalysisResult . . . 74
ArrayBase . . . 78
Backend . . . 81
l0_backend.Backend . . . 83
BuildPreparedInput . . . 131
ByteArray . . . 133
CaseArm . . . 137
CaseElse . . . 140
CCodeBuilder . . . 146
l0_c_emitter.CCodeBuilder . . . 147
CEmitter . . . 150
l0_c_emitter.CEmitter . . . 151
CharBuffer . . . 200
CivilDate . . . 201
CleanupHeaderGuard . . . 202
CliOptions . . . 203
CliParseResult . . . 206
CodegenOptions . . . 207
l0_context.CompilationContext . . . 209
l0_compilation.CompilationUnit . . . 211
CompilerWorkspace . . . 212
DateTime . . . 214
DiagCollector . . . 216
Diagnostic . . . 217
l0_diagnostics.Diagnostic . . . 218
DriverState . . . 221
DriverUnit . . . 222
Duration . . . 225

EnumDecl . . . 225
EnumInfo . . . 228
l0_signatures.EnumInfo . . . 230
EnumVariant . . . 232
EnumVariantInfo . . . 234
l0_signatures.EnumVariantInfo . . . 235
EscapedChar . . . 236
Exception
  l0_driver.ImportCycleError . . . 290
  l0_driver.SourceEncodingError . . . 422
  l0_parser._ParseSyncException . . . 66
ExprArena . . . 240
ExpressionTypeChecker . . . 242
l0_expr_types.ExpressionTypeChecker . . . 245
ExprNode . . . 264
FieldDecl . . . 269
FileInfo . . . 272
FuncDecl . . . 275
FunctionEnv . . . 279
l0_locals.FunctionEnv . . . 281
l0_internal_error.ICELocation . . . 283
ImplParseResult . . . 286
Import . . . 287
IntSet . . . 296
IntStringLinearMap . . . 298
IntStringLinearMapEntry . . . 299
l0_opt_bool . . . 299
l0_opt_byte . . . 300
l0_opt_int . . . 301
l0_opt_string . . . 301
l0_string . . . 302
l0_sys_rt_RtFileInfo . . . 304
l0_sys_rt_RtTimeParts . . . 306
l0_driver.L0Driver . . . 306
LetDecl . . . 312
l0_lexer.Lexer . . . 315
LexerState . . . 321
LinearMapBase . . . 323
LinearMapEntryBase . . . 324
l0_locals.LocalScopeResolver . . . 326
LocalsResult . . . 331
l0_locals.LocalSymbol . . . 332
LocalSymbol . . . 333
LogConfig . . . 334
LoopCleanupScopes . . . 336
LoopFixedPointFlow . . . 338
l0_expr_types.LoopFlowCapture . . . 339
LoopFlowCapture . . . 340
LoopIterationFlow . . . 341
LoopLabels . . . 343
MatchArm . . . 346
MatchArmScopeEntry . . . 347
Module . . . 352
l0_symbols.ModuleEnv . . . 353
ModuleEnv . . . 355
MonotonicTime . . . 357
NameResolution . . . 358
l0_name_resolver.NameResolver . . . 359

- l0_ast.Node . . . 363
  - l0_ast.CaseArm . . . 138
  - l0_ast.CaseElse . . . 141
  - l0_ast.EnumVariant . . . 233
  - l0_ast.Expr . . . 238
    - l0_ast.BinaryOp . . . 125
    - l0_ast.BoolLiteral . . . 128
    - l0_ast.ByteLiteral . . . 134
    - l0_ast.CallExpr . . . 136
    - l0_ast.CastExpr . . . 144
    - l0_ast.FieldAccessExpr . . . 268
    - l0_ast.IndexExpr . . . 291
    - l0_ast.IntLiteral . . . 294
    - l0_ast.NewExpr . . . 361
    - l0_ast.NullLiteral . . . 367
    - l0_ast.ParenExpr . . . 372
    - l0_ast.StringLiteral . . . 438
    - l0_ast.TryExpr . . . 476
    - l0_ast.TypeExpr . . . 482
    - l0_ast.UnaryOp . . . 491
    - l0_ast.VarRef . . . 495
  - l0_ast.FieldDecl . . . 270
  - l0_ast.Import . . . 288
  - l0_ast.MatchArm . . . 344
  - l0_ast.Module . . . 350
  - l0_ast.Param . . . 369
  - l0_ast.Pattern . . . 394
    - l0_ast.VariantPattern . . . 493
    - l0_ast.WildcardPattern . . . 502
  - l0_ast.Stmt . . . 429
    - l0_ast.AssignStmt . . . 79
    - l0_ast.Block . . . 127
    - l0_ast.BreakStmt . . . 130
    - l0_ast.CaseStmt . . . 142
    - l0_ast.ContinueStmt . . . 213
    - l0_ast.DropStmt . . . 223
    - l0_ast.ExprStmt . . . 266
    - l0_ast.ForStmt . . . 273
    - l0_ast.IfStmt . . . 284
    - l0_ast.LetStmt . . . 313
    - l0_ast.MatchStmt . . . 348
    - l0_ast.ReturnStmt . . . 402
    - l0_ast.WhileStmt . . . 500
    - l0_ast.WithStmt . . . 506
  - l0_ast.TopLevelDecl . . . 474
    - l0_ast.EnumDecl . . . 226
    - l0_ast.FuncDecl . . . 277
    - l0_ast.LetDecl . . . 310
    - l0_ast.StructDecl . . . 444
    - l0_ast.TypeAliasDecl . . . 480
  - l0_ast.TypeRef . . . 484
  - l0_ast.WithItem . . . 503
- OSError
  - l0c._TemporarySourceWriteError . . . 71
- Param . . . 371
- l0_parser.Parser . . . 374
- ParseResult . . . 391
- ParserState . . . 392

PatternArena . . . 395
PatternNode . . . 396
PreparedInput . . . 399
QualifiedNameResult . . . 400
RtFileInfo . . . 404
RtTimeParts . . . 405
RuntimeError
    l0_internal_error.InternalCompilerError . . . 292
l0_locals.Scope . . . 406
Scope . . . 407
l0_scope_context.ScopeContext . . . 408
ScopeContext . . . 411
ScopeSlot . . . 413
ScopeVar . . . 414
l0_signatures.SignatureResolver . . . 415
SignatureTables . . . 420
l0_paths.SourceSearchPaths . . . 423
SourceSearchPaths . . . 426
l0_ast.Span . . . 426
Span . . . 427
StmtArena . . . 430
StmtNode . . . 433
StringBuffer . . . 435
StringIntMap . . . 436
StringPtrMap . . . 439
StringSet . . . 441
StringStringLinearMap . . . 442
StringStringLinearMapEntry . . . 443
StructDecl . . . 445
l0_signatures.StructFieldInfo . . . 446
StructFieldInfo . . . 447
l0_signatures.StructInfo . . . 449
StructInfo . . . 450
l0_symbols.Symbol . . . 453
Symbol . . . 454
l0_resolve.SymbolResolution . . . 458
l0_lexer.Token . . . 459
Token . . . 461
TokenVector . . . 473
TopLevelDecl . . . 475
l0_types.Type . . . 477
    l0_types.BuiltinType . . . 132
    l0_types.EnumType . . . 231
    l0_types.FuncType . . . 282
    l0_types.NullType . . . 368
    l0_types.NullableType . . . 366
    l0_types.PointerType . . . 397
    l0_types.StructType . . . 451
Type . . . 478
TypeAliasDecl . . . 481
TypeRef . . . 486
l0_resolve.TypeResolution . . . 487
Unit . . . 492
ValueError
    l0_string_escape.EscapeDecodeError . . . 237
VectorBase . . . 498
WallTime . . . 499
WithItem . . . 504

Enum
l0_analysis.VarRefResolution 497
l0_expr_types.StmtFlow 432
l0_lexer.TokenKind 462
l0_locals.LocalKind 325
l0_resolve.ResolveErrorKind 401
l0_resolve.TypeResolveErrorKind 489
l0_symbols.SymbolKind 456
IntEnum
l0_context.LogLevel 335

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_l0_base_opt
: Base structure for optional types to access `has_value` . . . 63

_l0_h_string
: Heap-allocated L0 string header . . . 63

check_trace_log._MessageCollector
: Count messages while retaining only a bounded leading sample . . . 64

l0_parser._ParseSyncException
: Internal exception used for error recovery during parsing . . . 66

_rt_alloc_record
: Hot allocation fields read by checked pointer cache hits and allocation tracker mutation paths . 67

_rt_alloc_record_cold
: Cold allocation fields used by diagnostics and rare metadata reads . . . 69

_rt_ptr_site
: Per-call-site pointer check cache . . . 70

l0c._TemporarySourceWriteError
: Report a generated-C write failure that also retained its path . . . 71

AnalysisResult
: Represents the AnalysisResult structure . . . 72

l0_analysis.AnalysisResult
: Full front-end analysis result for an entry module . . . 74

ArrayBase
: ArrayBase implements an array of elements of a given size . . . 78

l0_ast.AssignStmt
: An assignment statement . . . 79

Backend
: Represents one Stage 2 backend generation session . . . 81

l0_backend.Backend
: Language-agnostic code generation backend . . . 83

l0_ast.BinaryOp
: A binary operator expression . . . 125

l0_ast.Block
: A block containing a sequence of statements . . . 127

l0_ast.BoolLiteral
: A boolean literal expression . . . 128

l0_ast.BreakStmt
: A break statement . . . 130

BuildPreparedInput
Prepared input bundle for build/run commands . . . 131
l0_types.BuiltinType
A built-in primitive type . . . 132
ByteArray
Byte-specialized fixed-size array wrapper . . . 133
l0_ast.ByteLiteral
A byte literal expression . . . 134
l0_ast.CallExpr
A function or method call expression . . . 136
CaseArm
Represents the CaseArm structure . . . 137
l0_ast.CaseArm
A single arm in a case statement . . . 138
CaseElse
Represents the CaseElse structure . . . 140
l0_ast.CaseElse
The else arm of a case statement . . . 141
l0_ast.CaseStmt
A case statement (switch-like) . . . 142
l0_ast.CastExpr
A type cast expression . . . 144
CCodeBuilder
Represents a line-oriented C source builder with indentation tracking . . . 146
l0_c_emitter.CCodeBuilder
Helper for building C code with indentation tracking . . . 147
CEmitter
Represents the Stage 2 C emitter state . . . 150
l0_c_emitter.CEmitter
C-specific code emitter . . . 151
CharBuffer
CharBuffer allows for efficient building of strings by accumulating individual characters . . . 200
CivilDate
Represents a civil date without time components . . . 201
CleanupHeaderGuard
Linked stack entry for guarded `with` header locals during cleanup-block checking . . . 202
CliOptions
Represents the CliOptions structure . . . 203
CliParseResult
Represents the CliParseResult structure . . . 206
CodegenOptions
Represents backend and emitter options that affect generated C text . . . 207
l0_context.CompilationContext
Cross-cutting compiler options affecting multiple compilation stages . . . 209
l0_compilation.CompilationUnit
A closed set of modules starting from an entry module . . . 211
CompilerWorkspace
Invocation-owned native build/run workspace . . . 212
l0_ast.ContinueStmt
A continue statement . . . 213
DateTime
Represents a broken-down date and time . . . 214
DiagCollector
Represents the DiagCollector structure . . . 216
Diagnostic
Diagnostic represents an error, warning, or note message produced during compilation . . . 217
l0_diagnostics.Diagnostic
Represents a compiler diagnostic (error or warning) . . . 218

DriverState
Represents the DriverState structure . . . 221
DriverUnit
Represents the DriverUnit structure . . . 222
l0_ast.DropStmt
An explicit drop statement for manual memory/resource release . . . 223
Duration
Represents a duration of time in seconds and nanoseconds . . . 225
EnumDecl
Represents the EnumDecl structure . . . 225
l0_ast.EnumDecl
An enum declaration . . . 226
EnumInfo
Represents the EnumInfo structure . . . 228
l0_signatures.EnumInfo
Resolved type information for an enum . . . 230
l0_types.EnumType
A user-defined enum type . . . 231
EnumVariant
Represents the EnumVariant structure . . . 232
l0_ast.EnumVariant
A variant declaration within an enum . . . 233
EnumVariantInfo
Represents the EnumVariantInfo structure . . . 234
l0_signatures.EnumVariantInfo
Information about a single enum variant . . . 235
EscapedChar
Represents the EscapedChar structure . . . 236
l0_string_escape.EscapeDecodeError
Raised when an invalid escape sequence is encountered during decoding . . . 237
l0_ast.Expr
Base class for all expressions . . . 238
ExprArena
Represents the ExprArena structure . . . 240
ExpressionTypeChecker
Represents the ExpressionTypeChecker structure . . . 242
l0_expr_types.ExpressionTypeChecker
Expression-level type checker for L0 . . . 245
ExprNode
Represents the ExprNode structure . . . 264
l0_ast.ExprStmt
A statement consisting of a single expression . . . 266
l0_ast.FieldAccessExpr
A field access expression . . . 268
FieldDecl
Represents the FieldDecl structure . . . 269
l0_ast.FieldDecl
A field declaration within a struct . . . 270
FileInfo
Shared file metadata wrapper around `sys.rt::RtFileInfo` . . . 272
l0_ast.ForStmt
A for loop statement . . . 273
FuncDecl
Represents the FuncDecl structure . . . 275
l0_ast.FuncDecl
A function declaration . . . 277
FunctionEnv
Represents the FunctionEnv structure . . . 279

l0_locals.FunctionEnv
Environment for a single function . . . 281
l0_types.FuncType
A function signature type . . . 282
l0_internal_error.ICELocation
Represents the location where an internal compiler error occurred . . . 283
l0_ast.IfStmt
An if-else statement . . . 284
ImplParseResult
Represents the ImplParseResult structure . . . 286
Import
Represents the Import structure . . . 287
l0_ast.Import
An import declaration . . . 288
l0_driver.ImportCycleError
Raised when a cyclic import is detected . . . 290
l0_ast.IndexExpr
An array indexing expression . . . 291
l0_internal_error.InternalCompilerError
Exception raised when an internal compiler invariant is violated . . . 292
l0_ast.IntLiteral
An integer literal expression . . . 294
IntSet
IntSet is an open-addressing hash set of integers with linear probing . . . 296
IntStringLinearMap
A linear map specialized for integer keys and string values . . . 298
IntStringLinearMapEntry
Entry for an integer-to-string linear map . . . 299
l0_opt_bool
Optional boolean wrapper . . . 299
l0_opt_byte
Optional byte wrapper . . . 300
l0_opt_int
Optional integer wrapper . . . 301
l0_opt_string
Optional string wrapper . . . 301
l0_string
Unified L0 string type (static or heap-allocated) . . . 302
l0_sys_rt_RtFileInfo
Definition for `sys.rt::RtFileInfo` . . . 304
l0_sys_rt_RtTimeParts
Definition for `sys.rt::RtTimeParts` . . . 306
l0_driver.L0Driver
Stage-1 driver for the L0 compiler . . . 306
l0_ast.LetDecl
A top-level 'let' declaration . . . 310
LetDecl
Represents the LetDecl structure . . . 312
l0_ast.LetStmt
A local variable declaration statement . . . 313
l0_lexer.Lexer
Tokenizes L0 source code . . . 315
LexerState
Lexer state object that keeps track of the current position in the source code being lexed . . . 321
LinearMapBase
Base structure for a linear map . . . 323
LinearMapEntryBase
Linear map implementation using linear search . . . 324

l0_locals.LocalKind
Enumeration of the different kinds of local bindings . . . 325
l0_locals.LocalScopeResolver
Builds lexical scopes for functions within a compilation unit . . . 326
LocalsResult
Represents the LocalsResult structure . . . 331
l0_locals.LocalSymbol
Represents a single local binding . . . 332
LocalSymbol
Represents the LocalSymbol structure . . . 333
LogConfig
Represents the LogConfig structure . . . 334
l0_context.LogLevel
Hierarchical logging levels for the L0 compiler . . . 335
LoopCleanupScopes
Represents the cleanup targets for one active loop . . . 336
LoopFixedPointFlow
Result of fixed-point liveness convergence for one loop . . . 338
l0_expr_types.LoopFlowCapture
Liveness states for loop-control exits from one loop body . . . 339
LoopFlowCapture
Captured liveness states for loop-control exits from one loop body . . . 340
LoopIterationFlow
Result of checking one loop iteration for liveness flow . . . 341
LoopLabels
Represents one active loop label pair . . . 343
l0_ast.MatchArm
A single arm in a match statement . . . 344
MatchArm
Represents the MatchArm structure . . . 346
MatchArmScopeEntry
Represents the MatchArmScopeEntry structure . . . 347
l0_ast.MatchStmt
A match statement for pattern matching . . . 348
l0_ast.Module
A complete L0 module . . . 350
Module
Represents the Module structure . . . 352
l0_symbols.ModuleEnv
Per-module symbol environment . . . 353
ModuleEnv
Represents the ModuleEnv structure . . . 355
MonotonicTime
Represents monotonic time in seconds and nanoseconds . . . 357
NameResolution
Represents the NameResolution structure . . . 358
l0_name_resolver.NameResolver
Stage-1 name resolver for module-level symbols . . . 359
l0_ast.NewExpr
A heap allocation or constructor expression . . . 361
l0_ast.Node
Base class for all AST nodes . . . 363
l0_types.NullableType
A nullable (optional) wrapper for another type (T?) . . . 366
l0_ast.NullLiteral
A null literal expression . . . 367
l0_types.NullType
The type of the 'null' literal . . . 368

l0_ast.Param
A function parameter declaration . . . 369
Param
Represents the Param structure . . . 371
l0_ast.ParenExpr
A parenthesized expression . . . 372
l0_parser.Parser
Recursive descent parser for L0 . . . 374
ParseResult
Represents the ParseResult structure . . . 391
ParserState
Represents the ParserState structure . . . 392
l0_ast.Pattern
Base class for match patterns . . . 394
PatternArena
Represents the PatternArena structure . . . 395
PatternNode
Represents the PatternNode structure . . . 396
l0_types.PointerType
A pointer to another type (T∗) . . . 397
PreparedInput
Represents the PreparedInput structure . . . 399
QualifiedNameResult
Represents the QualifiedNameResult structure . . . 400
l0_resolve.ResolveErrorKind
Kinds of errors that can occur during symbol resolution . . . 401
l0_ast.ReturnStmt
A return statement . . . 402
RtFileInfo
Represents basic file metadata used by bootstrap tooling . . . 404
RtTimeParts
Represents components of wall or monotonic time . . . 405
l0_locals.Scope
A lexical scope for local bindings . . . 406
Scope
Represents the Scope structure . . . 407
l0_scope_context.ScopeContext
Tracks variables and cleanup requirements for a single lexical scope . . . 408
ScopeContext
Represents backend cleanup state for a single lexical scope . . . 411
ScopeSlot
Represents the ScopeSlot structure . . . 413
ScopeVar
Represents one declared variable tracked by the backend . . . 414
l0_signatures.SignatureResolver
Resolves top-level type signatures across all modules . . . 415
SignatureTables
Represents the SignatureTables structure . . . 420
l0_driver.SourceEncodingError
Raised when a source file cannot be decoded as UTF-8 . . . 422
l0_paths.SourceSearchPaths
Search configuration for L0 modules . . . 423
SourceSearchPaths
Represents the SourceSearchPaths structure . . . 426
l0_ast.Span
Represents a range of text in a source file . . . 426
Span
Represents the Span structure . . . 427

l0_ast.Stmt
Base class for all statements . . . 429
StmtArena
Represents the StmtArena structure . . . 430
l0_expr_types.StmtFlow
Reachability after checking a statement or block . . . 432
StmtNode
Represents the StmtNode structure . . . 433
StringBuffer
StringBuffer allows for efficient construction of strings by accumulating parts in a vector . . . 435
StringIntMap
StringIntMap stores associations between strings and integers . . . 436
l0_ast.StringLiteral
A string literal expression . . . 438
StringPtrMap
StringPtrMap is an open-addressing hash table with linear probing . . . 439
StringSet
StringSet is an open-addressing hash set with linear probing . . . 441
StringStringLinearMap
A linear map specialized for string keys and string values . . . 442
StringStringLinearMapEntry
Entry for a string-to-string linear map . . . 443
l0_ast.StructDecl
A struct declaration . . . 444
StructDecl
Represents the StructDecl structure . . . 445
l0_signatures.StructFieldInfo
Information about a single struct field . . . 446
StructFieldInfo
Represents the StructFieldInfo structure . . . 447
l0_signatures.StructInfo
Resolved type information for a struct . . . 449
StructInfo
Represents the StructInfo structure . . . 450
l0_types.StructType
A user-defined struct type . . . 451
l0_symbols.Symbol
A resolved module-level symbol . . . 453
Symbol
Represents the Symbol structure . . . 454
l0_symbols.SymbolKind
Enumeration of all top-level symbol categories . . . 456
l0_resolve.SymbolResolution
The result of a symbol resolution attempt . . . 458
l0_lexer.Token
A single token produced by the lexer . . . 459
Token
Represents the Token structure . . . 461
l0_lexer.TokenKind
Enumeration of all supported token types in L0 . . . 462
TokenVector
Represents the TokenVector structure . . . 473
l0_ast.TopLevelDecl
Base class for top-level declarations (functions, structs, etc.) . . . 474
TopLevelDecl
Represents the TopLevelDecl structure . . . 475
l0_ast.TryExpr
A try expression for error handling . . . 476

l0_types.Type
Base class for all semantic types . . . 477
Type
Represents the Type structure . . . 478
l0_ast.TypeAliasDecl
A type alias declaration . . . 480
TypeAliasDecl
Represents the TypeAliasDecl structure . . . 481
l0_ast.TypeExpr
A type used in an expression position . . . 482
l0_ast.TypeRef
Reference to a type in the source code . . . 484
TypeRef
Represents the TypeRef structure . . . 486
l0_resolve.TypeResolution
The result of a type reference resolution attempt . . . 487
l0_resolve.TypeResolveErrorKind
Kinds of errors that can occur during type resolution . . . 489
l0_ast.UnaryOp
A unary operator expression . . . 491
Unit
Definition of the Unit type, representing a value with no information . . . 492
l0_ast.VariantPattern
A pattern that matches an enum variant . . . 493
l0_ast.VarRef
A variable reference expression . . . 495
l0_analysis.VarRefResolution
Enumeration of variable reference resolution targets . . . 497
VectorBase
VectorBase implements a dynamic array (vector) of elements of a given size . . . 498
WallTime
Represents wall clock time in seconds and nanoseconds since the Unix epoch . . . 499
l0_ast.WhileStmt
A while loop statement . . . 500
l0_ast.WildcardPattern
A wildcard pattern ('_') that matches anything . . . 502
l0_ast.WithItem
An item in a with statement . . . 503
WithItem
Represents the WithItem structure . . . 504
l0_ast.WithStmt
A with statement for RAII-style resource management . . . 506

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

compiler/shared/l0/stdlib/std/array.l0 . . . 509
compiler/shared/l0/stdlib/std/assert.l0 . . . 514
compiler/shared/l0/stdlib/std/fs.l0 . . . 515
compiler/shared/l0/stdlib/std/hashmap.l0 . . . 518
compiler/shared/l0/stdlib/std/hashset.l0 . . . 539
compiler/shared/l0/stdlib/std/integer.l0 . . . 550
compiler/shared/l0/stdlib/std/io.l0 . . . 559
compiler/shared/l0/stdlib/std/linear_map.l0 . . . 574
compiler/shared/l0/stdlib/std/optional.l0 . . . 589
compiler/shared/l0/stdlib/std/path.l0 . . . 1112
compiler/shared/l0/stdlib/std/rand.l0 . . . 591
compiler/shared/l0/stdlib/std/string.l0 . . . 593
compiler/shared/l0/stdlib/std/system.l0 . . . 603
compiler/shared/l0/stdlib/std/text.l0 . . . 605
compiler/shared/l0/stdlib/std/time.l0 . . . 619
compiler/shared/l0/stdlib/std/unit.l0 . . . 624
compiler/shared/l0/stdlib/std/vector.l0 . . . 625
compiler/shared/l0/stdlib/sys/hash.l0 . . . 633
compiler/shared/l0/stdlib/sys/memory.l0 . . . 638
compiler/shared/l0/stdlib/sys/rt.l0 . . . 643
compiler/shared/runtime/dea_siphash.h
SipHash-1-3 and SipHash-2-4 (64-bit tag), portable C99 . . . 656
compiler/shared/runtime/l0_runtime.h
L0 Runtime Library (K0 - Kernel Layer) . . . 662
compiler/stage1_py/l0_analysis.py . . . 759
compiler/stage1_py/l0_ast.py . . . 759
compiler/stage1_py/l0_ast_printer.py . . . 762
compiler/stage1_py/l0_backend.py . . . 762
compiler/stage1_py/l0_c_emitter.py . . . 762
compiler/stage1_py/l0_compilation.py . . . 763
compiler/stage1_py/l0_context.py . . . 763
compiler/stage1_py/l0_diagnostics.py . . . 763
compiler/stage1_py/l0_driver.py . . . 764
compiler/stage1_py/l0_expr_types.py . . . 764
compiler/stage1_py/l0_internal_error.py . . . 764

compiler/stage1_py/l0_lexer.py . . . 765
compiler/stage1_py/l0_locals.py . . . 765
compiler/stage1_py/l0_logger.py . . . 766
compiler/stage1_py/l0_name_resolver.py . . . 766
compiler/stage1_py/l0_parser.py . . . 766
compiler/stage1_py/l0_paths.py . . . 767
compiler/stage1_py/l0_resolve.py . . . 767
compiler/stage1_py/l0_scope_context.py . . . 768
compiler/stage1_py/l0_signatures.py . . . 768
compiler/stage1_py/l0_string_escape.py . . . 768
compiler/stage1_py/l0_symbols.py . . . 769
compiler/stage1_py/l0_types.py . . . 769
compiler/stage1_py/l0c.py . . . 770
compiler/stage2_l0/scripts/check_trace_log.py . . . 772
compiler/stage2_l0/src/analysis.l0 . . . 773
compiler/stage2_l0/src/ast.l0 . . . 774
compiler/stage2_l0/src/ast_printer.l0 . . . 792
compiler/stage2_l0/src/backend.l0 . . . 800
compiler/stage2_l0/src/build_driver.l0 . . . 835
compiler/stage2_l0/src/build_info.l0 . . . 849
compiler/stage2_l0/src/c_emitter.l0 . . . 853
compiler/stage2_l0/src/cli_args.l0 . . . 907
compiler/stage2_l0/src/codegen_options.l0 . . . 917
compiler/stage2_l0/src/compiler_filesystem.l0 . . . 918
compiler/stage2_l0/src/diag_print.l0 . . . 923
compiler/stage2_l0/src/driver.l0 . . . 926
compiler/stage2_l0/src/expr_types.l0 . . . 933
compiler/stage2_l0/src/l0c.l0 . . . 964
compiler/stage2_l0/src/l0c_lib.l0 . . . 965
compiler/stage2_l0/src/lexer.l0 . . . 973
compiler/stage2_l0/src/locals.l0 . . . 984
compiler/stage2_l0/src/name_resolver.l0 . . . 993
compiler/stage2_l0/src/parser.l0 . . . 1003
compiler/stage2_l0/src/scope_context.l0 . . . 1043
compiler/stage2_l0/src/sem_context.l0 . . . 1046
compiler/stage2_l0/src/signatures.l0 . . . 1054
compiler/stage2_l0/src/source_paths.l0 . . . 1062
compiler/stage2_l0/src/string_escape.l0 . . . 1069
compiler/stage2_l0/src/symbols.l0 . . . 1071
compiler/stage2_l0/src/tokens.l0 . . . 1073
compiler/stage2_l0/src/type_resolve.l0 . . . 1082
compiler/stage2_l0/src/types.l0 . . . 1087
compiler/stage2_l0/src/parser/decl.l0 . . . 1006
compiler/stage2_l0/src/parser/expr.l0 . . . 1012
compiler/stage2_l0/src/parser/shared.l0 . . . 1017
compiler/stage2_l0/src/parser/stmt.l0 . . . 1036
compiler/stage2_l0/src/util/diag.l0 . . . 1095
compiler/stage2_l0/src/util/intset.l0 . . . 1101
compiler/stage2_l0/src/util/log.l0 . . . 1108
compiler/stage2_l0/src/util/path.l0 . . . 1116
compiler/stage2_l0/src/util/strings.l0 . . . 1119

Chapter 6

Namespace Documentation

6.1 check_trace_log Namespace Reference

Classes

- class _MessageCollector

  Count messages while retaining only a bounded leading sample.

Functions

- tuple[Iterable[dict[str, str]], _MessageCollector, dict[str, int]] _iter_events (Iterable[str] lines, int max_↩ details=20)

  Yield structured trace events from one raw line iterator.
- tuple[_MessageCollector, _MessageCollector, dict[str, int], dict] _validate_events (Iterable[dict[str, str]] events, int max_details=20)

  Validate trace event sequences for definite runtime misuse patterns.
- None _print_report (dict[str, int] counts, _MessageCollector parse_warnings, _MessageCollector errors, _MessageCollector warnings, dict[str, int] op_counts, dict triage, int max_details, bool show_triage)

  Print a validation summary and optional leak triage details.
- argparse.ArgumentParser _build_arg_parser ()

  Build the command-line parser for the trace log checker.
- int main (list[str] argv)

  Run the trace log checker command-line entry point.

Variables

- TRACE_RE = re.compile(r"^\[I0\]\[(mem|arc)\]\s+(.*)$")
- KV_RE = re.compile(r"(\w+)=([^\s]+)")

6.1.1 Function Documentation

6.1.1.1 _build_arg_parser()

```
argparse.ArgumentParser check_trace_log._build_arg_parser ( )  [protected]
```

Build the command-line parser for the trace log checker.

Returns

Configured argument parser for the `check_trace_log.py` CLI.

See also

`main`: Uses this parser to handle CLI arguments.

6.1.1.2 _iter_events()

```
tuple[Iterable[dict[str, str]], _MessageCollector, dict[str, int]] check_trace_log._iter_↩
events (
            Iterable[str] lines,
            int  max_details = 20 )  [protected]
```

Yield structured trace events from one raw line iterator.

Parameters

lines	Raw trace log lines, typically captured from Stage 2 stderr.
max_details	Maximum number of non-fatal parse-warning details to retain.

Returns

A triple containing the parsed event iterator, non-fatal parse warnings, and summary event counts.

See also

`_validate_events`: Consumes the parsed events for semantic checks. `main`: Opens the input file and drives the full validation flow.

6.1.1.3 _print_report()

```
None check_trace_log._print_report (
            dict[str, int] counts,
            _MessageCollector parse_warnings,
            _MessageCollector errors,
            _MessageCollector warnings,
            dict[str, int] op_counts,
            dict triage,
            int max_details,
            bool show_triage )  [protected]
```

Print a validation summary and optional leak triage details.

Parameters

counts	Parsed trace event counts by family and in total.
parse_warnings	Non-fatal warnings produced during parsing.
errors	Validation errors to report.
warnings	Validation warnings to report.
op_counts	Per-family operation counts.
triage	Leak triage metadata from `_validate_events`.
max_details	Maximum number of detail lines to print per section.
show_triage	Whether to print the triage section.

See also

`_validate_events`: Produces the summary and triage data rendered here.

6.1.1.4 _validate_events()

```
tuple[_MessageCollector, _MessageCollector, dict[str, int], dict] check_trace_log._validate_↩
events (
            Iterable[dict[str, str]] events,
            int  max_details = 20 )  [protected]
```

Validate trace event sequences for definite runtime misuse patterns.

Parameters

events	Parsed trace events from `_parse_events`.
max_details	Maximum number of validation error and warning details to retain.

Returns

A tuple of "(errors, warnings, op_counts, triage)" summarizing the validation result and leak triage metadata.

See also

`_parse_events`: Produces the event stream consumed here. `_print_report`: Renders the returned validation summary.

6.1.1.5 main()

```
int check_trace_log.main (
            list[str] argv )
```

Run the trace log checker command-line entry point.

Parameters

argv	Process argument vector, including program name at index 0.

Returns

Exit status `0` on success, `1` when validation finds errors, or `2` for invalid usage or file-read failures.

See also

`_build_arg_parser`: Defines the accepted CLI arguments. `_iter_events`: Parses the raw trace file contents. `_validate_events`: Checks the parsed events for runtime issues. `_print_report`: Prints the final human-readable report.

6.1.2 Variable Documentation

6.1.2.1 KV_RE

```
check_trace_log.KV_RE = re.compile(r"(\w+)=([^\s]+)")
```

6.1.2.2 TRACE_RE

```
check_trace_log.TRACE_RE = re.compile(r"^\[l0\]\[(mem|arc)\]\s+(.*)$")
```

6.2 l0_analysis Namespace Reference

Classes

- class AnalysisResult

  Full front-end analysis result for an entry module.
- class VarRefResolution

  Enumeration of variable reference resolution targets.

6.3 l0_ast Namespace Reference

Classes

- class AssignStmt

  An assignment statement.
- class BinaryOp

  A binary operator expression.
- class Block

  A block containing a sequence of statements.
- class BoolLiteral

  A boolean literal expression.
- class BreakStmt

  A break statement.
- class ByteLiteral

  A byte literal expression.
- class CallExpr

A function or method call expression.

- class CaseArm

  A single arm in a case statement.
- class CaseElse

  The else arm of a case statement.
- class CaseStmt

  A case statement (switch-like).
- class CastExpr

  A type cast expression.
- class ContinueStmt

  A continue statement.
- class DropStmt

  An explicit drop statement for manual memory/resource release.
- class EnumDecl

  An enum declaration.
- class EnumVariant

  A variant declaration within an enum.
- class Expr

  Base class for all expressions.
- class ExprStmt

  A statement consisting of a single expression.
- class FieldAccessExpr

  A field access expression.
- class FieldDecl

  A field declaration within a struct.
- class ForStmt

  A for loop statement.
- class FuncDecl

  A function declaration.
- class IfStmt

  An if-else statement.
- class Import

  An import declaration.
- class IndexExpr

  An array indexing expression.
- class IntLiteral

  An integer literal expression.
- class LetDecl

  A top-level 'let' declaration.
- class LetStmt

  A local variable declaration statement.
- class MatchArm

  A single arm in a match statement.
- class MatchStmt

  A match statement for pattern matching.
- class Module

  A complete L0 module.
- class NewExpr

  A heap allocation or constructor expression.
- class Node

  Base class for all AST nodes.

- class NullLiteral
  A null literal expression.
- class Param
  A function parameter declaration.
- class ParenExpr
  A parenthesized expression.
- class Pattern
  Base class for match patterns.
- class ReturnStmt
  A return statement.
- class Span
  Represents a range of text in a source file.
- class Stmt
  Base class for all statements.
- class StringLiteral
  A string literal expression.
- class StructDecl
  A struct declaration.
- class TopLevelDecl
  Base class for top-level declarations (functions, structs, etc.).
- class TryExpr
  A try expression for error handling.
- class TypeAliasDecl
  A type alias declaration.
- class TypeExpr
  A type used in an expression position.
- class TypeRef
  Reference to a type in the source code.
- class UnaryOp
  A unary operator expression.
- class VariantPattern
  A pattern that matches an enum variant.
- class VarRef
  A variable reference expression.
- class WhileStmt
  A while loop statement.
- class WildcardPattern
  A wildcard pattern ('_') that matches anything.
- class WithItem
  An item in a with statement.
- class WithStmt
  A with statement for RAII-style resource management.

6.4 l0_ast_printer Namespace Reference

Functions

- str _format_span (Span|None span)
  Format a source span for printing.
- List[str] format_node (Any node, int indent=0)
  Generic, reflection-based AST pretty-printer.
- str format_module (Module mod)
  Convenience function to pretty-print a single Module as a string.

6.4.1 Function Documentation

6.4.1.1 _format_span()

```
str l0_ast_printer._format_span (
            Span | None span )  [protected]
```

Format a source span for printing.

Parameters

span	The source span to format, or None.

Returns

A string representation of the span (e.g., " @1:1-2:1"), or an empty string if the span is None.

6.4.1.2 format_module()

```
str l0_ast_printer.format_module (
            Module mod )
```

Convenience function to pretty-print a single Module as a string.

Parameters

mod	The module to format.

Returns

A multiline string representing the formatted module.

6.4.1.3 format_node()

```
List[str] l0_ast_printer.format_node (
            Any node,
            int  indent = 0 )
```

Generic, reflection-based AST pretty-printer.

This function recursively traverses an AST node (or a list of nodes) and produces a human-readable string representation. It uses reflection to discover fields in dataclasses.

- Shows the node class name.
- Prints simple scalar fields inline (excluding `span`).
- Recursively prints child Node or list-of-Node fields on new indented lines.
- Appends a concise span annotation when available.

Parameters

node	The AST node, list of nodes, or other object to format.
indent	The current indentation level (number of 2-space increments).

Returns

A list of strings representing the formatted AST, one line per string.

6.5 l0_backend Namespace Reference

Classes

- class Backend

  Language-agnostic code generation backend.

6.6 l0_c_emitter Namespace Reference

Classes

- class CCodeBuilder

  Helper for building C code with indentation tracking.
- class CEmitter

  C-specific code emitter.

6.7 l0_compilation Namespace Reference

Classes

- class CompilationUnit

  A closed set of modules starting from an entry module.

6.8 l0_context Namespace Reference

Classes

- class CompilationContext

  Cross-cutting compiler options affecting multiple compilation stages.
- class LogLevel

  Hierarchical logging levels for the L0 compiler.

6.9 l0_diagnostics Namespace Reference

Classes

- class Diagnostic

  Represents a compiler diagnostic (error or warning).

Functions

- Diagnostic diag_from_node (str kind, str message, *Optional[str] module_name, Optional[str] filename, Optional[Node] node)

  Create a Diagnostic from an AST node.
- Diagnostic diag_from_token (str kind, str message, *Optional[str] module_name, Optional[str] filename, Optional["Token"] token)

  Create a Diagnostic from a lexer token.

Variables

- dict DIAGNOSTIC_CODE_FAMILIES

6.9.1 Function Documentation

6.9.1.1 diag_from_node()

```
Diagnostic l0_diagnostics.diag_from_node (
            str kind,
            str message,
            *Optional[str] module_name,
            Optional[str] filename,
            Optional[Node] node )
```

Create a Diagnostic from an AST node.

Parameters

kind	The kind of diagnostic ("error" or "warning").
message	The diagnostic message.
module_name	Optional module name.
filename	Optional filename.
node	Optional AST node providing span information.

Returns

A new Diagnostic instance.

6.9.1.2 diag_from_token()

```
Diagnostic l0_diagnostics.diag_from_token (
            str kind,
```

```
            str message,
            *Optional[str] module_name,
            Optional[str] filename,
            Optional["Token"] token )
```

Create a Diagnostic from a lexer token.

Parameters

kind	The kind of diagnostic ("error" or "warning").
message	The diagnostic message.
module_name	Optional module name.
filename	Optional filename.
token	Optional lexer token providing location information.

Returns

A new Diagnostic instance.

6.9.2 Variable Documentation

6.9.2.1 DIAGNOSTIC_CODE_FAMILIES

```
dict l0_diagnostics.DIAGNOSTIC_CODE_FAMILIES
```

6.10 l0_driver Namespace Reference

Classes

- class ImportCycleError

  Raised when a cyclic import is detected.
- class L0Driver

  Stage-1 driver for the L0 compiler.
- class SourceEncodingError

  Raised when a source file cannot be decoded as UTF-8.

Functions

- str load_source_utf8 (str|Path path)

  Read source bytes and decode as UTF-8.

6.10.1 Function Documentation

6.10.1.1 load_source_utf8()

```
str l0_driver.load_source_utf8 (
            str | Path path )
```

Read source bytes and decode as UTF-8.

UTF-8 BOM is accepted and silently stripped.

Parameters

path	Path to the source file to read.

Returns

The decoded text content of the file.

Exceptions

SourceEncodingError	If the file cannot be decoded as UTF-8.
OSError	If there is an error reading the file.

6.11 l0_expr_types Namespace Reference

Classes

- class ExpressionTypeChecker

  Expression-level type checker for L0.
- class LoopFlowCapture

  Liveness states for loop-control exits from one loop body.
- class StmtFlow

  Reachability after checking a statement or block.

6.12 l0_internal_error Namespace Reference

Classes

- class ICELocation

  Represents the location where an internal compiler error occurred.
- class InternalCompilerError

  Exception raised when an internal compiler invariant is violated.

6.13 l0_lexer Namespace Reference

Classes

- class Lexer

  Tokenizes L0 source code.
- class Token

  A single token produced by the lexer.
- class TokenKind

  Enumeration of all supported token types in L0.

Functions

- bool _is_ascii_ident_start (str c)

  Return whether `c` can start an ASCII Dea identifier.
- bool _is_ascii_ident_part (str c)

  Return whether `c` can continue an ASCII Dea identifier.
- bool _logical_predecessor_ends_expression (Token token, bool previous)

  Return whether the parser-visible predecessor ends an expression.
- bool is_reserved_keyword (str word)

  Check if a word is a reserved L0 keyword.

Variables

- dict KEYWORDS
- _EXPR_ENDING_TOKENS
- str OCT_CHARS = "01234567"
- str HEX_CHARS = "0123456789abcdefABCDEF"

6.13.1 Function Documentation

6.13.1.1 _is_ascii_ident_part()

```
bool l0_lexer._is_ascii_ident_part (
            str c )  [protected]
```

Return whether `c` can continue an ASCII Dea identifier.

6.13.1.2 _is_ascii_ident_start()

```
bool l0_lexer._is_ascii_ident_start (
            str c )  [protected]
```

Return whether `c` can start an ASCII Dea identifier.

6.13.1.3 _logical_predecessor_ends_expression()

```
bool l0_lexer._logical_predecessor_ends_expression (
            Token token,
            bool previous )  [protected]
```

Return whether the parser-visible predecessor ends an expression.

6.13.1.4 is_reserved_keyword()

```
bool l0_lexer.is_reserved_keyword (
            str word )
```

Check if a word is a reserved L0 keyword.

Parameters

word	The word to check.

Returns

True if the word is a keyword, False otherwise.

6.13.2 Variable Documentation

6.13.2.1 _EXPR_ENDING_TOKENS

```
l0_lexer._EXPR_ENDING_TOKENS  [protected]
```

Initial value:

```
00001 =  frozenset({
00002     TokenKind.IDENT,
00003     TokenKind.INT,
00004     TokenKind.BYTE,
00005     TokenKind.STRING,
00006     TokenKind.TRUE,
00007     TokenKind.FALSE,
00008     TokenKind.NULL,
00009     TokenKind.RPAREN,
00010     TokenKind.RBRACKET,
00011 })
```

6.13.2.2 HEX_CHARS

```
str l0_lexer.HEX_CHARS = "0123456789abcdefABCDEF"
```

6.13.2.3 KEYWORDS

```
dict l0_lexer.KEYWORDS
```

6.13.2.4 OCT_CHARS

```
str l0_lexer.OCT_CHARS = "01234567"
```

6.14 l0_locals Namespace Reference

Classes

- class FunctionEnv

  Environment for a single function.
- class LocalKind

  Enumeration of the different kinds of local bindings.
- class LocalScopeResolver

  Builds lexical scopes for functions within a compilation unit.
- class LocalSymbol

  Represents a single local binding.
- class Scope

  A lexical scope for local bindings.

6.15 l0_logger Namespace Reference

Functions

- None log (CompilationContext context, LogLevel log_level, str message)

  Log a message according to the current context and level.
- None log_error (CompilationContext context, str message)

  Log an error-level message.
- None log_warning (CompilationContext context, str message)

  Log a warning-level message.
- None log_info (CompilationContext context, str message)

  Log an info-level message.
- None log_debug (CompilationContext context, str message)

  Log a debug-level message.
- None log_stage (CompilationContext context, str stage, Optional[str] module=None)

  Log the start or transition of a compilation stage.

6.15.1 Function Documentation

6.15.1.1 log()

```
None l0_logger.log (
            CompilationContext context,
            LogLevel log_level,
            str message )
```

Log a message according to the current context and level.

Parameters

context	The compilation context containing logging configuration.
log_level	The level of the message to log.
message	The message to log.

6.15.1.2 log_debug()

```
None l0_logger.log_debug (
            CompilationContext context,
            str message )
```

Log a debug-level message.

Parameters

context	The compilation context.
message	The message to log.

6.15.1.3 log_error()

```
None l0_logger.log_error (
            CompilationContext context,
            str message )
```

Log an error-level message.

Parameters

context	The compilation context.
message	The message to log.

6.15.1.4 log_info()

```
None l0_logger.log_info (
            CompilationContext context,
            str message )
```

Log an info-level message.

Parameters

context	The compilation context.
message	The message to log.

6.15.1.5 log_stage()

```
None l0_logger.log_stage (
            CompilationContext context,
            str stage,
            Optional[str]  module = None )
```

Log the start or transition of a compilation stage.

Parameters

context	The compilation context.
stage	The name of the compilation stage (e.g., "Lexing", "Parsing").
module	Optional module name being processed.

6.15.1.6 log_warning()

```
None l0_logger.log_warning (
            CompilationContext context,
            str message )
```

Log a warning-level message.

Parameters

context	The compilation context.
message	The message to log.

6.16 l0_name_resolver Namespace Reference

Classes

- class NameResolver

  Stage-1 name resolver for module-level symbols.

6.17 l0_parser Namespace Reference

Classes

- class _ParseSyncException

  Internal exception used for error recovery during parsing.
- class Parser

  Recursive descent parser for L0.

Functions

- int token_len (Token tok)

  Calculate the display length of a token.

6.17.1 Function Documentation

6.17.1.1 token_len()

```
int l0_parser.token_len (
            Token tok )
```

Calculate the display length of a token.

Parameters

tok	The token to measure.

Returns

The length of the token's text, accounting for literal quotes.

6.18 l0_paths Namespace Reference

Classes

- class SourceSearchPaths

  Search configuration for L0 modules.

6.19 l0_resolve Namespace Reference

Classes

- class ResolveErrorKind

  Kinds of errors that can occur during symbol resolution.
- class SymbolResolution

  The result of a symbol resolution attempt.
- class TypeResolution

  The result of a type reference resolution attempt.
- class TypeResolveErrorKind

  Kinds of errors that can occur during type resolution.

Functions

- bool _is_imported (ModuleEnv current_env, str module_name)

  Check if a module is explicitly imported by the current environment.
- SymbolResolution resolve_symbol (Dict[str, ModuleEnv] module_envs, str current_module, str name, Optional[List[str]] module_path=None, *bool require_import=True)

  Resolve a symbol by name, optionally qualified by module path.
- TypeResolution resolve_type_ref (Dict[str, ModuleEnv] module_envs, str current_module, TypeRef tref, Optional[List[str]] module_path=None, *Optional[Callable[[Symbol], Optional[Type]]] resolve_alias=None, bool require_import=True)

  Resolve an AST TypeRef into a semantic Type.

6.19.1 Function Documentation

6.19.1.1 _is_imported()

```
bool l0_resolve._is_imported (
            ModuleEnv current_env,
            str module_name )  [protected]
```

Check if a module is explicitly imported by the current environment.

6.19.1.2 resolve_symbol()

```
SymbolResolution l0_resolve.resolve_symbol (
            Dict[str, ModuleEnv] module_envs,
            str current_module,
            str name,
            Optional[List[str]]  module_path = None,
            *bool  require_import = True )
```

Resolve a symbol by name, optionally qualified by module path.

Parameters

module_envs	The mapping of module names to their environments.
current_module	The name of the module performing the lookup.
name	The name of the symbol to resolve.
module_path	Optional list of module components for qualified names.
require_import	Whether to require an explicit import for qualified lookups.

Returns

A SymbolResolution object containing the result or error information.

6.19.1.3 resolve_type_ref()

```
TypeResolution l0_resolve.resolve_type_ref (
            Dict[str, ModuleEnv] module_envs,
            str current_module,
            TypeRef tref,
            Optional[List[str]]  module_path = None,
            *Optional[Callable[[Symbol], Optional[Type]]]  resolve_alias = None,
            bool  require_import = True )
```

Resolve an AST TypeRef into a semantic Type.

Parameters

module_envs	The mapping of module names to their environments.
current_module	The name of the module performing the resolution.
tref	The TypeRef AST node.
module_path	Optional list of module components for qualified types.
resolve_alias	Optional callback to resolve type aliases.
require_import	Whether to require an explicit import for qualified lookups.

Returns

A TypeResolution object containing the result or error information.

6.20 l0_scope_context Namespace Reference

Classes

- class ScopeContext

  Tracks variables and cleanup requirements for a single lexical scope.

6.21 l0_signatures Namespace Reference

Classes

- class EnumInfo

  Resolved type information for an enum.
- class EnumVariantInfo

  Information about a single enum variant.
- class SignatureResolver

  Resolves top-level type signatures across all modules.
- class StructFieldInfo

  Information about a single struct field.
- class StructInfo

  Resolved type information for a struct.

6.22 l0_string_escape Namespace Reference

Classes

- class EscapeDecodeError

  Raised when an invalid escape sequence is encountered during decoding.

Functions

- bytes decode_l0_string_token (str text)

  Decode an L0 string literal payload to raw bytes.
- str encode_c_string_bytes (bytes data)

  Encode raw bytes into a C-safe string literal body.

Variables

- str _HEX_CHARS = "0123456789abcdefABCDEF"
- str _OCT_CHARS = "01234567"
- _TRIGRAPH_THIRD_BYTES = frozenset(map(ord, "=/'()!<>-"))
- dict _SIMPLE_ESCAPES

6.22.1 Function Documentation

6.22.1.1 decode_l0_string_token()

```
bytes l0_string_escape.decode_l0_string_token (
            str text )
```

Decode an L0 string literal payload to raw bytes.

Processes C-style escape sequences including hex, octal, and Unicode.

Parameters

text	The string literal content (without surrounding quotes).

Returns

The decoded bytes.

Exceptions

EscapeDecodeError	If a malformed Unicode escape is encountered.

6.22.1.2 encode_c_string_bytes()

```
str l0_string_escape.encode_c_string_bytes (
            bytes data )
```

Encode raw bytes into a C-safe string literal body.

Uses standard escapes for common control characters and fixed-width octal escapes for other non-printable bytes to avoid ambiguity in C.

Parameters

data	The raw bytes to encode.

Returns

A string containing the encoded C literal content (without quotes).

6.22.2 Variable Documentation

6.22.2.1 _HEX_CHARS

```
str l0_string_escape._HEX_CHARS = "0123456789abcdefABCDEF"  [protected]
```

6.22.2.2 _OCT_CHARS

```
str l0_string_escape._OCT_CHARS = "01234567"  [protected]
```

6.22.2.3 _SIMPLE_ESCAPES

```
dict l0_string_escape._SIMPLE_ESCAPES  [protected]
```

Initial value:

```
00001 =  {
00002     "\\": ord("\\"),
00003     "'": ord("'"),
00004     '"': ord('"'),
00005     "?": ord("?"),
00006     "a": 0x07,
00007     "b": 0x08,
00008     "f": 0x0C,
00009     "n": 0x0A,
00010     "r": 0x0D,
00011     "t": 0x09,
00012     "v": 0x0B,
00013 }
```

6.22.2.4 _TRIGRAPH_THIRD_BYTES

```
l0_string_escape._TRIGRAPH_THIRD_BYTES = frozenset(map(ord, "=/'()!<>-"))  [protected]
```

6.23 l0_symbols Namespace Reference

Classes

- class ModuleEnv

  Per-module symbol environment.
- class Symbol

  A resolved module-level symbol.
- class SymbolKind

  Enumeration of all top-level symbol categories.

6.24 l0_types Namespace Reference

Classes

- class BuiltinType

  A built-in primitive type.
- class EnumType

  A user-defined enum type.
- class FuncType

  A function signature type.
- class NullableType

  A nullable (optional) wrapper for another type (T?).
- class NullType

  The type of the 'null' literal.
- class PointerType

  A pointer to another type (T∗).
- class StructType

  A user-defined struct type.
- class Type

  Base class for all semantic types.

Functions

- BuiltinType get_builtin_type (str name)

  Get or create a canonical BuiltinType for a given name.
- NullType get_null_type ()

  Get the canonical NullType instance.
- str format_type (Optional[Type] t)

  Format a type into a human-readable string.

Variables

- tuple L0_PRIMITIVE_TYPES = ("bool", "byte", "int", "string", "void")
- _NULL_TYPE = NullType()

6.24.1 Function Documentation

6.24.1.1 format_type()

```
str l0_types.format_type (
            Optional[Type] t )
```

Format a type into a human-readable string.

Parameters

t	The Type to format, or None.

Returns

A string representation of the type (e.g., "int*", "std::Point?").

6.24.1.2 get_builtin_type()

```
BuiltinType l0_types.get_builtin_type (
            str name )
```

Get or create a canonical BuiltinType for a given name.

Parameters

name	The primitive type name.

Returns

The cached or new BuiltinType instance.

6.24.1.3 get_null_type()

```
NullType l0_types.get_null_type ( )
```

Get the canonical NullType instance.

Returns

The singleton NullType instance.

6.24.2 Variable Documentation

6.24.2.1 _NULL_TYPE

```
l0_types._NULL_TYPE = NullType()  [protected]
```

6.24.2.2 L0_PRIMITIVE_TYPES

```
tuple l0_types.L0_PRIMITIVE_TYPES = ("bool", "byte", "int", "string", "void")
```

6.25 l0c Namespace Reference

Classes

- class _TemporarySourceWriteError

  Report a generated-C write failure that also retained its path.

Functions

- str compiler_identity_text ()

  L0 Compiler (l0c) - Stage 1: Analysis and C Code Generation This is the main entry point for the L0 compiler's stage 1, which includes:
- Tuple[int, bool] _scan_cli_presentation_options (Sequence[str] argv)

  Return fallback verbosity and rich-log settings before --.
- None _emit_verbose_compiler_identity (argparse.Namespace args)

  Emit the compiler identity through the normal verbose logging path.
- None _init_env_defaults ()

  Initialize default environment variables based on L0_HOME.
- List[str] _load_file_lines (str path, Dict[str, List[str]] cache)

  Load lines of a source file, using a cache to avoid redundant reads.
- None _emit_diagnostic (str message)

  Write one normative compiler diagnostic line to standard error.
- None print_diagnostics (AnalysisResult result)

  Print diagnostics from the analysis result.
- None print_diagnostic_list (List[Diagnostic] diagnostics)

  Print a list of diagnostics, using a file cache for source snippets.
- None print_diagnostic_with_snippet (Diagnostic diag, Dict[str, List[str]] file_cache)

Print a single diagnostic, including the source line and a caret.

- bool _is_valid_module_name (str module_name)

  Check if a module name is valid (dot-separated identifiers).
- Optional[SourceSearchPaths] build_search_paths (CompilationContext context, argparse.Namespace args)

  Build source search paths from command-line arguments.
- CompilationContext build_compilation_context (argparse.Namespace args)

  Build a CompilationContext from command-line arguments.
- _run_analysis (argparse.Namespace args)

  Run the analysis pipeline.
- List[str] _get_module_names (argparse.Namespace args, CompilationUnit cu)

  Get the list of module names based on the `--all-modules` flag.
- Optional[str] _find_cc ()

  Find the best available C compiler.
- _compiler_flag_family (str compiler)

  Determine the compiler family for flag selection using a simple heuristic.
- bool _is_windows_host ()

  Return whether the current Python host is Windows.
- str _default_executable_name ()

  Return the default build output name for the current platform.
- list[str] _runtime_include_flags (str flag_family, str runtime_include)

  Return compiler-family-specific runtime include flags.
- list[str] _runtime_library_flags (str flag_family, str runtime_lib)

  Return compiler-family-specific runtime library search-path flags.
- list[str] _output_flags (str flag_family, Path exe_path)

  Return compiler-family-specific output flags.
- Path _validated_temporary_directory ()

  Resolve and validate the host temporary directory.
- None _emit_temporary_source_cleanup_failure (Path c_path)

  Report a compiler-temporary source retained after failed cleanup.
- bool _remove_temporary_c_source (Path c_path)

  Remove a generated-C temporary and report retained recovery state.
- Path _write_temporary_c_source (str c_code)

  Atomically create and write one anonymous generated-C source.
- bool _check_entry_main_for_build (AnalysisResult result, str entry_name)

  Check that the entry module defines a valid 'main' function for build/run.
- bool _validate_runtime_library_path (str runtime_lib_path)

  Validate that the provided runtime library path exists and is a directory.
- List[str] _split_c_options (Optional[str] raw_options)

  Split a raw C options string into individual compiler arguments.
- Optional[str] _get_optimize_flag (str flag_family, List[str] extra_opts)

  Determine the appropriate optimization flag for the compiler family.
- int _compile_generated_c (argparse.Namespace args, CompilationContext context, Path c_path, Path exe↩ _path)

  Invoke the selected host compiler for one generated-C source.
- int cmd_build (argparse.Namespace args)

  Build an executable from an L0 module.
- int cmd_run (argparse.Namespace args)

  Build and run an L0 module.
- int cmd_codegen (argparse.Namespace args)

  Generate C code for a module.
- int cmd_check (argparse.Namespace args)

Run analysis and type checking without code generation.

- int cmd_compile (argparse.Namespace args)

  Report that separate compilation is not implemented in Stage 1 yet.
- int cmd_ast (argparse.Namespace args)

  Pretty-print the parsed AST.
- str _format_token_dump_text (tok)

  Return token payload text for `--tok` output.
- int _dump_tokens_for_file (Path path, bool include_eof)

  Dump lexer tokens for a single file.
- int cmd_tok (argparse.Namespace args)

  Dump lexer tokens.
- int cmd_sym (argparse.Namespace args)

  Dump module-level symbol tables.
- int cmd_type (argparse.Namespace args)

  Dump resolved type information.
- None _add_target_args (argparse.ArgumentParser parser)

  Add target module/file arguments to the parser.
- None _add_all_modules_arg (argparse.ArgumentParser parser)

  Add the –all-modules flag to the parser.
- None _add_runtime_args (argparse.ArgumentParser parser)

  Add runtime-related arguments to the parser.
- None _add_codegen_arg (argparse.ArgumentParser parser)

  Add codegen-related arguments to the parser.
- Tuple[List[str], List[str], bool] _split_cli_and_program_args (Sequence[str] argv)

  Split compiler CLI arguments from program arguments.
- List[str] _normalize_cli_argv (argparse.ArgumentParser parser, Sequence[str] argv)

  Normalize exact short-option spellings before passing them to argparse.
- None _validate_mode_scoped_flags (argparse.ArgumentParser parser, argparse.Namespace args)

  Reject flags that were provided for a mode where they are not valid.
- None _validate_reserved_canonical_flags (argparse.ArgumentParser parser, argparse.Namespace args)

  Reject canonical compiler options whose behavior is not implemented.
- None main (Optional[List[str]] argv=None)

  Parse CLI arguments, dispatch the selected mode, and exit.

Variables

- dict _CLI_LONG_VALUE_OPTIONS
- dict _CLI_NAMESPACED_SHORT_VALUE_OPTIONS
- dict _CLI_CANONICAL_SHORT_VALUE_OPTIONS
- dict _CLI_OTHER_SHORT_VALUE_OPTIONS
- tuple _CLI_VALUE_OPTIONS

6.25.1 Function Documentation

6.25.1.1 _add_all_modules_arg()

```
None l0c._add_all_modules_arg (
            argparse.ArgumentParser parser )  [protected]
```

Add the –all-modules flag to the parser.

Parameters

parser	The argument parser to update.

6.25.1.2 _add_codegen_arg()

```
None l0c._add_codegen_arg (
            argparse.ArgumentParser parser )  [protected]
```

Add codegen-related arguments to the parser.

Parameters

parser	The argument parser to update.

6.25.1.3 _add_runtime_args()

```
None l0c._add_runtime_args (
            argparse.ArgumentParser parser )  [protected]
```

Add runtime-related arguments to the parser.

Parameters

parser	The argument parser to update.

6.25.1.4 _add_target_args()

```
None l0c._add_target_args (
            argparse.ArgumentParser parser )  [protected]
```

Add target module/file arguments to the parser.

Parameters

parser	The argument parser to update.

6.25.1.5 _check_entry_main_for_build()

```
bool l0c._check_entry_main_for_build (
            AnalysisResult result,
            str entry_name )  [protected]
```

Check that the entry module defines a valid 'main' function for build/run.

Parameters

result	The analysis result.
entry_name	The name of the entry module.

Returns

True if a valid 'main' function is found, False otherwise.

6.25.1.6 _compile_generated_c()

```
int l0c._compile_generated_c (
            argparse.Namespace args,
            CompilationContext context,
            Path c_path,
            Path exe_path )  [protected]
```

Invoke the selected host compiler for one generated-C source.

Parameters

args	Parsed build arguments.
context	Active compiler context.
c_path	Closed generated-C source path.
exe_path	Requested executable output path.

Returns

Zero when the host compiler succeeds, otherwise one.

6.25.1.7 _compiler_flag_family()

```
l0c._compiler_flag_family (
            str compiler )  [protected]
```

Determine the compiler family for flag selection using a simple heuristic.

Uses pattern matching to handle cases like "gcc-10" or "clang-14" on Unix and "gcc.exe" or "clang.exe" on Windows.

Parameters

compiler	The compiler command string.

Returns

A string representing the compiler family ("tcc", "gcc", "cc", "msvc", or "unknown").

6.25.1.8 _default_executable_name()

```
str l0c._default_executable_name ( )  [protected]
```

Return the default build output name for the current platform.

6.25.1.9 _dump_tokens_for_file()

```
int l0c._dump_tokens_for_file (
            Path path,
            bool include_eof )  [protected]
```

Dump lexer tokens for a single file.

Parameters

path	Path to the source file.
include_eof	Whether to include the EOF token in the output.

Returns

Exit code (0 for success, non-zero for failure).

6.25.1.10 _emit_diagnostic()

```
None l0c._emit_diagnostic (
            str message )  [protected]
```

Write one normative compiler diagnostic line to standard error.

Parameters

message	Diagnostic text without a trailing newline.

6.25.1.11 _emit_temporary_source_cleanup_failure()

```
None l0c._emit_temporary_source_cleanup_failure (
            Path c_path )  [protected]
```

Report a compiler-temporary source retained after failed cleanup.

Parameters

c_path	Retained temporary generated-C path.

6.25.1.12 _emit_verbose_compiler_identity()

```
None l0c._emit_verbose_compiler_identity (
            argparse.Namespace args )  [protected]
```

Emit the compiler identity through the normal verbose logging path.

6.25.1.13 _find_cc()

```
Optional[str] l0c._find_cc ( )  [protected]
```

Find the best available C compiler.

Used in codegen and build stages if the user didn't specify one explicitly.

The search order is:

1. L0_CC environment variable. 2. Common compiler names in PATH: tcc, gcc, clang, cc (in that order). 3. CC environment variable.

Returns

The compiler command if found, or None if no compiler is found.

6.25.1.14 _format_token_dump_text()

```
str l0c._format_token_dump_text (
             tok )  [protected]
```

Return token payload text for `--tok` output.

6.25.1.15 _get_module_names()

```
List[str] l0c._get_module_names (
            argparse.Namespace args,
            CompilationUnit cu )  [protected]
```

Get the list of module names based on the `--all-modules` flag.

Parameters

args	Parsed command-line arguments.
cu	The compilation unit.

Returns

A list of module names to process.

6.25.1.16 _get_optimize_flag()

```
Optional[str] l0c._get_optimize_flag (
            str flag_family,
            List[str] extra_opts )  [protected]
```

Determine the appropriate optimization flag for the compiler family.

Parameters

flag_family	The detected compiler family string.
extra_opts	List of extra C compiler options.

Returns

The optimization flag string (e.g., '-O1') or None if no suitable flag is found or if the user provided explicit optimization flags.

6.25.1.17 _init_env_defaults()

```
None l0c._init_env_defaults ( )  [protected]
```

Initialize default environment variables based on L0_HOME.

6.25.1.18 _is_valid_module_name()

```
bool l0c._is_valid_module_name (
            str module_name )  [protected]
```

Check if a module name is valid (dot-separated identifiers).

Parameters

module_name	The module name to validate.

Returns

True if the module name is valid, False otherwise.

6.25.1.19 _is_windows_host()

```
bool l0c._is_windows_host ( )  [protected]
```

Return whether the current Python host is Windows.

6.25.1.20 _load_file_lines()

```
List[str] l0c._load_file_lines (
            str path,
            Dict[str, List[str]] cache )  [protected]
```

Load lines of a source file, using a cache to avoid redundant reads.

Parameters

path	Path to the source file.
cache	Cache of file lines keyed by path.

Returns

A list of strings representing the lines of the file.

6.25.1.21 _normalize_cli_argv()

```
List[str] l0c._normalize_cli_argv (
            argparse.ArgumentParser parser,
            Sequence[str]  argv )  [protected]
```

Normalize exact short-option spellings before passing them to argparse.

Namespaced value options support a following value or `=VALUE`, but not a concatenated value. Canonical `-I`, `-L`, and `-l` options accept attached or following values, but not `=VALUE`. Only `-v` may form a cluster.

Parameters

parser	The argument parser for reporting errors.
argv	Compiler arguments before the optional `--` separator.

Returns

A normalized compiler argument list.

6.25.1.22 _output_flags()

```
list[str] l0c._output_flags (
            str flag_family,
            Path exe_path )  [protected]
```

Return compiler-family-specific output flags.

6.25.1.23 _remove_temporary_c_source()

```
bool l0c._remove_temporary_c_source (
            Path c_path )  [protected]
```

Remove a generated-C temporary and report retained recovery state.

Parameters

c_path	Temporary generated-C path to remove.

Returns

True when the path is absent after cleanup, otherwise False.

6.25.1.24 _run_analysis()

```
l0c._run_analysis (
            argparse.Namespace args )  [protected]
```

Run the analysis pipeline.

Parameters

args	Parsed command-line arguments.

Returns

A tuple of (AnalysisResult, CompilationContext, int) where the integer is the suggested exit code.

6.25.1.25 _runtime_include_flags()

```
list[str] l0c._runtime_include_flags (
            str flag_family,
            str runtime_include )  [protected]
```

Return compiler-family-specific runtime include flags.

6.25.1.26 _runtime_library_flags()

```
list[str] l0c._runtime_library_flags (
            str flag_family,
            str runtime_lib )  [protected]
```

Return compiler-family-specific runtime library search-path flags.

6.25.1.27 _scan_cli_presentation_options()

```
Tuple[int, bool] l0c._scan_cli_presentation_options (
            Sequence[str] argv )  [protected]
```

Return fallback verbosity and rich-log settings before `--`.

Values following value-taking options are skipped even when they resemble presentation flags, matching the normalization grammar used for parsing.

Parameters

argv	The complete list of command-line arguments.

Returns

A tuple of (verbosity count, rich-log enabled).

6.25.1.28 _split_c_options()

```
List[str] l0c._split_c_options (
            Optional[str] raw_options )  [protected]
```

Split a raw C options string into individual compiler arguments.

Parameters

raw_options	Raw options string, or None.

Returns

A list of whitespace-delimited compiler option tokens.

6.25.1.29 _split_cli_and_program_args()

```
Tuple[List[str], List[str], bool] l0c._split_cli_and_program_args (
            Sequence[str] argv )  [protected]
```

Split compiler CLI arguments from program arguments.

Parameters

argv	The complete list of command-line arguments.

Returns

A tuple of (compiler_args, program_args, separator_present).

6.25.1.30 _validate_mode_scoped_flags()

```
None l0c._validate_mode_scoped_flags (
            argparse.ArgumentParser parser,
            argparse.Namespace args )  [protected]
```

Reject flags that were provided for a mode where they are not valid.

Parameters

parser	The argument parser for reporting errors.
args	The parsed command-line arguments.

6.25.1.31 _validate_reserved_canonical_flags()

```
None l0c._validate_reserved_canonical_flags (
            argparse.ArgumentParser parser,
            argparse.Namespace  args )  [protected]
```

Reject canonical compiler options whose behavior is not implemented.

Parameters

parser	The argument parser for reporting errors.
args	The parsed command-line arguments.

6.25.1.32 _validate_runtime_library_path()

```
bool l0c._validate_runtime_library_path (
            str runtime_lib_path )  [protected]
```

Validate that the provided runtime library path exists and is a directory.

Parameters

runtime_lib_path	Path to the runtime library directory.

Returns

True if the path is a directory, False otherwise.

6.25.1.33 _validated_temporary_directory()

```
Path l0c._validated_temporary_directory ( )  [protected]
```

Resolve and validate the host temporary directory.

POSIX temporary sources are safe to pass to an external compiler only when every component in the resolved directory hierarchy is owned by root or the effective user. Group- or other-writable components must also carry the sticky bit. Windows relies on the platform temporary directory ACL.

Returns

The resolved temporary directory.

Exceptions

OSError	If the directory cannot be resolved or its POSIX hierarchy is not trusted.

6.25.1.34 _write_temporary_c_source()

```
Path l0c._write_temporary_c_source (
            str c_code )  [protected]
```

Atomically create and write one anonymous generated-C source.

The returned descriptor reserves the path before any generated content is written. Writing through that descriptor avoids reopening an attacker- substituted path between selection and creation.

Parameters

c_code	Generated C source text.

Returns

The temporary source path, closed and ready for the host compiler.

Exceptions

OSError	If the path cannot be created or written.

6.25.1.35 build_compilation_context()

```
CompilationContext l0c.build_compilation_context (
            argparse.Namespace args )
```

Build a CompilationContext from command-line arguments.

Parameters

args	Parsed command-line arguments.

Returns

A new CompilationContext configured based on arguments.

6.25.1.36 build_search_paths()

```
Optional[SourceSearchPaths] l0c.build_search_paths (
            CompilationContext context,
            argparse.Namespace args )
```

Build source search paths from command-line arguments.

Handles entry path parsing and default values from environment variables.

Parameters

context	The compilation context for logging.
args	Parsed command-line arguments.

Returns

A SourceSearchPaths object if successful, or None if the entry name is invalid.

6.25.1.37 cmd_ast()

```
int l0c.cmd_ast (
            argparse.Namespace args )
```

Pretty-print the parsed AST.

By default, prints only the entry module. With –all-modules, prints every module in the compilation unit.

Parameters

args	Parsed command-line arguments.

Returns

Exit code (0 for success, non-zero for failure).

6.25.1.38 cmd_build()

```
int l0c.cmd_build (
            argparse.Namespace args )
```

Build an executable from an L0 module.

Parameters

args	Parsed command-line arguments.

Returns

Exit code (0 for success, non-zero for failure).

6.25.1.39 cmd_check()

```
int l0c.cmd_check (
            argparse.Namespace args )
```

Run analysis and type checking without code generation.

Parameters

args	Parsed command-line arguments.

Returns

Exit code (0 for success, non-zero for failure).

6.25.1.40 cmd_codegen()

```
int l0c.cmd_codegen (
            argparse.Namespace args )
```

Generate C code for a module.

Parameters

args	Parsed command-line arguments.

Returns

Exit code (0 for success, non-zero for failure).

6.25.1.41 cmd_compile()

```
int l0c.cmd_compile (
            argparse.Namespace args )
```

Report that separate compilation is not implemented in Stage 1 yet.

Parameters

args	Parsed command-line arguments.

Returns

Exit code 1.

6.25.1.42 cmd_run()

```
int l0c.cmd_run (
            argparse.Namespace args )
```

Build and run an L0 module.

Parameters

args	Parsed command-line arguments.

Returns

Exit code from the executed program or the build process.

6.25.1.43 cmd_sym()

```
int l0c.cmd_sym (
            argparse.Namespace args )
```

Dump module-level symbol tables.

By default, dumps symbols only for the entry module. With –all-modules, dumps symbols for all modules in the compilation unit.

Parameters

args	Parsed command-line arguments.

Returns

Exit code (0 for success, non-zero for failure).

6.25.1.44 cmd_tok()

```
int l0c.cmd_tok (
            argparse.Namespace args )
```

Dump lexer tokens.

By default, dumps tokens for the entry module only. With –all-modules, dumps tokens for all modules in the compilation unit.

Parameters

args	Parsed command-line arguments.

Returns

Exit code (0 for success, non-zero for failure).

6.25.1.45 cmd_type()

```
int l0c.cmd_type (
            argparse.Namespace args )
```

Dump resolved type information.

Dumps function signatures, struct field types, enum variant payloads, and type aliases.

Parameters

args	Parsed command-line arguments.

Returns

Exit code (0 for success, non-zero for failure).

6.25.1.46 compiler_identity_text()

```
str l0c.compiler_identity_text ( )
```

L0 Compiler (l0c) - Stage 1: Analysis and C Code Generation This is the main entry point for the L0 compiler's stage 1, which includes:

- Command-line interface parsing
  - Environment variable handling for defaults
  - Building the compilation context and search paths

Return the user-facing compiler identity/version text.

6.25.1.47 main()

```
None l0c.main (
            Optional[List[str]]  argv = None )
```

Parse CLI arguments, dispatch the selected mode, and exit.

Parameters

argv	Optional list of arguments to parse. Defaults to sys.argv[1:].

6.25.1.48 print_diagnostic_list()

```
None l0c.print_diagnostic_list (
            List[Diagnostic] diagnostics )
```

Print a list of diagnostics, using a file cache for source snippets.

Parameters

diagnostics	List of diagnostics to print.

6.25.1.49 print_diagnostic_with_snippet()

```
None l0c.print_diagnostic_with_snippet (
            Diagnostic diag,
            Dict[str, List[str]]  file_cache )
```

Print a single diagnostic, including the source line and a caret.

Parameters

diag	The diagnostic to print.
file_cache	Cache of file lines keyed by path.

6.25.1.50 print_diagnostics()

```
None l0c.print_diagnostics (
            AnalysisResult result )
```

Print diagnostics from the analysis result.

Parameters

result	The analysis result containing diagnostics.

6.25.2 Variable Documentation

6.25.2.1 _CLI_CANONICAL_SHORT_VALUE_OPTIONS

```
dict l0c._CLI_CANONICAL_SHORT_VALUE_OPTIONS  [protected]
```

Initial value:

```
00001 =  {
00002     "-I",
00003     "-L",
00004     "-l",
00005 }
```

6.25.2.2 _CLI_LONG_VALUE_OPTIONS

```
dict l0c._CLI_LONG_VALUE_OPTIONS  [protected]
```

Initial value:

```
00001 =  {
00002     "--project-root",
00003     "--sys-root",
00004     "--c-compiler",
00005     "--c-options",
00006     "--c-source",
00007     "--runtime-include",
00008     "--runtime-lib",
00009     "--interface-path",
00010     "--output",
00011 }
```

6.25.2.3 _CLI_NAMESPACED_SHORT_VALUE_OPTIONS

```
dict l0c._CLI_NAMESPACED_SHORT_VALUE_OPTIONS  [protected]
```

Initial value:

```
00001 =  {
00002     "-Rp",
00003     "-Rs",
00004     "-Cc",
00005     "-Co",
00006     "-Cs",
00007     "-Ri",
00008     "-Rl",
00009 }
```

6.25.2.4 _CLI_OTHER_SHORT_VALUE_OPTIONS

```
dict l0c._CLI_OTHER_SHORT_VALUE_OPTIONS  [protected]
```

Initial value:

```
00001 =  {
00002     "-o",
00003 }
```

6.25.2.5 _CLI_VALUE_OPTIONS

```
tuple l0c._CLI_VALUE_OPTIONS  [protected]
```

Initial value:

```
00001 =  (
00002     _CLI_LONG_VALUE_OPTIONS
00003     | _CLI_NAMESPACED_SHORT_VALUE_OPTIONS
00004     | _CLI_CANONICAL_SHORT_VALUE_OPTIONS
00005     | _CLI_OTHER_SHORT_VALUE_OPTIONS
00006 )
```

Chapter 7

Class Documentation

7.1 _l0_base_opt Struct Reference

Base structure for optional types to access `has_value`.

`#include <compiler/shared/runtime/l0_runtime.h>`

Public Attributes

- l0_bool has_value

7.1.1 Detailed Description

Base structure for optional types to access `has_value`.

7.1.2 Member Data Documentation

7.1.2.1 has_value

```
l0_bool _l0_base_opt::has_value
```

The documentation for this struct was generated from the following file:

- compiler/shared/runtime/l0_runtime.h

7.2 _l0_h_string Struct Reference

Heap-allocated L0 string header.

`#include <compiler/shared/runtime/l0_runtime.h>`

Public Attributes

- l0_int refcount

  Reference count for memory management, or INT32_MAX if not reference counted.
- l0_int len

  Length in bytes (must be >= 0)
- char bytes [ ]

  Mutable character data, 0-terminated for C interoperability.

7.2.1 Detailed Description

Heap-allocated L0 string header.

L0 string: length-tracked, reference counted, immutable character sequence. Strings are always length-tracked to prevent out-of-bounds access. An l0_string with len=0 represents an empty string. Data should be NULL for empty strings to maintain consistency, but non-NULL is tolerated. refcount is used for memory management; if refcount == INT32_MAX, the string is not reference counted (e.g. allocated strings). Strings with refcount > 0 are reference counted and should be freed when refcount reaches zero. Strings with refcount == INT32_MAX are not ref-counted, but heap-allocated or empty, and should be freed manually. Strings with refcount == _RT_MEM_SENTINEL have already been freed (double-free detected). Data is null-terminated for C interoperability, but length is authoritative.

7.2.2 Member Data Documentation

7.2.2.1 bytes

```
char _l0_h_string::bytes[]
```

Mutable character data, 0-terminated for C interoperability.

7.2.2.2 len

```
l0_int _l0_h_string::len
```

Length in bytes (must be >= 0)

7.2.2.3 refcount

```
l0_int _l0_h_string::refcount
```

Reference count for memory management, or INT32_MAX if not reference counted.

The documentation for this struct was generated from the following file:

- compiler/shared/runtime/l0_runtime.h

7.3 check_trace_log._MessageCollector Class Reference

Count messages while retaining only a bounded leading sample.

Public Member Functions

- None __init__ (self, int limit)
- None append (self, str message)

Public Attributes

- limit
- total
- messages

7.3.1 Detailed Description

Count messages while retaining only a bounded leading sample.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
None check_trace_log._MessageCollector.__init__ (
             self,
            int limit )
```

7.3.3 Member Function Documentation

7.3.3.1 append()

```
None check_trace_log._MessageCollector.append (
             self,
            str message )
```

7.3.4 Member Data Documentation

7.3.4.1 limit

```
check_trace_log._MessageCollector.limit
```

7.3.4.2 messages

```
check_trace_log._MessageCollector.messages
```

7.3.4.3 total

`check_trace_log._MessageCollector.total`

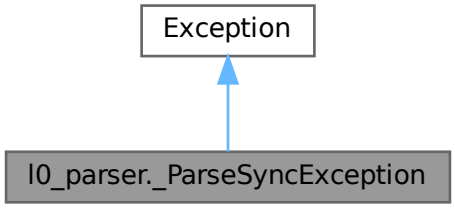
The documentation for this class was generated from the following file:

- compiler/stage2_l0/scripts/check_trace_log.py

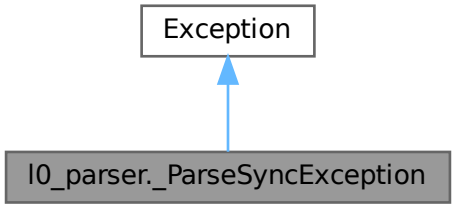
7.4 l0_parser._ParseSyncException Class Reference

Internal exception used for error recovery during parsing.

Inheritance diagram for l0_parser._ParseSyncException:



Collaboration diagram for l0_parser._ParseSyncException:



7.4.1 Detailed Description

Internal exception used for error recovery during parsing.

The documentation for this class was generated from the following file:

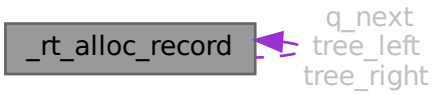
- compiler/stage1_py/l0_parser.py

7.5 _rt_alloc_record Struct Reference

Hot allocation fields read by checked pointer cache hits and allocation tracker mutation paths.

`#include <compiler/shared/runtime/l0_runtime.h>`

Collaboration diagram for _rt_alloc_record:



Public Attributes

- void * base
- size_t size
- uint64_t generation
- _rt_alloc_record * tree_left
- _rt_alloc_record * tree_right
- _rt_alloc_record * q_next
- uint32_t tree_prio
- uint32_t cold_index
- uint8_t state
- uint8_t read_only
- uint8_t mem_kind
- uint8_t hot_pad [25]

7.5.1 Detailed Description

Hot allocation fields read by checked pointer cache hits and allocation tracker mutation paths.

Records are pool-allocated and never freed, so a stale record pointer held by a call-site cache stays dereferenceable; the generation counter rejects reuse.

7.5.2 Member Data Documentation

7.5.2.1 base

```
void* _rt_alloc_record::base
```

7.5.2.2 cold_index

```
uint32_t _rt_alloc_record::cold_index
```

7.5.2.3 generation

```
uint64_t _rt_alloc_record::generation
```

7.5.2.4 hot_pad

```
uint8_t _rt_alloc_record::hot_pad[25]
```

7.5.2.5 mem_kind

```
uint8_t _rt_alloc_record::mem_kind
```

7.5.2.6 q_next

```
_rt_alloc_record* _rt_alloc_record::q_next
```

7.5.2.7 read_only

```
uint8_t _rt_alloc_record::read_only
```

7.5.2.8 size

```
size_t _rt_alloc_record::size
```

7.5.2.9 state

```
uint8_t _rt_alloc_record::state
```

7.5.2.10 tree_left

```
_rt_alloc_record* _rt_alloc_record::tree_left
```

7.5.2.11 tree_prio

```
uint32_t _rt_alloc_record::tree_prio
```

7.5.2.12 tree_right

```
_rt_alloc_record* _rt_alloc_record::tree_right
```

The documentation for this struct was generated from the following file:

- compiler/shared/runtime/l0_runtime.h

7.6 _rt_alloc_record_cold Struct Reference

Cold allocation fields used by diagnostics and rare metadata reads.

`#include <compiler/shared/runtime/l0_runtime.h>`

Public Attributes

- size_t align
- uint32_t type_id
- int alloc_line
- int drop_line
- uint32_t reserved
- const char * alloc_file
- const char * drop_file

7.6.1 Detailed Description

Cold allocation fields used by diagnostics and rare metadata reads.

7.6.2 Member Data Documentation

7.6.2.1 align

```
size_t _rt_alloc_record_cold::align
```

7.6.2.2 alloc_file

```
const char* _rt_alloc_record_cold::alloc_file
```

7.6.2.3 alloc_line

```
int _rt_alloc_record_cold::alloc_line
```

7.6.2.4 drop_file

```
const char* _rt_alloc_record_cold::drop_file
```

7.6.2.5 drop_line

```
int _rt_alloc_record_cold::drop_line
```

7.6.2.6 reserved

```
uint32_t _rt_alloc_record_cold::reserved
```

7.6.2.7 type_id

```
uint32_t _rt_alloc_record_cold::type_id
```

The documentation for this struct was generated from the following file:

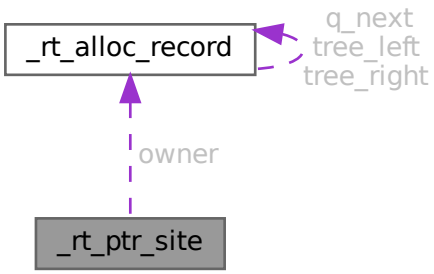
- compiler/shared/runtime/l0_runtime.h

7.7 _rt_ptr_site Struct Reference

Per-call-site pointer check cache.

```
#include <compiler/shared/runtime/l0_runtime.h>
```

Collaboration diagram for _rt_ptr_site:



Public Attributes

- _rt_alloc_record * owner
- uint64_t generation

7.7.1 Detailed Description

Per-call-site pointer check cache.

Generated code declares one static instance per checked access site; a hit validates with one generation compare and one range check, without hashing.

7.7.2 Member Data Documentation

7.7.2.1 generation

```
uint64_t _rt_ptr_site::generation
```

7.7.2.2 owner

```
_rt_alloc_record* _rt_ptr_site::owner
```

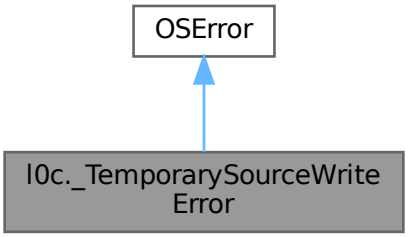
The documentation for this struct was generated from the following file:

- compiler/shared/runtime/l0_runtime.h

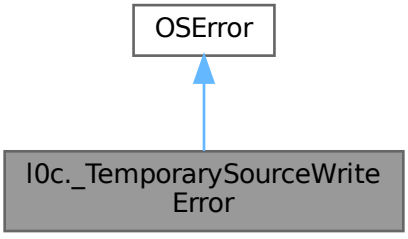
7.8 l0c._TemporarySourceWriteError Class Reference

Report a generated-C write failure that also retained its path.

Inheritance diagram for l0c._TemporarySourceWriteError:



Collaboration diagram for l0c._TemporarySourceWriteError:



Public Member Functions

- __init__ (self, Path retained_path)

  Initialize the failure.

Public Attributes

- retained_path

7.8.1 Detailed Description

Report a generated-C write failure that also retained its path.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 __init__()

```
l0c._TemporarySourceWriteError.__init__ (
              self,
             Path retained_path )
```

Initialize the failure.

Parameters

retained_path	Temporary generated-C path that could not be removed.

7.8.3 Member Data Documentation

7.8.3.1 retained_path

```
l0c._TemporarySourceWriteError.retained_path
```

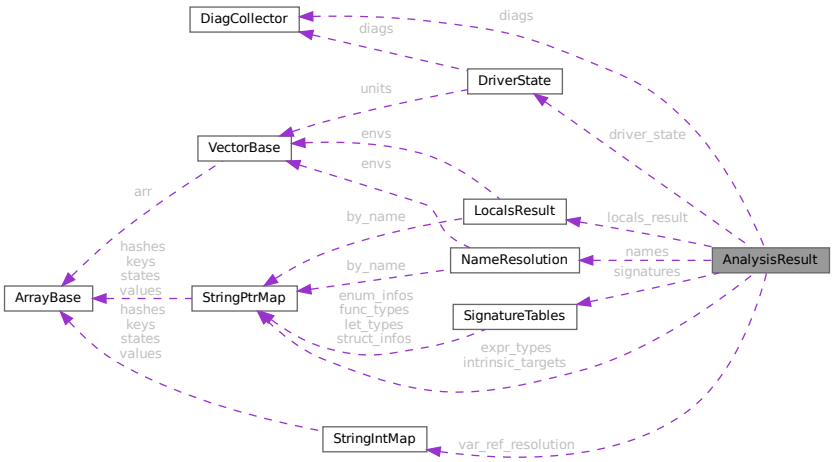
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0c.py

7.9 AnalysisResult Struct Reference

Represents the AnalysisResult structure.

Collaboration diagram for AnalysisResult:



Public Attributes

- driver_state: DriverState∗
- diags: DiagCollector∗
- names: NameResolution∗?
- signatures: SignatureTables∗?
- locals_result: LocalsResult∗?
- expr_types: StringPtrMap∗
- var_ref_resolution: StringIntMap∗
- intrinsic_targets: StringPtrMap∗

7.9.1 Detailed Description

Represents the AnalysisResult structure.

7.9.2 Member Data Documentation

7.9.2.1 diags

diags: `diags: DiagCollector*`

7.9.2.2 driver_state

driver_state: `driver_state: DriverState*`

7.9.2.3 expr_types

expr_types: `expr_types: StringPtrMap*`

7.9.2.4 intrinsic_targets

intrinsic_targets: `intrinsic_targets: StringPtrMap*`

7.9.2.5 locals_result

locals_result: `locals_result: LocalsResult*?`

7.9.2.6 names

names: `names: NameResolution*?`

7.9.2.7 signatures

signatures: `signatures: SignatureTables*?`

7.9.2.8 var_ref_resolution

var_ref_resolution: `var_ref_resolution: StringIntMap*`

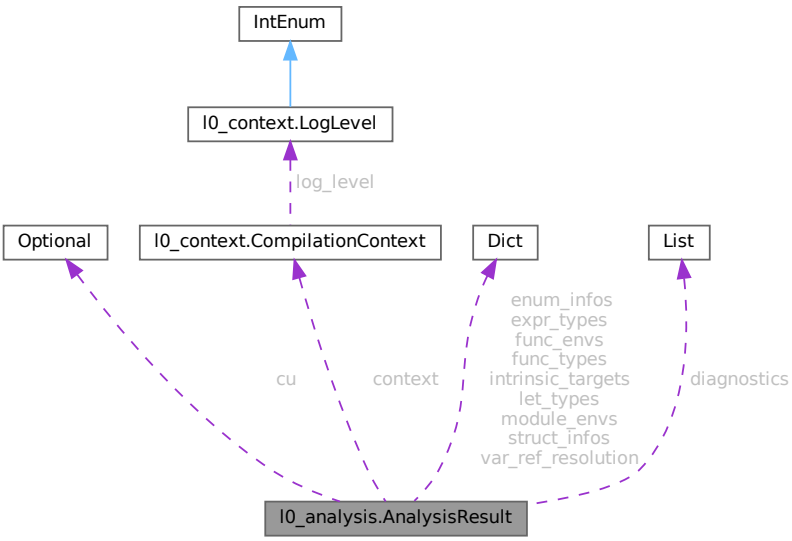
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/sem_context.l0

7.10 l0_analysis.AnalysisResult Class Reference

Full front-end analysis result for an entry module.

Collaboration diagram for l0_analysis.AnalysisResult:



Public Member Functions

- bool has_errors (self)

  Check if any 'error' diagnostics were reported.
- bool has_warnings (self)

  Check if any 'warning' diagnostics were reported.
- bool is_arc_type (self, Type ty)

  Check if a type requires Automatic Reference Counting (ARC).
- bool has_arc_data (self, Type typ)

  Check if a type transitively contains ARC-managed data.

Static Public Attributes

- Optional cu = None
- CompilationContext context = field(default_factory=CompilationContext.default)
- Dict module_envs = field(default_factory=dict)
- Dict func_types = field(default_factory=dict)
- Dict struct_infos = field(default_factory=dict)
- Dict enum_infos = field(default_factory=dict)
- Dict func_envs = field(default_factory=dict)
- Dict let_types = field(default_factory=dict)
- Dict expr_types = field(default_factory=dict)
- Dict var_ref_resolution = field(default_factory=dict)
- Dict intrinsic_targets = field(default_factory=dict)
- List diagnostics = field(default_factory=list)

7.10.1 Detailed Description

Full front-end analysis result for an entry module.

Attributes

- `cu`: The compilation unit containing all modules.
- `context`: Cross-cutting compiler options and configuration.
- `module_envs`: Module-level symbol environments keyed by module name.
- `func_types`: Function signatures keyed by (module_name, decl_name).
- `struct_infos`: Struct definitions keyed by (module_name, decl_name).
- `enum_infos`: Enum definitions keyed by (module_name, decl_name).
- `func_envs`: Function scope information keyed by (module_name, decl_name).
- `let_types`: Types of top-level 'let' declarations keyed by (module_name, decl_name).
- `expr_types`: Resolved types of expressions keyed by id(expr_node).
- `var_ref_resolution`: Resolution targets for VarRef nodes keyed by id(expr_node).
- `intrinsic_targets`: Target types for intrinsic operations keyed by id(expr_node).
- `diagnostics`: List of all diagnostics accumulated during analysis.

7.10.2 Member Function Documentation

7.10.2.1 has_arc_data()

```
bool l0_analysis.AnalysisResult.has_arc_data (
            self,
          Type typ )
```

Check if a type transitively contains ARC-managed data.

This is used to determine if a value of this type requires retain/release orchestration during assignment or when going out of scope.

Parameters

typ	The type to check.

Returns

True if the type or any of its components are managed by ARC, False otherwise.

7.10.2.2 has_errors()

```
bool l0_analysis.AnalysisResult.has_errors (
              self )
```

Check if any 'error' diagnostics were reported.

Returns

True if at least one error exists, False otherwise.

7.10.2.3 has_warnings()

```
bool l0_analysis.AnalysisResult.has_warnings (
              self )
```

Check if any 'warning' diagnostics were reported.

Returns

True if at least one warning exists, False otherwise.

7.10.2.4 is_arc_type()

```
bool l0_analysis.AnalysisResult.is_arc_type (
              self,
             Type ty )
```

Check if a type requires Automatic Reference Counting (ARC).

Currently, only the 'string' builtin type requires ARC.

Parameters

ty	The type to check.

Returns

True if the type is managed by ARC, False otherwise.

7.10.3 Member Data Documentation

7.10.3.1 context

```
CompilationContext l0_analysis.AnalysisResult.context = field(default_factory=CompilationContext.default)
[static]
```

7.10.3.2 cu

```
Optional l0_analysis.AnalysisResult.cu = None  [static]
```

7.10.3.3 diagnostics

```
List l0_analysis.AnalysisResult.diagnostics = field(default_factory=list)  [static]
```

7.10.3.4 enum_infos

```
Dict l0_analysis.AnalysisResult.enum_infos = field(default_factory=dict)  [static]
```

7.10.3.5 expr_types

```
Dict l0_analysis.AnalysisResult.expr_types = field(default_factory=dict)  [static]
```

7.10.3.6 func_envs

```
Dict l0_analysis.AnalysisResult.func_envs = field(default_factory=dict)  [static]
```

7.10.3.7 func_types

```
Dict l0_analysis.AnalysisResult.func_types = field(default_factory=dict)  [static]
```

7.10.3.8 intrinsic_targets

```
Dict l0_analysis.AnalysisResult.intrinsic_targets = field(default_factory=dict)  [static]
```

7.10.3.9 let_types

```
Dict l0_analysis.AnalysisResult.let_types = field(default_factory=dict)  [static]
```

7.10.3.10 module_envs

```
Dict l0_analysis.AnalysisResult.module_envs = field(default_factory=dict)  [static]
```

7.10.3.11 struct_infos

```
Dict l0_analysis.AnalysisResult.struct_infos = field(default_factory=dict)  [static]
```

7.10.3.12 var_ref_resolution

```
Dict l0_analysis.AnalysisResult.var_ref_resolution = field(default_factory=dict)  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_analysis.py

7.11 ArrayBase Struct Reference

ArrayBase implements an array of elements of a given size.

Public Attributes

- capacity: int
- element_size: int
- data: void*

7.11.1 Detailed Description

ArrayBase implements an array of elements of a given size.

Elements can be accessed via index, and the array can be resized on demand. See VectorBase for a dynamic array implementation.

Note: This is a low-level implementation and does not provide type safety. It is the caller's responsibility to manage types and casting.

7.11.2 Member Data Documentation

7.11.2.1 capacity

capacity: `capacity:  int`

7.11.2.2 data

data: `data:  void*`

7.11.2.3 element_size

element_size: `element_size: int`

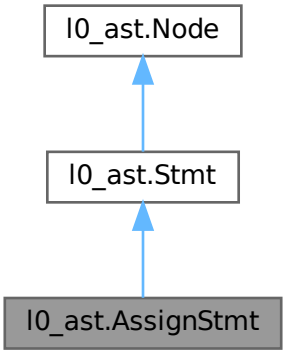
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/array.l0

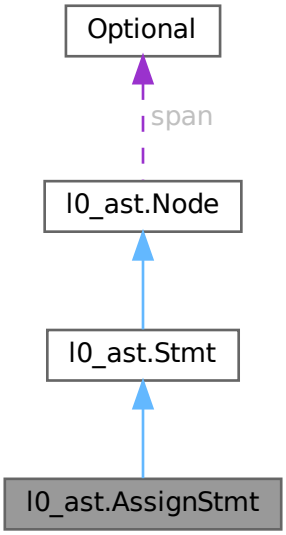
7.12 l0_ast.AssignStmt Class Reference

An assignment statement.

Inheritance diagram for l0_ast.AssignStmt:



Collaboration diagram for l0_ast.AssignStmt:



Static Public Attributes

- str target : "Expr"
- str value : "Expr"

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.12.1 Detailed Description

An assignment statement.

Attributes

- `target`: The target expression (must be an l-value).
- `value`: The expression whose value is being assigned.

7.12.2 Member Data Documentation

7.12.2.1 target

```
str l0_ast.AssignStmt.target :  "Expr"  [static]
```

7.12.2.2 value

```
str l0_ast.AssignStmt.value :  "Expr"  [static]
```

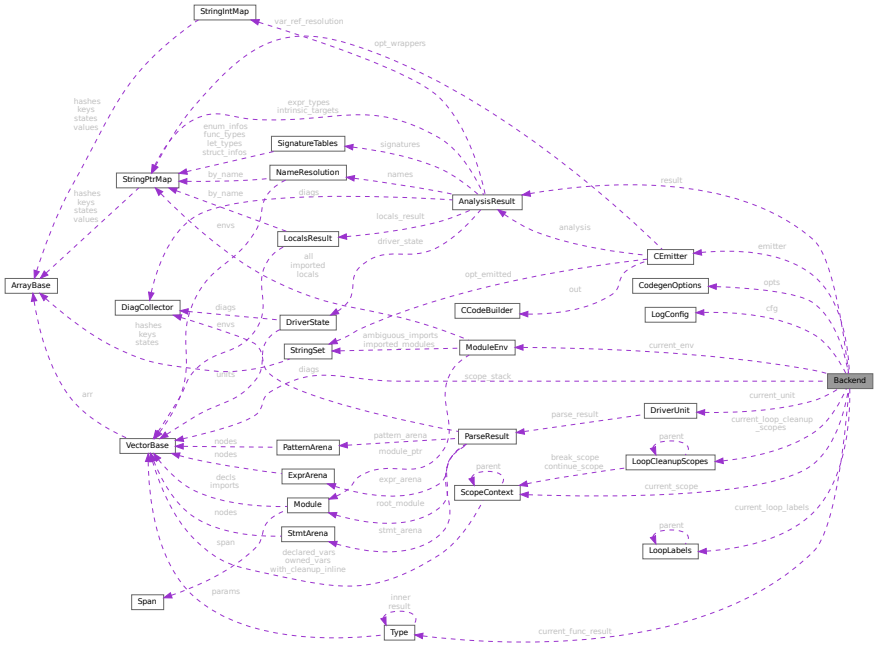
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.13 Backend Struct Reference

Represents one Stage 2 backend generation session.

Collaboration diagram for Backend:



Public Attributes

- result: AnalysisResult*
- opts: CodegenOptions*
- cfg: LogConfig*
- emitter: CEmitter*
- current_module: string?
- current_unit: DriverUnit*?
- current_env: ModuleEnv*?
- current_func_result: Type*?
- current_scope: ScopeContext*?
- scope_stack: VectorBase*
- current_loop_labels: LoopLabels*?
- current_loop_cleanup_scopes: LoopCleanupScopes*?
- scope_counter: int
- label_counter: int
- next_stmt_unreachable: bool

7.13.1 Detailed Description

Represents one Stage 2 backend generation session.

7.13.2 Member Data Documentation

7.13.2.1 cfg

cfg: `cfg: LogConfig*`

7.13.2.2 current_env

current_env: `current_env: ModuleEnv*?`

7.13.2.3 current_func_result

current_func_result: `current_func_result: Type*?`

7.13.2.4 current_loop_cleanup_scopes

current_loop_cleanup_scopes: `current_loop_cleanup_scopes: LoopCleanupScopes*?`

7.13.2.5 current_loop_labels

current_loop_labels: `current_loop_labels: LoopLabels*?`

7.13.2.6 current_module

current_module: `current_module: string?`

7.13.2.7 current_scope

current_scope: `current_scope: ScopeContext*?`

7.13.2.8 current_unit

current_unit: `current_unit: DriverUnit*?`

7.13.2.9 emitter

emitter: `emitter: CEmitter*`

7.13.2.10 label_counter

label_counter: `label_counter: int`

7.13.2.11 next_stmt_unreachable

next_stmt_unreachable: `next_stmt_unreachable: bool`

7.13.2.12 opts

opts: `opts: CodegenOptions*`

7.13.2.13 result

result: `result: AnalysisResult*`

7.13.2.14 scope_counter

scope_counter: `scope_counter: int`

7.13.2.15 scope_stack

scope_stack: `scope_stack: VectorBase*`

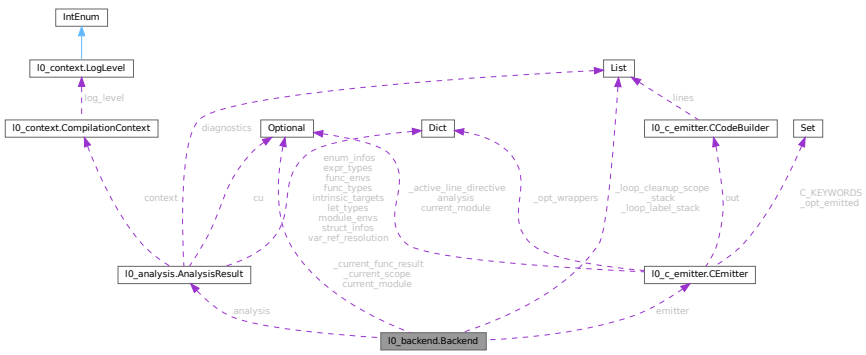
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/backend.l0

7.14 l0_backend.Backend Class Reference

Language-agnostic code generation backend.

Collaboration diagram for l0_backend.Backend:



Public Member Functions

- __post_init__ (self)

  Initialize emitter with analysis data.
- str generate (self)

  Main entry point: generate complete C source for the compilation unit.
- NoReturn ice (self, str message, *Optional[Node] node=None)

  Raise an internal compiler error.
- Optional[EnumVariant] find_variant_decl (self, str module_name, str enum_name, str variant_name)

  Find the EnumVariant AST node for a given variant in an enum.

Public Attributes

- analysis
- current_module

Static Public Attributes

- AnalysisResult analysis
- CEmitter emitter = field(default_factory=CEmitter)
- Optional current_module = None

Protected Member Functions

- str _fresh_label (self, str prefix)

  Generate a unique C label name.
- ScopeContext _push_scope (self)

  Enter a new scope.
- None _pop_scope (self)

  Exit current scope.
- bool _types_equal (self, Type a, Type b)

  Check if two types are structurally equal.
- bool _is_int_assignable (self, Type typ)

  Check if a type is assignable to an integer.
- bool _is_binary_op_enabled (self, Type typ)

  Check if a type supports binary operations.
- bool _is_place_expr (self, Expr expr)

  Check if an expression refers to an existing binding.
- bool _is_unwrap_cast_from_place (self, Expr expr)

  Check if a cast expression still borrows from an existing owner.
- bool _needs_arc_temp (self, Expr expr)

  Check if a non-place rvalue with ARC data needs temp materialization.
- bool _should_materialize_arc_temp (self, Expr expr, Type expr_type)

  Check if an ARC expression should be hoisted to a cleanup temp.
- str _materialize_arc_temp (self, str c_expr, Type expr_type)

  Materialize an ARC rvalue into a scope-owned temporary for automatic cleanup.
- bool _has_side_effects (self, Expr expr)

  Check if the expression has side effects or contains function calls.
- Optional[PointerType] _pointer_type_or_none (self, Optional[Type] ty)

  Return the represented pointer type for pointer-shaped values.

- str _sizeof_expr_for_type (self, Type ty)

  Return a C `sizeof` expression for the runtime access extent.
- str _alignof_expr_for_type (self, Type ty)

  Return a C alignment expression for the runtime access target.
- str _emit_checked_pointer_expr (self, str c_ptr_expr, Type ptr_ty, Optional[Node] node=None, str access_↩ mode="_RT_ACCESS_READ")

  Emit a pointer expression checked for one pointee-sized access.
- str _emit_pointer_index_lvalue (self, str c_base, str c_index, Type base_ty, Optional[Node] node=None, str access_mode="_RT_ACCESS_WRITE")

  Emit a checked pointer-index lvalue expression.
- Optional[Type] _lookup_local_var_type (self, str var_name)

  Look up a local variable's type in the current scope chain.
- Optional[str] _lookup_owned_local_name (self, VarRef expr)

  Return the mangled local name when a VarRef resolves to an owned local binding.
- Set[Tuple[str, str]] _extract_value_type_dependencies (self, Type typ)

  Extract type dependencies for VALUE fields only.
- Dict[Tuple[str, str], Set[Tuple[str, str]]] _build_type_dependency_graph (self)

  Build dependency graph for type definitions.
- str _find_cycle_details (self, Dict[Tuple[str, str], Set[Tuple[str, str]]] graph, List[Tuple[str, str]] unresolved)

  Find and format cycle details for error message.
- List[Tuple[str, str]] _topological_sort (self, Dict[Tuple[str, str], Set[Tuple[str, str]]] graph)

  Perform topological sort on type dependency graph using Kahn's algorithm.
- Optional[StructDecl] _find_struct_decl (self, str module_name, str struct_name)

  Find the StructDecl AST node for a given struct.
- Optional[EnumDecl] _find_enum_decl (self, str module_name, str enum_name)

  Find the EnumDecl AST node for a given enum.
- Type _expect_expr_type (self, Expr expr)

  Look up an expression's type and fail if missing.
- None _emit_line_directive (self, Node node)

  Emit #line directive if node has span info and context allows it.
- None _emit_let_declarations (self)

  Emit static global variables for top-level let declarations.
- None _emit_let_declaration (self, str module_name, LetDecl decl)

  Emit a single top-level let declaration as a static variable.
- str _emit_let_initializer (self, Expr expr, Type expected_type)

  Generate C initializer expression for a top-level let.
- str _emit_bare_variant_static_initializer (self, VarRef expr, Type expected_type)

  Emit a bare zero-argument enum variant for static initialization.
- str _emit_const_constructor (self, CallExpr expr, Type expected_type)

  Emit a constant struct or enum constructor for static initialization.
- str _emit_const_struct_constructor (self, CallExpr expr, StructType struct_type)

  Emit constant struct constructor for static initialization.
- str _emit_const_variant_constructor (self, CallExpr expr, EnumType enum_type)

  Emit constant enum variant constructor for static initialization.
- None _emit_function_declarations (self)

  Emit forward declarations for all functions.
- None _emit_function_declaration (self, str module_name, FuncDecl decl)

  Emit a single function declaration.
- None _emit_function_definitions (self)

  Emit function definitions (bodies).
- None _emit_function_definition (self, str module_name, FuncDecl decl)

Emit a complete function definition with body.

- None _emit_main_wrapper_if_needed (self)

  If the entry module has a main function, emit a C main() wrapper.
- bool _scope_chain_has_cleanup (self)

  Check if any scope in the chain has cleanup requirements.
- None _emit_cleanup_for_return (self, Optional[str] returned_var=None)

  Emit cleanup logic for a return statement.
- None _emit_cleanup_for_loop_exit (self, *bool is_break)

  Emit cleanup for break/continue.
- None _emit_cleanup_at_scope_exit (self, ScopeContext scope)

  Emit cleanup at scope exit.
- None _emit_with_cleanup_from_scope (self, ScopeContext scope, str module_name)

  Emit with-statement cleanup for a scope.
- None _emit_value_cleanup (self, str c_expr, Type ty)

  Emit cleanup code for a by-value variable before reassignment.
- None _emit_struct_cleanup (self, str c_ptr_expr, StructType struct_type)

  Emit cleanup code for all owned fields in a struct.
- None _emit_enum_cleanup (self, str c_ptr_expr, EnumType enum_type)

  Emit cleanup code for owned fields in an enum's active variant.
- None _emit_block_sequence (self, Block block, str module_name)

  Emit statements in a block.
- None _emit_stmt (self, Stmt stmt, str module_name)

  Emit a single statement.
- Any _emit_block (self, Block stmt, str module_name)

  Emit a block statement with its own scope.
- Any _emit_return (self, ReturnStmt stmt, Optional[Callable[[ ], None]] before_cleanup=None)

  Emit a return statement with cleanup.
- None _register_inline_with_cleanup (self, ScopeContext scope, "WithItem" item)

  Register one inline with-item cleanup in LIFO order.
- None _emit_inline_with_header_item (self, "WithItem" item, str module_name, ScopeContext scope)

  Emit one inline with header item and register its cleanup at the committed point.
- None _emit_condition_branch (self, Expr expr, str true_label, str false_label)

  Emit control flow for one condition expression with short-circuit semantics.
- str _emit_condition_expr (self, Expr expr)

  Emit a top-level condition expression for direct statement headers.
- str _emit_condition_value (self, Expr expr)

  Evaluate a statement condition into a stable boolean temporary.
- Any _emit_while (self, WhileStmt stmt, str module_name)

  Emit a while loop.
- Any _emit_for (self, ForStmt stmt, str module_name)

  Emit a for loop.
- Any _emit_if_else (self, IfStmt stmt, str module_name)

  Emit an if-else statement.
- bool _gen_if_else_branch (self, Stmt stmt, str module_name)

  Emit a branch of an if/else.
- _iter_body_stmts (self, Optional[Stmt] stmt)

  Yield every statement reachable from stmt, recursing into block-bearing nodes.
- set _collect_reassigned_arc_params (self, FuncDecl decl, FuncType func_type)

  Collect the names of ARC-typed parameters reassigned syntactically in the body.
- None _emit_reassignment (self, AssignStmt stmt)

  Emit an assignment statement.

- str _emit_lvalue_with_caching (self, Expr target)

  Emit an lvalue expression, caching sub-expressions with side effects.
- Any _emit_let (self, LetStmt stmt, str module_name)

  Emit a local 'let' declaration.
- Type _resolve_let_type (self, LetStmt stmt, str module_name)

  Resolve concrete type for a let declaration.
- Optional[Type] _emit_with_cleanup_header_let_predecl (self, LetStmt stmt, str module_name)

  Predeclare a nullable `with-header` let for cleanup-block form.
- None _emit_with_cleanup_header_let_assign (self, LetStmt stmt, Type var_ty)

  Emit initializer assignment for a predeclared cleanup-block let.
- None _emit_retain_for_copied_value (self, str c_expr, Type ty)

  Emit retain operations for a copied owned value.
- str _emit_copy_expr_with_retains (self, str c_expr, Type ty)

  Materialize copied values in a temp and emit retain logic when needed.
- None _emit_match (self, MatchStmt stmt, str module_name)

  Emit a match statement as a switch on the tag field.
- None _emit_case (self, CaseStmt stmt, str module_name)

  Emit a case statement as a scalar switch or string if/else chain.
- str _emit_case_literal (self, Expr expr)

  Emit a constant literal for a case statement.
- None _emit_pattern_bindings (self, VariantPattern pattern, EnumType enum_type, ScopeContext arm_↩ scope)

  Emit pattern variable bindings and add them to arm scope.
- None _emit_with (self, WithStmt stmt, str module_name)

  Emit a with statement.
- None _emit_drop (self, DropStmt stmt, str module_name)

  Emit drop statement with automatic cleanup of owned fields.
- Optional[str] _try_emit_intrinsic (self, CallExpr expr)

  Expand compiler intrinsics inline.
- str _emit_sizeof_intrinsic (self, CallExpr expr)

  Emit sizeof intrinsic.
- str _emit_ord_intrinsic (self, CallExpr expr)

  Emit ord(enum_value) intrinsic.
- Optional[str] _try_emit_constructor (self, CallExpr expr)

  Check if expr is a constructor call and emit appropriate initialization.
- str _emit_struct_constructor (self, CallExpr expr, StructType struct_type)

  Emit struct constructor as C designated initializer.
- str _emit_variant_constructor (self, CallExpr expr, EnumType enum_type)

  Emit enum variant constructor as C tagged union initializer.
- str _emit_new_expr (self, NewExpr expr)

  Emit a heap allocation `new` expression.
- str _convert_expr_with_expected_type (self, str c_expr, Optional[Type] natural_ty, Type expected)

  Convert a pre-emitted expression into the expected type when required.
- str _emit_expr_with_expected_type (self, Expr e, Type expected)

  Emit expression with implicit type conversion to expected type.
- str _emit_owned_expr_with_expected_type (self, Expr e, Type expected)

  Emit expression for contexts that create a new owner.
- str _emit_expr (self, Expr expr, *bool is_statement=False)

  Emit an expression and return the C code.
- str _emit_unwrap (self, str c_dst, str c_inner, NullableType src_ty)

  Emit code to unwrap a nullable value.

- str _emit_binary_op (self, Expr expr_node, str expr_op, Expr expr_left, Expr expr_right, *bool for_↩ condition=False)

  Emit code for a binary operation.
- _resolve_type_ref (self, TypeRef tref, str module_name)

  Resolve an AST TypeRef into an l0_types.Type.
- Optional[Symbol] _lookup_symbol (self, str name, str current_module_name, Optional[List[str]] module_↩ path=None)

  Look up a symbol in the current module's environment.
- bool _is_extern_function (self, Symbol sym)

  Check if a symbol is an 'extern' function.
- _int_type_size (self, Type src_ty)

  Get the byte size of an integer builtin type.

Protected Attributes

- _current_scope
- _emit_let_initializer
- _next_stmt_unreachable
- _current_func_result

Static Protected Attributes

- Optional _current_func_result = None
- Optional _current_scope = None
- List _loop_cleanup_scope_stack = field(default_factory=list)
- int _switch_depth = 0
- List _loop_label_stack = field(default_factory=list)
- int _label_counter = 0
- bool _next_stmt_unreachable = False

7.14.1 Detailed Description

Language-agnostic code generation backend.

Orchestrates code generation by:

- Managing compilation unit structure and emission order.
- Resolving types and symbols.
- Tracking variable scopes and lifetimes.
- Scheduling cleanup operations.
- Delegating all target-specific code emission to the emitter.

The backend decides WHAT to emit and WHEN, but not HOW (that's the emitter's job). This allows the same backend logic to work with different emitters (C, LLVM IR, WASM, etc.).

Attributes

- `analysis`: Full front-end analysis result.
- `emitter`: Target-specific emitter (handles all code emission).
- `current_module`: Name of the module currently being processed.

7.14.2 Member Function Documentation

7.14.2.1 __post_init__()

```
l0_backend.Backend.__post_init__ (
              self )
```

Initialize emitter with analysis data.

7.14.2.2 _alignof_expr_for_type()

```
str l0_backend.Backend._alignof_expr_for_type (
              self,
             Type ty )  [protected]
```

Return a C alignment expression for the runtime access target.

7.14.2.3 _build_type_dependency_graph()

```
Dict[Tuple[str, str], Set[Tuple[str, str]]] l0_backend.Backend._build_type_dependency_graph (
              self )  [protected]
```

Build dependency graph for type definitions.

A type X depends on type Y if X has a VALUE field of type Y. Pointer fields do NOT create dependencies (forward declarations handle them).

Returns

Dict mapping (module, type_name) → Set of (module, type_name) dependencies.

7.14.2.4 _collect_reassigned_arc_params()

```
set l0_backend.Backend._collect_reassigned_arc_params (
              self,
             FuncDecl decl,
             FuncType func_type )  [protected]
```

Collect the names of ARC-typed parameters reassigned syntactically in the body.

Any function whose body contains an `AssignStmt` whose target is a bare `VarRef` naming an ARC-typed parameter needs a defensive retain on that parameter at entry.

Parameters

decl	The FuncDecl AST node.
func_type	The resolved FuncType for the declaration.

Returns

Set of parameter names (source names, not mangled) to retain at entry.

7.14.2.5 _convert_expr_with_expected_type()

```
str l0_backend.Backend._convert_expr_with_expected_type (
            self,
            str c_expr,
            Optional[Type] natural_ty,
            Type expected )  [protected]
```

Convert a pre-emitted expression into the expected type when required.

Parameters

c_expr	The C expression string.
natural_ty	The natural type of the expression.
expected	The expected type.

Returns

A C expression string, potentially wrapped or widened.

7.14.2.6 _emit_bare_variant_static_initializer()

```
str l0_backend.Backend._emit_bare_variant_static_initializer (
            self,
            VarRef expr,
            Type expected_type )  [protected]
```

Emit a bare zero-argument enum variant for static initialization.

7.14.2.7 _emit_binary_op()

```
str l0_backend.Backend._emit_binary_op (
            self,
            Expr expr_node,
            str expr_op,
            Expr expr_left,
            Expr expr_right,
            *bool  for_condition = False )  [protected]
```

Emit code for a binary operation.

Parameters

expr_node	The BinaryOp AST node.
expr_op	The operator string.
expr_left	The left operand.
expr_right	The right operand.
for_condition	Whether to preserve condition-context code generation.

Returns

C code string for the operation.

Exceptions

InternalCompilerError	If types are missing, mismatch, or operation is unsupported.

7.14.2.8 _emit_block()

```
Any l0_backend.Backend._emit_block (
              self,
             Block stmt,
             str module_name )  [protected]
```

Emit a block statement with its own scope.

Parameters

stmt	The Block AST node.
module_name	Name of current module.

7.14.2.9 _emit_block_sequence()

```
None l0_backend.Backend._emit_block_sequence (
              self,
             Block block,
             str module_name )  [protected]
```

Emit statements in a block.

Parameters

block	The Block AST node.
module_name	Name of current module.

7.14.2.10 _emit_case()

```
None l0_backend.Backend._emit_case (
              self,
             CaseStmt stmt,
             str module_name )  [protected]
```

Emit a case statement as a scalar switch or string if/else chain.

Parameters

stmt	The CaseStmt AST node.
module_name	Name of current module.

Exceptions

InternalCompilerError	If types or literals are unsupported.

7.14.2.11 _emit_case_literal()

```
str l0_backend.Backend._emit_case_literal (
              self,
             Expr expr )  [protected]
```

Emit a constant literal for a case statement.

Parameters

expr	The literal expression.

Returns

A C constant string.

Exceptions

InternalCompilerError	If literal type is unsupported.

7.14.2.12 _emit_checked_pointer_expr()

```
str l0_backend.Backend._emit_checked_pointer_expr (
              self,
             str c_ptr_expr,
             Type ptr_ty,
             Optional[Node]  node = None,
             str  access_mode = "_RT_ACCESS_READ" )  [protected]
```

Emit a pointer expression checked for one pointee-sized access.

7.14.2.13 _emit_cleanup_at_scope_exit()

```
None l0_backend.Backend._emit_cleanup_at_scope_exit (
              self,
             ScopeContext scope )  [protected]
```

Emit cleanup at scope exit.

Only cleans variables declared in THIS scope that have owned fields.

Parameters

scope	The scope being exited.

7.14.2.14 _emit_cleanup_for_loop_exit()

```
None l0_backend.Backend._emit_cleanup_for_loop_exit (
             self,
            *bool is_break )  [protected]
```

Emit cleanup for break/continue.

Walks from current scope up to and including the innermost loop cleanup target, executing any with-statement cleanup data along the way. The with-cleanup runs before owned-var cleanup (see `_emit_cleanup_for_return` for rationale).

Parameters

is_break	True if cleaning for 'break', False for 'continue'.

Exceptions

InternalCompilerError	If called outside of a loop.

7.14.2.15 _emit_cleanup_for_return()

```
None l0_backend.Backend._emit_cleanup_for_return (
             self,
            Optional[str]  returned_var = None )  [protected]
```

Emit cleanup logic for a return statement.

Walks up scope chain, executes any with-statement cleanup data, then cleans ALL owned variables (except return value). The with-cleanup runs first because user cleanup code may reference variables whose owned resources (e.g. string refcounts) are released by the automatic owned-var cleanup.

Parameters

returned_var	Mangled name of variable being returned (to skip cleanup).

7.14.2.16 _emit_condition_branch()

```
None l0_backend.Backend._emit_condition_branch (
             self,
            Expr expr,
            str true_label,
            str false_label )  [protected]
```

Emit control flow for one condition expression with short-circuit semantics.

This path is used only for statement conditions so ARC temps emitted by expression lowering stay inside the correct structural block instead of being hoisted into an enclosing "if (...)" or "while (...)" header.

Parameters

expr	Condition expression to lower.
true_label	Jump target when the condition is true.
false_label	Jump target when the condition is false.

7.14.2.17 _emit_condition_expr()

```
str l0_backend.Backend._emit_condition_expr (
              self,
             Expr expr )  [protected]
```

Emit a top-level condition expression for direct statement headers.

7.14.2.18 _emit_condition_value()

```
str l0_backend.Backend._emit_condition_value (
              self,
             Expr expr )  [protected]
```

Evaluate a statement condition into a stable boolean temporary.

The returned temp is safe to reference from an `if/while header` because any ARC temps created while evaluating the condition are scoped to the emitted condition block and cleaned before control continues.

Parameters

expr	Condition expression to lower.

Returns

Name of the generated boolean temp.

7.14.2.19 _emit_const_constructor()

```
str l0_backend.Backend._emit_const_constructor (
              self,
             CallExpr expr,
             Type expected_type )  [protected]
```

Emit a constant struct or enum constructor for static initialization.

Similar to _try_emit_constructor but only handles constant expressions.

Parameters

expr	The constructor call expression.
expected_type	The expected struct or enum type.

Returns

A C initializer string.

Exceptions

InternalCompilerError	If symbol or constructor type is invalid.

7.14.2.20 _emit_const_struct_constructor()

```
str l0_backend.Backend._emit_const_struct_constructor (
              self,
             CallExpr expr,
             StructType struct_type )  [protected]
```

Emit constant struct constructor for static initialization.

Parameters

expr	The constructor call expression.
struct_type	The struct type.

Returns

A C struct initializer string.

Exceptions

InternalCompilerError	If struct info is missing or argument count mismatches.

7.14.2.21 _emit_const_variant_constructor()

```
str l0_backend.Backend._emit_const_variant_constructor (
              self,
             CallExpr expr,
             EnumType enum_type )  [protected]
```

Emit constant enum variant constructor for static initialization.

Parameters

expr	The variant call expression.
enum_type	The enum type.

Returns

A C enum variant initializer string.

Exceptions

InternalCompilerError	If variant info or declaration is missing.

7.14.2.22 _emit_copy_expr_with_retains()

```
str l0_backend.Backend._emit_copy_expr_with_retains (
              self,
             str c_expr,
             Type ty )  [protected]
```

Materialize copied values in a temp and emit retain logic when needed.

Parameters

c_expr	C expression evaluating to the value.
ty	The type of the value.

Returns

The name of the temporary containing the copied and retained value.

7.14.2.23 _emit_drop()

```
None l0_backend.Backend._emit_drop (
              self,
             DropStmt stmt,
             str module_name )  [protected]
```

Emit drop statement with automatic cleanup of owned fields.

For structs: releases all string fields. For enums: switches on tag, releases strings in active variant. Then calls the drop-finish helper to release the memory.

Parameters

stmt	The DropStmt AST node.
module_name	Name of current module.

Exceptions

InternalCompilerError	If variable is undefined or not a pointer.

7.14.2.24 _emit_enum_cleanup()

```
None l0_backend.Backend._emit_enum_cleanup (
              self,
```

```
            str c_ptr_expr,
            EnumType enum_type )  [protected]
```

Emit cleanup code for owned fields in an enum's active variant.

Uses switch on tag to only clean up the fields that are actually present.

Parameters

c_ptr_expr	C expression evaluating to a pointer to the enum.
enum_type	The enum type.

7.14.2.25 _emit_expr()

```
str l0_backend.Backend._emit_expr (
             self,
            Expr expr,
            *bool  is_statement = False )  [protected]
```

Emit an expression and return the C code.

Parameters

expr	The expression to emit.
is_statement	True if the expression is used as a statement.

Returns

A C expression string, or empty string if used as a statement and no-op.

Exceptions

InternalCompilerError	If expression type is unsupported or resolution fails.

7.14.2.26 _emit_expr_with_expected_type()

```
str l0_backend.Backend._emit_expr_with_expected_type (
             self,
            Expr e,
            Type expected )  [protected]
```

Emit expression with implicit type conversion to expected type.

Parameters

e	The expression to emit.
expected	The expected type.

Returns

A C expression string.

Exceptions

InternalCompilerError	If null literal is assigned to invalid type.

7.14.2.27 _emit_for()

```
Any l0_backend.Backend._emit_for (
              self,
             ForStmt stmt,
             str module_name )  [protected]
```

Emit a for loop.

Parameters

stmt	The ForStmt AST node.
module_name	Name of current module.

7.14.2.28 _emit_function_declaration()

```
None l0_backend.Backend._emit_function_declaration (
              self,
             str module_name,
             FuncDecl decl )  [protected]
```

Emit a single function declaration.

Parameters

module_name	Name of the module.
decl	The FuncDecl AST node.

Exceptions

InternalCompilerError	If FuncType is missing.

7.14.2.29 _emit_function_declarations()

```
None l0_backend.Backend._emit_function_declarations (
              self )  [protected]
```

Emit forward declarations for all functions.

7.14.2.30 _emit_function_definition()

```
None l0_backend.Backend._emit_function_definition (
              self,
             str module_name,
             FuncDecl decl )  [protected]
```

Emit a complete function definition with body.

Parameters

module_name	Name of the module.
decl	The FuncDecl AST node.

Exceptions

InternalCompilerError	If scope is not reset or FuncType is missing.

7.14.2.31 _emit_function_definitions()

```
None l0_backend.Backend._emit_function_definitions (
              self )  [protected]
```

Emit function definitions (bodies).

7.14.2.32 _emit_if_else()

```
Any l0_backend.Backend._emit_if_else (
              self,
             IfStmt stmt,
             str module_name )  [protected]
```

Emit an if-else statement.

Parameters

stmt	The IfStmt AST node.
module_name	Name of current module.

7.14.2.33 _emit_inline_with_header_item()

```
None l0_backend.Backend._emit_inline_with_header_item (
              self,
             "WithItem" item,
             str module_name,
             ScopeContext scope )  [protected]
```

Emit one inline with header item and register its cleanup at the committed point.

7.14.2.34 _emit_let()

```
Any l0_backend.Backend._emit_let (
              self,
             LetStmt stmt,
             str module_name )  [protected]
```

Emit a local 'let' declaration.

Parameters

stmt	The LetStmt AST node.
module_name	Name of current module.

Exceptions

InternalCompilerError	If type cannot be inferred.

7.14.2.35 _emit_let_declaration()

```
None l0_backend.Backend._emit_let_declaration (
              self,
             str module_name,
             LetDecl decl )  [protected]
```

Emit a single top-level let declaration as a static variable.

Parameters

module_name	Name of the module containing the declaration.
decl	The LetDecl AST node.

7.14.2.36 _emit_let_declarations()

```
None l0_backend.Backend._emit_let_declarations (
              self )  [protected]
```

Emit static global variables for top-level let declarations.

7.14.2.37 _emit_let_initializer()

```
str l0_backend.Backend._emit_let_initializer (
              self,
             Expr expr,
             Type expected_type )  [protected]
```

Generate C initializer expression for a top-level let.

Supports compile-time constant literals and struct/enum construction.

Parameters

expr	The initializer expression.
expected_type	The expected type of the constant.

Returns

A C initializer expression string.

Exceptions

InternalCompilerError	If the initializer is not constant or supported.

7.14.2.38 _emit_line_directive()

```
None l0_backend.Backend._emit_line_directive (
              self,
             Node node )  [protected]
```

Emit #line directive if node has span info and context allows it.

Parameters

node	The AST node containing span information.

7.14.2.39 _emit_lvalue_with_caching()

```
str l0_backend.Backend._emit_lvalue_with_caching (
              self,
             Expr target )  [protected]
```

Emit an lvalue expression, caching sub-expressions with side effects.

For targets like ∗(func_call()), the pointer expression `func_call()` must be evaluated exactly once, not multiple times during release/assign/retain.

Parameters

target	The lvalue expression.

Returns

A C lvalue expression string.

Exceptions

InternalCompilerError	If types are missing for complex lvalues.

7.14.2.40 _emit_main_wrapper_if_needed()

```
None l0_backend.Backend._emit_main_wrapper_if_needed (
            self )  [protected]
```

If the entry module has a main function, emit a C main() wrapper.

This allows us to consistently mangle all L0 functions (including main) while still providing the expected C entry point.

7.14.2.41 _emit_match()

```
None l0_backend.Backend._emit_match (
             self,
            MatchStmt stmt,
            str module_name )  [protected]
```

Emit a match statement as a switch on the tag field.

Parameters

stmt	The MatchStmt AST node.
module_name	Name of current module.

Exceptions

InternalCompilerError	If types or patterns are unsupported.

7.14.2.42 _emit_new_expr()

```
str l0_backend.Backend._emit_new_expr (
             self,
            NewExpr expr )  [protected]
```

Emit a heap allocation `new` expression.

Parameters

expr	The NewExpr AST node.

Returns

A C expression string for the newly allocated pointer.

Exceptions

InternalCompilerError	If type or symbol resolution fails, or if args mismatch.

7.14.2.43 _emit_ord_intrinsic()

```
str l0_backend.Backend._emit_ord_intrinsic (
              self,
             CallExpr expr )  [protected]
```

Emit ord(enum_value) intrinsic.

Returns 0-based ordinal of enum variant.

Parameters

expr	The ord call expression.

Returns

A C expression string for the ordinal value.

Exceptions

InternalCompilerError	If argument count is incorrect.

7.14.2.44 _emit_owned_expr_with_expected_type()

```
str l0_backend.Backend._emit_owned_expr_with_expected_type (
              self,
             Expr e,
             Type expected )  [protected]
```

Emit expression for contexts that create a new owner.

This applies retain-on-copy when a place expression is copied into an owned destination, while delegating regular type conversion to `_emit_expr_with_expected_type`.

Parameters

e	The expression to emit.
expected	The expected type.

Returns

A C expression string.

7.14.2.45 _emit_pattern_bindings()

```
None l0_backend.Backend._emit_pattern_bindings (
              self,
             VariantPattern pattern,
```

```
            EnumType enum_type,
            ScopeContext  arm_scope )  [protected]
```

Emit pattern variable bindings and add them to arm scope.

Parameters

pattern	The variant pattern.
enum_type	The enum type.
arm_scope	The scope for the match arm.

Exceptions

InternalCompilerError	If variant decl is missing.

7.14.2.46 _emit_pointer_index_lvalue()

```
str l0_backend.Backend._emit_pointer_index_lvalue (
             self,
            str c_base,
            str c_index,
            Type base_ty,
            Optional[Node]  node = None,
            str  access_mode = "_RT_ACCESS_WRITE" )  [protected]
```

Emit a checked pointer-index lvalue expression.

7.14.2.47 _emit_reassignment()

```
None l0_backend.Backend._emit_reassignment (
             self,
            AssignStmt stmt )  [protected]
```

Emit an assignment statement.

Parameters

stmt	The AssignStmt AST node.

Exceptions

InternalCompilerError	If types are missing or emission fails.

7.14.2.48 _emit_retain_for_copied_value()

```
None l0_backend.Backend._emit_retain_for_copied_value (
             self,
```

```
            str c_expr,
            Type ty )  [protected]
```

Emit retain operations for a copied owned value.

Used when copying from place expressions so source and destination own independent references.

Parameters

c_expr	C expression evaluating to the value.
ty	The type of the value.

Exceptions

InternalCompilerError	If variant decl is missing for enum types.

7.14.2.49 _emit_return()

```
Any l0_backend.Backend._emit_return (
             self,
            ReturnStmt stmt,
            Optional[Callable[[], None]]  before_cleanup = None )  [protected]
```

Emit a return statement with cleanup.

Parameters

stmt	The ReturnStmt AST node.
before_cleanup	Optional hook to run after the return value is evaluated and before scope cleanup is emitted.

7.14.2.50 _emit_sizeof_intrinsic()

```
str l0_backend.Backend._emit_sizeof_intrinsic (
             self,
            CallExpr expr )  [protected]
```

Emit sizeof intrinsic.

Parameters

expr	The sizeof call expression.

Returns

A C sizeof expression string.

Exceptions

InternalCompilerError	If target type cannot be resolved.

7.14.2.51 _emit_stmt()

```
None l0_backend.Backend._emit_stmt (
              self,
             Stmt stmt,
             str module_name )  [protected]
```

Emit a single statement.

Parameters

stmt	The Stmt AST node.
module_name	Name of current module.

Exceptions

InternalCompilerError	If statement type is unsupported.

7.14.2.52 _emit_struct_cleanup()

```
None l0_backend.Backend._emit_struct_cleanup (
              self,
             str c_ptr_expr,
             StructType struct_type )  [protected]
```

Emit cleanup code for all owned fields in a struct.

Recursively handles nested structs (by-value fields).

Parameters

c_ptr_expr	C expression evaluating to a pointer to the struct.
struct_type	The struct type.

7.14.2.53 _emit_struct_constructor()

```
str l0_backend.Backend._emit_struct_constructor (
              self,
             CallExpr expr,
             StructType struct_type )  [protected]
```

Emit struct constructor as C designated initializer.

Point(1, 2) → (struct l0_modulename_Point){ .x = 1, .y = 2 }

Parameters

expr	The constructor call expression.
struct_type	The struct type.

Returns

A C struct initializer expression string.

Exceptions

InternalCompilerError	If struct info is missing or argument count mismatches.

7.14.2.54 _emit_unwrap()

```
str l0_backend.Backend._emit_unwrap (
              self,
             str c_dst,
             str c_inner,
             NullableType src_ty )  [protected]
```

Emit code to unwrap a nullable value.

Parameters

c_dst	C type of destination.
c_inner	C expression of nullable value.
src_ty	The NullableType.

Returns

C code for unwrapped value.

7.14.2.55 _emit_value_cleanup()

```
None l0_backend.Backend._emit_value_cleanup (
              self,
             str c_expr,
             Type ty )  [protected]
```

Emit cleanup code for a by-value variable before reassignment.

Similar to _emit_field_cleanup, but expects c_expr to be a direct value reference (not a pointer), so uses '.' instead of '→'.

Parameters

c_expr	C expression for the value (e.g., "x__v", "obj.field")
ty	The type of the value being cleaned up

7.14.2.56 _emit_variant_constructor()

```
str l0_backend.Backend._emit_variant_constructor (
              self,
             CallExpr expr,
             EnumType enum_type )  [protected]
```

Emit enum variant constructor as C tagged union initializer.

Example: Int(42) → (struct l0_modulename_Int){ .tag = l0_modulename_Int_Int, .data.Int.value = 42 }

Parameters

expr	The variant call expression.
enum_type	The enum type.

Returns

A C variant initializer expression string.

Exceptions

InternalCompilerError	If variant info or declaration is missing.

7.14.2.57 _emit_while()

```
Any l0_backend.Backend._emit_while (
              self,
             WhileStmt stmt,
             str module_name )  [protected]
```

Emit a while loop.

Parameters

stmt	The WhileStmt AST node.
module_name	Name of current module.

7.14.2.58 _emit_with()

```
None l0_backend.Backend._emit_with (
              self,
             WithStmt stmt,
             str module_name )  [protected]
```

Emit a with statement.

Inline => form (LIFO cleanup): Emit init statements, then body, then cleanup statements in reverse order.

Cleanup block form: Emit init statements, then body, then cleanup block statements.

Cleanup is emitted at block end and before every early exit (return, break, continue). The scope stores cleanup data so `_emit_cleanup_for_return` and `_emit_cleanup_for_loop_exit` can emit it before leaving.

The body and cleanup block are each emitted as real nested C blocks so that any declarations inside them do not collide with the header scope (e.g., legal L0 shadowing like "let x" in both the header and body).

Parameters

stmt	The WithStmt AST node.
module_name	Name of current module.

7.14.2.59 _emit_with_cleanup_from_scope()

```
None l0_backend.Backend._emit_with_cleanup_from_scope (
              self,
             ScopeContext scope,
             str module_name )  [protected]
```

Emit with-statement cleanup for a scope.

Parameters

scope	The scope containing cleanup logic.
module_name	Name of current module.

7.14.2.60 _emit_with_cleanup_header_let_assign()

```
None l0_backend.Backend._emit_with_cleanup_header_let_assign (
              self,
             LetStmt stmt,
             Type var_ty )  [protected]
```

Emit initializer assignment for a predeclared cleanup-block let.

Parameters

stmt	The LetStmt AST node.
var_ty	The resolved type of the let.

7.14.2.61 _emit_with_cleanup_header_let_predecl()

```
Optional[Type] l0_backend.Backend._emit_with_cleanup_header_let_predecl (
              self,
             LetStmt stmt,
             str module_name )  [protected]
```

Predeclare a nullable `with-header` let for cleanup-block form.

Nullable lets are predeclared as `null` so cleanup code can reference them on header ? failure paths.

Non-nullable lets use the normal declaration+initializer path and return None here.

Parameters

stmt	The LetStmt AST node.
module_name	Name of current module.

Returns

The Type if it was a nullable let, otherwise None.

7.14.2.62 _expect_expr_type()

```
Type l0_backend.Backend._expect_expr_type (
            self,
           Expr expr )  [protected]
```

Look up an expression's type and fail if missing.

Parameters

expr	The expression to look up.

Returns

The resolved Type of the expression.

Exceptions

InternalCompilerError	If the type is missing from the analysis.

7.14.2.63 _extract_value_type_dependencies()

```
Set[Tuple[str, str]] l0_backend.Backend._extract_value_type_dependencies (
            self,
           Type typ )  [protected]
```

Extract type dependencies for VALUE fields only.

Value-type fields create dependencies (types must be fully defined). Pointer-type fields do NOT create dependencies (forward declarations suffice).

Examples:

- StructType("main", "Point") → {("main", "Point")}
- EnumType("main", "Status") → {("main", "Status")}
- PointerType(StructType("main", "Node")) → {} (no dependency, forward decl works)
- NullableType(PointerType(...)) → {} (pointer-optional, no dependency)
- NullableType(BuiltinType("int")) → {} (value-optional of builtin, no dependency)
- NullableType(StructType("main", "Point")) → {("main", "Point")} (value-optional of struct)
- BuiltinType("int") → {} (no dependency)

Parameters

typ	The type to extract dependencies from.

Returns

Set of (module, name) tuples for types that must be defined first.

7.14.2.64 _find_cycle_details()

```
str l0_backend.Backend._find_cycle_details (
              self,
             Dict[Tuple[str, str], Set[Tuple[str, str]]] graph,
             List[Tuple[str, str]]  unresolved )  [protected]
```

Find and format cycle details for error message.

Parameters

graph	The type dependency graph.
unresolved	List of unresolved nodes.

Returns

A string describing the detected cycle details.

7.14.2.65 _find_enum_decl()

```
Optional[EnumDecl] l0_backend.Backend._find_enum_decl (
              self,
             str module_name,
             str enum_name )  [protected]
```

Find the EnumDecl AST node for a given enum.

Parameters

module_name	Name of the module.
enum_name	Name of the enum.

Returns

The EnumDecl if found, otherwise None.

7.14.2.66 _find_struct_decl()

```
Optional[StructDecl] l0_backend.Backend._find_struct_decl (
            self,
            str module_name,
            str struct_name )  [protected]
```

Find the StructDecl AST node for a given struct.

Parameters

module_name	Name of the module.
struct_name	Name of the struct.

Returns

The StructDecl if found, otherwise None.

7.14.2.67 _fresh_label()

```
str l0_backend.Backend._fresh_label (
            self,
            str prefix )  [protected]
```

Generate a unique C label name.

Parameters

prefix	Prefix for the label name.

Returns

A unique label string.

7.14.2.68 _gen_if_else_branch()

```
bool l0_backend.Backend._gen_if_else_branch (
            self,
            Stmt stmt,
            str module_name )  [protected]
```

Emit a branch of an if/else.

Parameters

stmt	The statement in the branch.
module_name	Name of current module.

Returns

True if branch is unreachable at end.

7.14.2.69 _has_side_effects()

```
bool l0_backend.Backend._has_side_effects (
              self,
             Expr expr )  [protected]
```

Check if the expression has side effects or contains function calls.

Such expressions should be evaluated once and cached in a temporary to avoid multiple evaluation when used in contexts like assignment with ARC operations.

Parameters

expr	The expression to check.

Returns

True if the expression has potential side effects.

7.14.2.70 _int_type_size()

```
l0_backend.Backend._int_type_size (
              self,
             Type src_ty )  [protected]
```

Get the byte size of an integer builtin type.

Parameters

src↩ _ty	The type to check.

Returns

Byte size (1 for byte, 4 for int).

Exceptions

InternalCompilerError	If type is not an integer builtin.

7.14.2.71 _is_binary_op_enabled()

```
bool l0_backend.Backend._is_binary_op_enabled (
              self,
             Type typ )  [protected]
```

Check if a type supports binary operations.

Currently only int, byte, and bool support binary operations.

Parameters

typ	The type to check.

Returns

True if binary operations are supported for the type.

7.14.2.72 _is_extern_function()

```
bool l0_backend.Backend._is_extern_function (
              self,
             Symbol sym )  [protected]
```

Check if a symbol is an 'extern' function.

Parameters

sym	The symbol to check.

Returns

True if it is an extern function.

7.14.2.73 _is_int_assignable()

```
bool l0_backend.Backend._is_int_assignable (
              self,
             Type typ )  [protected]
```

Check if a type is assignable to an integer.

Parameters

typ	The type to check.

Returns

True if it's an 'int' or 'byte' builtin type.

7.14.2.74 _is_place_expr()

```
bool l0_backend.Backend._is_place_expr (
              self,
             Expr expr )  [protected]
```

Check if an expression refers to an existing binding.

Parameters

expr	The expression to check.

Returns

True if expr refers to an existing binding (retain on copy). False if expr produces a fresh value (ownership transfer, no retain).

7.14.2.75 _is_unwrap_cast_from_place()

```
bool l0_backend.Backend._is_unwrap_cast_from_place (
              self,
             Expr expr )  [protected]
```

Check if a cast expression still borrows from an existing owner.

Outer parentheses are ownership-transparent. Owner-producing ARC value-optional wraps are excluded.

Parameters

expr	The expression to check.

Returns

True for non-owner-producing casts whose source is a place.

7.14.2.76 _iter_body_stmts()

```
l0_backend.Backend._iter_body_stmts (
              self,
             Optional[Stmt] stmt )  [protected]
```

Yield every statement reachable from stmt, recursing into block-bearing nodes.

Parameters

stmt	The root statement (or None).

Returns

An iterator over every `Stmt` in the sub-tree, including `stmt` itself.

7.14.2.77 _lookup_local_var_type()

```
Optional[Type] l0_backend.Backend._lookup_local_var_type (
              self,
             str var_name )  [protected]
```

Look up a local variable's type in the current scope chain.

Searches declared_vars (includes both locals and parameters).

Parameters

var_name	The name of the variable to look up.

Returns

The variable's Type, or None if not found.

7.14.2.78 _lookup_owned_local_name()

```
Optional[str] l0_backend.Backend._lookup_owned_local_name (
              self,
            VarRef expr )  [protected]
```

Return the mangled local name when a VarRef resolves to an owned local binding.

Parameters are local VarRefs but are not owned by the callee, so they do not appear in owned_vars and return None.

Parameters

expr	The variable reference expression.

Returns

The mangled local name if it's an owned binding, otherwise None.

7.14.2.79 _lookup_symbol()

```
Optional[ Symbol] l0_backend.Backend._lookup_symbol (
              self,
```

```
            str name,
            str current_module_name,
            Optional[List[str]]  module_path = None )  [protected]
```

Look up a symbol in the current module's environment.

This is used to determine which module a function is defined in so we can generate the correct mangled name.

Parameters

name	The name of the symbol.
current_module_name	Name of current module.
module_path	Optional module path for qualified names.

Returns

The resolved Symbol, or None if not found.

7.14.2.80 _materialize_arc_temp()

```
str l0_backend.Backend._materialize_arc_temp (
             self,
            str c_expr,
            Type expr_type )  [protected]
```

Materialize an ARC rvalue into a scope-owned temporary for automatic cleanup.

Parameters

c_expr	The C expression string.
expr_type	The type of the expression.

Returns

The name of the generated temporary variable.

7.14.2.81 _needs_arc_temp()

```
bool l0_backend.Backend._needs_arc_temp (
             self,
            Expr expr )  [protected]
```

Check if a non-place rvalue with ARC data needs temp materialization.

String literals are static constants and don't need cleanup.

Parameters

expr	The expression to check.

Returns

True if temp materialization is needed.

7.14.2.82 _pointer_type_or_none()

```
Optional[PointerType] l0_backend.Backend._pointer_type_or_none (
            self,
           Optional[Type] ty )  [protected]
```

Return the represented pointer type for pointer-shaped values.

7.14.2.83 _pop_scope()

```
None l0_backend.Backend._pop_scope (
            self )  [protected]
```

Exit current scope.

Exceptions

InternalCompilerError	If there is no current scope to pop.

7.14.2.84 _push_scope()

```
ScopeContext l0_backend.Backend._push_scope (
            self )  [protected]
```

Enter a new scope.

Returns

The newly created ScopeContext.

7.14.2.85 _register_inline_with_cleanup()

```
None l0_backend.Backend._register_inline_with_cleanup (
            self,
           ScopeContext scope,
           "WithItem" item )  [protected]
```

Register one inline with-item cleanup in LIFO order.

7.14.2.86 _resolve_let_type()

```
Type l0_backend.Backend._resolve_let_type (
            self,
           LetStmt stmt,
           str module_name )  [protected]
```

Resolve concrete type for a let declaration.

Parameters

stmt	The LetStmt AST node.
module_name	Name of current module.

Returns

The resolved Type.

Exceptions

InternalCompilerError	If type cannot be inferred.

7.14.2.87 _resolve_type_ref()

```
l0_backend.Backend._resolve_type_ref (
              self,
             TypeRef tref,
             str module_name )  [protected]
```

Resolve an AST TypeRef into an l0_types.Type.

This is needed so "let x: int? = null;" uses the declared type (int?) instead of the initializer type (null).

Parameters

tref	The TypeRef AST node.
module_name	Name of current module.

Returns

The resolved Type.

7.14.2.88 _scope_chain_has_cleanup()

```
bool l0_backend.Backend._scope_chain_has_cleanup (
              self )  [protected]
```

Check if any scope in the chain has cleanup requirements.

Returns

True if any scope has a with-cleanup or owned ARC variables.

7.14.2.89 _should_materialize_arc_temp()

```
bool l0_backend.Backend._should_materialize_arc_temp (
              self,
             Expr expr,
             Type expr_type )  [protected]
```

Check if an ARC expression should be hoisted to a cleanup temp.

Parameters

expr	The expression to check.
expr_type	The type of the expression.

Returns

True if the expression should be materialized into a temporary.

7.14.2.90 _sizeof_expr_for_type()

```
str l0_backend.Backend._sizeof_expr_for_type (
              self,
             Type ty )  [protected]
```

Return a C `sizeof` expression for the runtime access extent.

7.14.2.91 _topological_sort()

```
List[Tuple[str, str]] l0_backend.Backend._topological_sort (
              self,
             Dict[Tuple[str, str], Set[Tuple[str, str]]]  graph )  [protected]
```

Perform topological sort on type dependency graph using Kahn's algorithm.

Parameters

graph	The type dependency graph.

Returns

List of (module, type_name) in dependency order (dependencies first).

Exceptions

InternalCompilerError	On cycles (value-type cycles are impossible in valid L0).

7.14.2.92 _try_emit_constructor()

```
Optional[str] l0_backend.Backend._try_emit_constructor (
              self,
             CallExpr expr )  [protected]
```

Check if expr is a constructor call and emit appropriate initialization.

Struct: Point(1, 2) → { .x = 1, .y = 2 } Enum: Int(42) → { .tag = Expr_Int, .data = { .Int = { .value = 42 } } }

Parameters

expr	The call expression to check.

Returns

C code string if this is a constructor, otherwise None.

7.14.2.93 _try_emit_intrinsic()

```
Optional[str] l0_backend.Backend._try_emit_intrinsic (
              self,
             CallExpr expr )  [protected]
```

Expand compiler intrinsics inline.

Parameters

expr	The call expression to check.

Returns

C code string if it is an intrinsic, otherwise None.

7.14.2.94 _types_equal()

```
bool l0_backend.Backend._types_equal (
              self,
             Type a,
             Type b )  [protected]
```

Check if two types are structurally equal.

Parameters

a	First type.
b	Second type.

Returns

True if types are equal, False otherwise.

7.14.2.95 find_variant_decl()

```
Optional[EnumVariant] l0_backend.Backend.find_variant_decl (
              self,
```

```
            str module_name,
            str enum_name,
            str  variant_name )
```

Find the EnumVariant AST node for a given variant in an enum.

This is needed to get field names when binding pattern variables, since pattern variables are positional, but we need to access fields by name.

Parameters

module_name	Name of module containing the enum.
enum_name	Name of the enum.
variant_name	Name of the variant.

Returns

The EnumVariant AST node if found, otherwise None.

7.14.2.96 generate()

```
str l0_backend.Backend.generate (
              self )
```

Main entry point: generate complete C source for the compilation unit.

Returns

C source code as a string.

Exceptions

ValueError	If there is no compilation unit or if there are semantic errors.

7.14.2.97 ice()

```
NoReturn l0_backend.Backend.ice (
             self,
            str message,
            *Optional[Node]  node = None )
```

Raise an internal compiler error.

Parameters

message	The error message.
node	Optional AST node associated with the error.

Exceptions

InternalCompilerError	Always raised with the provided message and location.

7.14.3 Member Data Documentation

7.14.3.1 _current_func_result [1/2]

```
Optional l0_backend.Backend._current_func_result = None  [static], [protected]
```

7.14.3.2 _current_func_result [2/2]

```
l0_backend.Backend._current_func_result  [protected]
```

7.14.3.3 _current_scope [1/2]

```
Optional l0_backend.Backend._current_scope = None  [static], [protected]
```

7.14.3.4 _current_scope [2/2]

```
l0_backend.Backend._current_scope  [protected]
```

7.14.3.5 _emit_let_initializer

```
l0_backend.Backend._emit_let_initializer  [protected]
```

7.14.3.6 _label_counter

```
int l0_backend.Backend._label_counter = 0  [static], [protected]
```

7.14.3.7 _loop_cleanup_scope_stack

```
List l0_backend.Backend._loop_cleanup_scope_stack = field(default_factory=list)  [static],
[protected]
```

7.14.3.8 _loop_label_stack

```
List l0_backend.Backend._loop_label_stack = field(default_factory=list)  [static], [protected]
```

7.14.3.9 _next_stmt_unreachable [1/2]

```
bool l0_backend.Backend._next_stmt_unreachable = False  [static], [protected]
```

7.14.3.10 _next_stmt_unreachable [2/2]

```
l0_backend.Backend._next_stmt_unreachable  [protected]
```

7.14.3.11 _switch_depth

```
int l0_backend.Backend._switch_depth = 0  [static], [protected]
```

7.14.3.12 analysis [1/2]

```
AnalysisResult l0_backend.Backend.analysis  [static]
```

7.14.3.13 analysis [2/2]

```
l0_backend.Backend.analysis
```

7.14.3.14 current_module [1/2]

```
Optional l0_backend.Backend.current_module = None  [static]
```

7.14.3.15 current_module [2/2]

```
l0_backend.Backend.current_module
```

7.14.3.16 emitter

```
CEmitter l0_backend.Backend.emitter = field(default_factory=CEmitter)  [static]
```

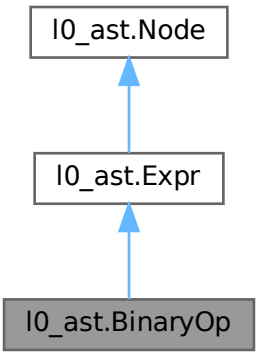
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_backend.py

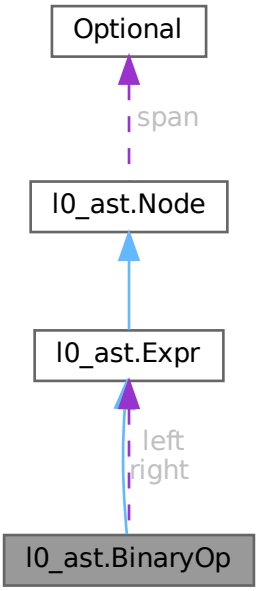
7.15 l0_ast.BinaryOp Class Reference

A binary operator expression.

Inheritance diagram for l0_ast.BinaryOp:



Collaboration diagram for l0_ast.BinaryOp:



Static Public Attributes

- str op
- Expr left
- Expr right

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.15.1 Detailed Description

A binary operator expression.

Attributes

- `op`: The operator string (e.g., "+", "&&").
- `left`: The left-hand side expression.
- `right`: The right-hand side expression.

7.15.2 Member Data Documentation

7.15.2.1 left

```
Expr l0_ast.BinaryOp.left  [static]
```

7.15.2.2 op

```
str l0_ast.BinaryOp.op  [static]
```

7.15.2.3 right

```
Expr l0_ast.BinaryOp.right  [static]
```

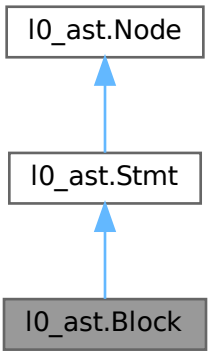
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

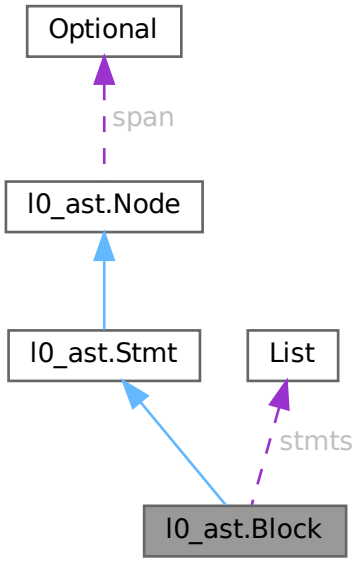
7.16 l0_ast.Block Class Reference

A block containing a sequence of statements.

Inheritance diagram for l0_ast.Block:



Collaboration diagram for l0_ast.Block:



Static Public Attributes

- List stmts [Stmt]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.16.1 Detailed Description

A block containing a sequence of statements.

Attributes

- `stmts`: List of statements in the block.

7.16.2 Member Data Documentation

7.16.2.1 stmts

```
List l0_ast.Block.stmts [Stmt]  [static]
```

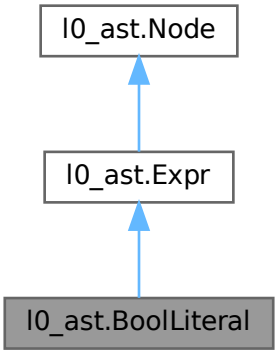
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

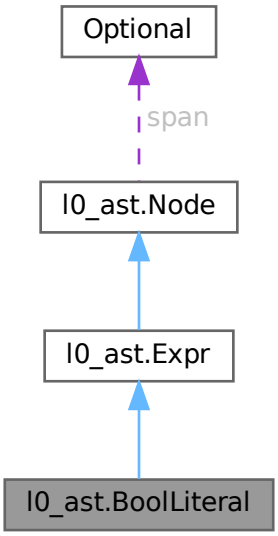
7.17 l0_ast.BoolLiteral Class Reference

A boolean literal expression.

Inheritance diagram for l0_ast.BoolLiteral:



Collaboration diagram for l0_ast.BoolLiteral:



Static Public Attributes

- bool value

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.17.1 Detailed Description

A boolean literal expression.

Attributes

- `value`: The boolean value.

7.17.2 Member Data Documentation

7.17.2.1 value

```
bool l0_ast.BoolLiteral.value  [static]
```

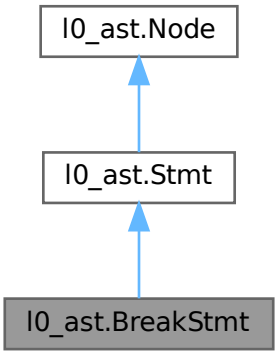
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

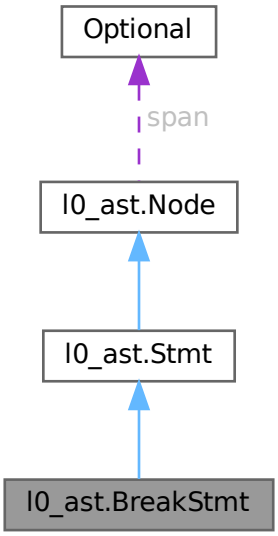
7.18 l0_ast.BreakStmt Class Reference

A break statement.

Inheritance diagram for l0_ast.BreakStmt:



Collaboration diagram for l0_ast.BreakStmt:



Additional Inherited Members

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.18.1 Detailed Description

A break statement.

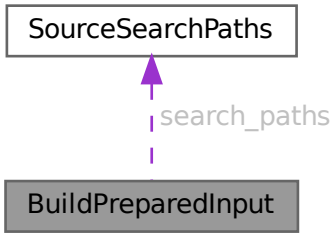
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.19 BuildPreparedInput Struct Reference

Prepared input bundle for build/run commands.

Collaboration diagram for BuildPreparedInput:



Public Attributes

- entry_module: string
- search_paths: SourceSearchPaths∗

7.19.1 Detailed Description

Prepared input bundle for build/run commands.

7.19.2 Member Data Documentation

7.19.2.1 entry_module

entry_module: `entry_module: string`

7.19.2.2 search_paths

search_paths: `search_paths: SourceSearchPaths*`

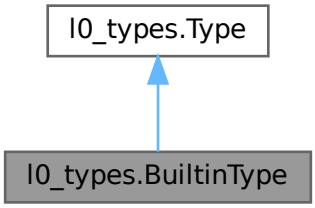
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/build_driver.l0

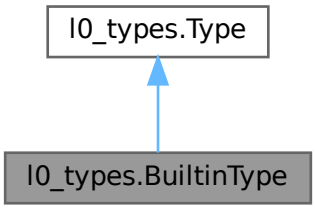
7.20 l0_types.BuiltinType Class Reference

A built-in primitive type.

Inheritance diagram for l0_types.BuiltinType:



Collaboration diagram for l0_types.BuiltinType:



Static Public Attributes

- str name

7.20.1 Detailed Description

A built-in primitive type.

Attributes

- `name`: The name of the type (e.g., "int", "bool").

7.20.2 Member Data Documentation

7.20.2.1 name

```
str l0_types.BuiltinType.name  [static]
```

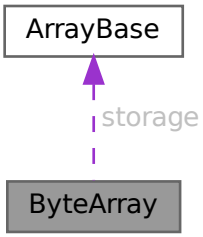
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_types.py

7.21 ByteArray Struct Reference

Byte-specialized fixed-size array wrapper.

Collaboration diagram for ByteArray:



Public Attributes

- storage: ArrayBase*

7.21.1 Detailed Description

Byte-specialized fixed-size array wrapper.

7.21.2 Member Data Documentation

7.21.2.1 storage

storage: `storage: ArrayBase*`

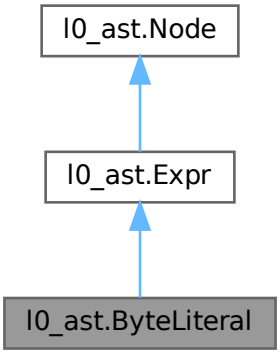
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/array.l0

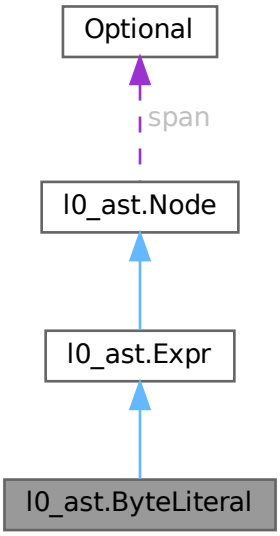
7.22 l0_ast.ByteLiteral Class Reference

A byte literal expression.

Inheritance diagram for l0_ast.ByteLiteral:



Collaboration diagram for l0_ast.ByteLiteral:



Static Public Attributes

- str value

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.22.1 Detailed Description

A byte literal expression.

Attributes

- `value`: The byte value as a string representation.

7.22.2 Member Data Documentation

7.22.2.1 value

```
str l0_ast.ByteLiteral.value  [static]
```

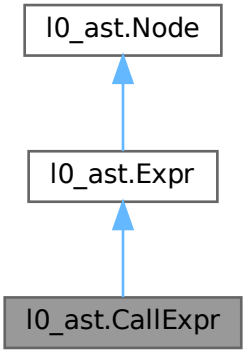
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

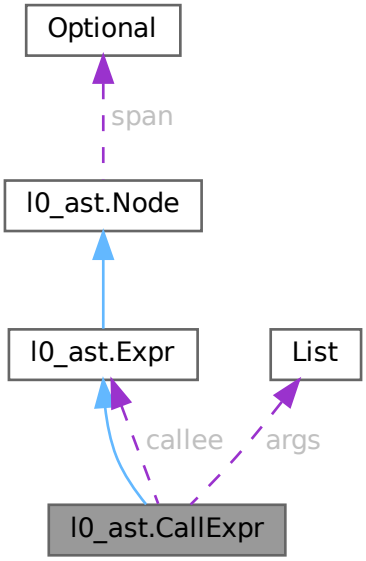
7.23 l0_ast.CallExpr Class Reference

A function or method call expression.

Inheritance diagram for l0_ast.CallExpr:



Collaboration diagram for l0_ast.CallExpr:



Static Public Attributes

- Expr callee
- List args [Expr]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.23.1 Detailed Description

A function or method call expression.

Attributes

- `callee`: The expression evaluating to the function to call.
- `args`: List of call arguments.

7.23.2 Member Data Documentation

7.23.2.1 args

```
List l0_ast.CallExpr.args [Expr]  [static]
```

7.23.2.2 callee

```
Expr l0_ast.CallExpr.callee  [static]
```

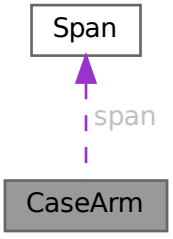
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.24 CaseArm Struct Reference

Represents the CaseArm structure.

Collaboration diagram for CaseArm:



Public Attributes

- literal: ExprId
- body: StmtId
- span: Span

7.24.1 Detailed Description

Represents the CaseArm structure.

7.24.2 Member Data Documentation

7.24.2.1 body

body: `body: StmtId`

7.24.2.2 literal

literal: `literal: ExprId`

7.24.2.3 span

span: `span: Span`

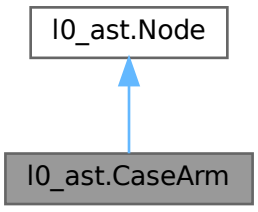
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

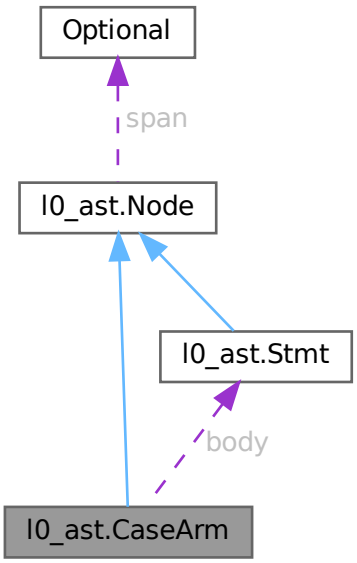
7.25 l0_ast.CaseArm Class Reference

A single arm in a case statement.

Inheritance diagram for l0_ast.CaseArm:



Collaboration diagram for l0_ast.CaseArm:



Static Public Attributes

- str literal : "Expr"
- Stmt body

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.25.1 Detailed Description

A single arm in a case statement.

Attributes

- `literal`: The constant expression to match.
- `body`: The statement to execute on match.

7.25.2 Member Data Documentation

7.25.2.1 body

```
Stmt l0_ast.CaseArm.body  [static]
```

7.25.2.2 literal

```
str l0_ast.CaseArm.literal :  "Expr"  [static]
```

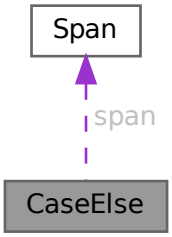
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.26 CaseElse Struct Reference

Represents the CaseElse structure.

Collaboration diagram for CaseElse:



Public Attributes

- body: StmtId
- span: Span

7.26.1 Detailed Description

Represents the CaseElse structure.

7.26.2 Member Data Documentation

7.26.2.1 body

body: `body:  StmtId`

7.26.2.2 span

span: `span: Span`

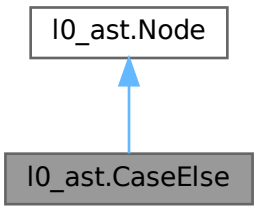
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

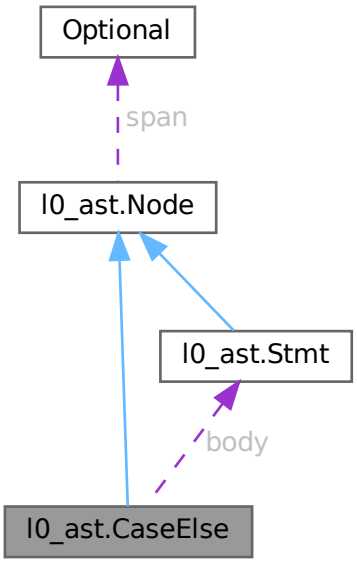
7.27 l0_ast.CaseElse Class Reference

The else arm of a case statement.

Inheritance diagram for l0_ast.CaseElse:



Collaboration diagram for l0_ast.CaseElse:



Static Public Attributes

- Stmt body

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.27.1 Detailed Description

The else arm of a case statement.

Attributes

- `body`: The statement to execute if no other arms match.

7.27.2 Member Data Documentation

7.27.2.1 body

```
Stmt l0_ast.CaseElse.body  [static]
```

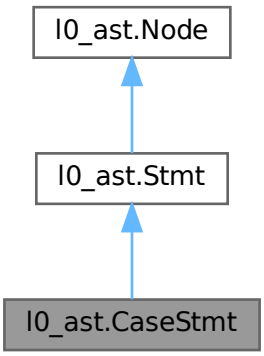
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

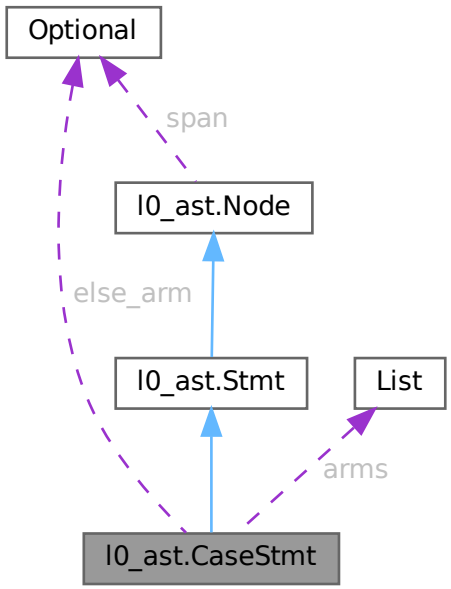
7.28 l0_ast.CaseStmt Class Reference

A case statement (switch-like).

Inheritance diagram for l0_ast.CaseStmt:



Collaboration diagram for l0_ast.CaseStmt:



Static Public Attributes

- str expr : "Expr"
- List arms [CaseArm]
- Optional else_arm [CaseElse]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.28.1 Detailed Description

A case statement (switch-like).

Attributes

- `expr`: The expression to evaluate.
- `arms`: List of constant arms.
- `else_arm`: Optional default arm.

7.28.2 Member Data Documentation

7.28.2.1 arms

```
List l0_ast.CaseStmt.arms [CaseArm]  [static]
```

7.28.2.2 else_arm

```
Optional l0_ast.CaseStmt.else_arm [CaseElse]  [static]
```

7.28.2.3 expr

```
str l0_ast.CaseStmt.expr :  "Expr"  [static]
```

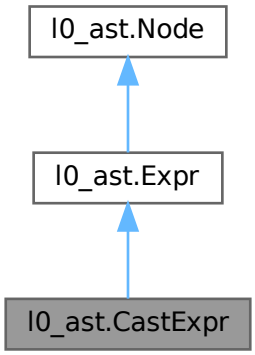
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

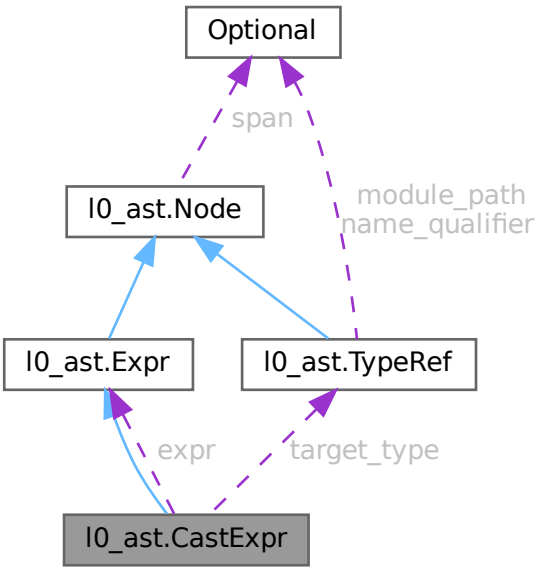
7.29 l0_ast.CastExpr Class Reference

A type cast expression.

Inheritance diagram for l0_ast.CastExpr:



Collaboration diagram for l0_ast.CastExpr:



Static Public Attributes

- Expr expr
- TypeRef target_type

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.29.1 Detailed Description

A type cast expression.

Attributes

- `expr`: The expression to cast.
- `target_type`: The type to cast to.

7.29.2 Member Data Documentation

7.29.2.1 expr

```
Expr l0_ast.CastExpr.expr  [static]
```

7.29.2.2 target_type

`TypeRef l0_ast.CastExpr.target_type [static]`

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.30 CCodeBuilder Struct Reference

Represents a line-oriented C source builder with indentation tracking.

Public Attributes

- lines: StringVector*
- indent_level: int
- indent_str: string
- cached_prefix: string
- cached_level: int

7.30.1 Detailed Description

Represents a line-oriented C source builder with indentation tracking.

7.30.2 Member Data Documentation

7.30.2.1 cached_level

cached_level: `cached_level: int`

7.30.2.2 cached_prefix

cached_prefix: `cached_prefix: string`

7.30.2.3 indent_level

indent_level: `indent_level: int`

7.30.2.4 indent_str

indent_str: `indent_str: string`

7.30.2.5 lines

lines: `lines: StringVector*`

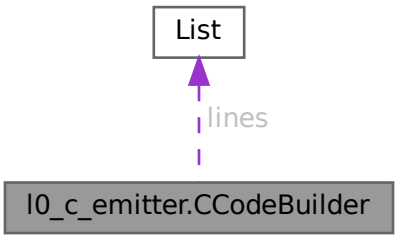
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/c_emitter.l0

7.31 l0_c_emitter.CCodeBuilder Class Reference

Helper for building C code with indentation tracking.

Collaboration diagram for l0_c_emitter.CCodeBuilder:



Public Member Functions

- None indent (self)
  Increase the indentation level.
- None dedent (self)
  Decrease the indentation level.
- None emit (self, str line="")
  Emit a line with current indentation.
- None emit_raw (self, str line)
  Emit a line without indentation.
- str to_string (self)
  Combine all lines into a single string.

Public Attributes

- lines

Static Public Attributes

- List lines = field(default_factory=list)
- int indent_level = 0
- str indent_str = " "

7.31.1 Detailed Description

Helper for building C code with indentation tracking.

Attributes

- `lines`: List of emitted code lines.
- `indent_level`: Current indentation depth.
- `indent_str`: String used for a single level of indentation. Defaults to 4 spaces.

7.31.2 Member Function Documentation

7.31.2.1 dedent()

```
None l0_c_emitter.CCodeBuilder.dedent (
              self )
```

Decrease the indentation level.

Exceptions

AssertionError	If indentation level is already zero.

7.31.2.2 emit()

```
None l0_c_emitter.CCodeBuilder.emit (
              self,
             str  line = "" )
```

Emit a line with current indentation.

Parameters

line	The C code line to emit. If empty, emits a blank line.

7.31.2.3 emit_raw()

```
None l0_c_emitter.CCodeBuilder.emit_raw (
              self,
             str line )
```

Emit a line without indentation.

Parameters

line	The C code line to emit directly.

7.31.2.4 indent()

```
None l0_c_emitter.CCodeBuilder.indent (
              self )
```

Increase the indentation level.

7.31.2.5 to_string()

```
str l0_c_emitter.CCodeBuilder.to_string (
              self )
```

Combine all lines into a single string.

Returns

The complete C source code string with a trailing newline.

7.31.3 Member Data Documentation

7.31.3.1 indent_level

```
int l0_c_emitter.CCodeBuilder.indent_level = 0  [static]
```

7.31.3.2 indent_str

```
str l0_c_emitter.CCodeBuilder.indent_str = "  "  [static]
```

7.31.3.3 lines [1/2]

```
List l0_c_emitter.CCodeBuilder.lines = field(default_factory=list)  [static]
```

7.31.3.4 lines [2/2]

```
l0_c_emitter.CCodeBuilder.lines
```

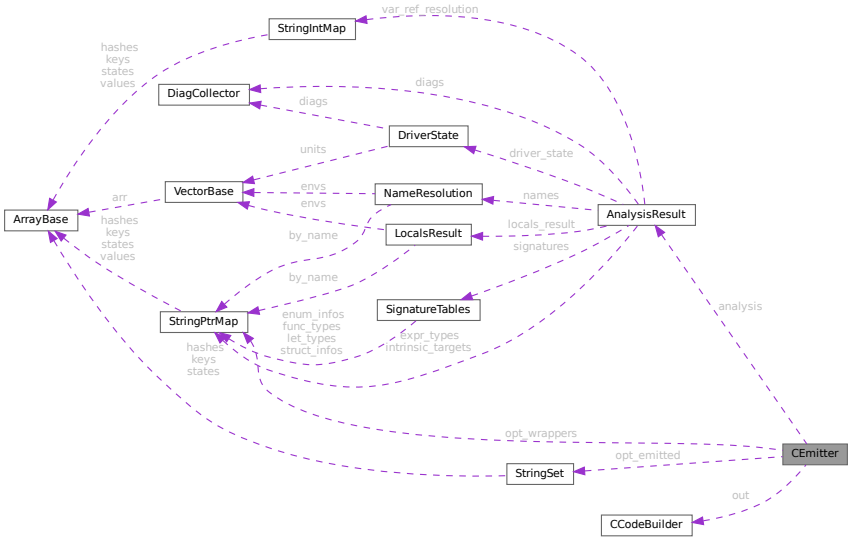
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_c_emitter.py

7.32 CEmitter Struct Reference

Represents the Stage 2 C emitter state.

Collaboration diagram for CEmitter:



Public Attributes

- analysis: AnalysisResult∗?
- current_module: string?
- out: CCodeBuilder∗
- tmp_counter: int
- active_line_directive: string
- opt_wrappers: StringPtrMap∗
- opt_emitted: StringSet∗

7.32.1 Detailed Description

Represents the Stage 2 C emitter state.

7.32.2 Member Data Documentation

7.32.2.1 active_line_directive

active_line_directive: `active_line_directive: string`

7.32.2.2 analysis

analysis: `analysis: AnalysisResult*?`

7.32.2.3 current_module

current_module: `current_module: string?`

7.32.2.4 opt_emitted

opt_emitted: `opt_emitted: StringSet*`

7.32.2.5 opt_wrappers

opt_wrappers: `opt_wrappers: StringPtrMap*`

7.32.2.6 out

out: `out: CCodeBuilder*`

7.32.2.7 tmp_counter

tmp_counter: `tmp_counter: int`

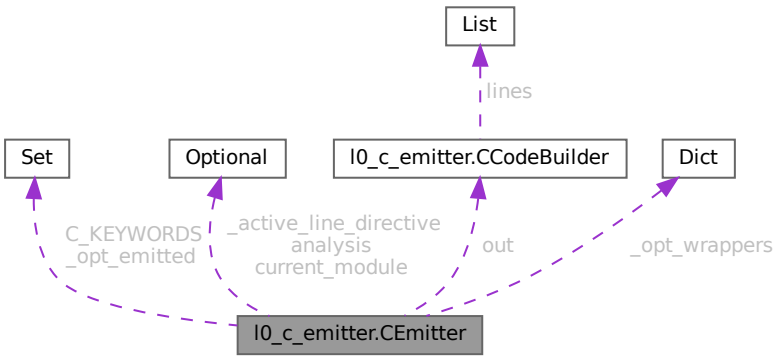
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/c_emitter.l0

7.33 l0_c_emitter.CEmitter Class Reference

C-specific code emitter.

Collaboration diagram for l0_c_emitter.CEmitter:



Public Member Functions

- None set_analysis (self, AnalysisResult analysis)

  Initialize emitter with analysis data.
- str get_output (self)

  Get the generated C code.
- Optional[EnumVariant] find_variant_decl (self, str module_name, str enum_name, str variant_name)

  Look up an enum variant's AST declaration.
- NoReturn ice (self, str message, Optional[object] node=None)

  Raise an internal compiler error with context.
- str mangle_struct_name (self, str module_name, str struct_name)

  Mangle a struct name to avoid C namespace collisions.
- str mangle_enum_name (self, str module_name, str enum_name)

  Mangle an enum name.
- str mangle_function_name (self, str module_name, str func_name)

  Mangle a function name.
- str mangle_let_name (self, str module_name, str let_name)

  Mangle a top-level let name to C identifier.
- str mangle_identifier (self, str name)

  Mangle an identifier if it conflicts with C keywords or L0 names.
- str fresh_tmp (self, str kind="tmp")

  Generate a unique temporary variable name.
- str emit_sizeof_type (self, Type typ)

  Emit C code for sizeof a given L0 type.
- str emit_ord (self, str c_enum_expr)

  Emit C code for ord(enum_value) intrinsic.
- str emit_type (self, Type typ)

  Convert an L0 Type to its C representation.
- bool is_niche_nullable (self, NullableType t)

  Public check for niche-optimized (pointer-shaped) nullable types.
- str emit_none_value_for_nullable (self, NullableType t)

  Emit C code for the L0 'null' value of a nullable type.
- str emit_some_value_for_nullable (self, NullableType t, str c_inner_expr)

  Emit C code for wrapping a value in 'some' for a nullable type.
- str emit_null_literal (self, Optional[Type] expected_type, *bool for_initializer=False)

  Emit a null literal appropriate for the expected type.
- str emit_widen_int (self, str c_expr, BuiltinType src_type, BuiltinType dst_type)

  Emit C code for implicit integer widening.
- str emit_pointer_type (self, Type base_type)

  Emit C pointer type for a base type.
- str emit_enum_tag (self, EnumType enum_type, str variant_name)

  Emit C tag enum value for an enum variant.
- None emit_section_comment (self, str text)

  Emit a decorative section comment.
- None emit_module_comment (self, str module_name)

  Emit a module metadata comment.
- None emit_module_separator (self, str module_name)

  Emit a decorative module separator.
- None emit_unreachable_comment (self)

  Emit an unreachable code comment for debugging.
- None emit_unreachable_marker (self, str reason="unreachable")

Emit a runtime panic call for unreachable code paths.

- None emit_header (self)

  Emit C header boilerplate, SipHash implementation, and L0 runtime.
- None emit_line_directive (self, Node node, str current_module)

  Emit #line directive for debugging generated C.
- None restore_active_line_directive (self)

  Re-emit the current source line directive after generated helper lines.
- None emit_forward_decls (self)

  Emit forward declarations for all structs and enums in compilation unit.
- None emit_struct (self, str module_name, StructDecl decl, StructInfo struct_info)

  Emit a complete C struct definition.
- None emit_enum (self, str module_name, EnumDecl decl, EnumInfo enum_info)

  Emit an L0 enum as a C tagged union.
- None emit_let_declaration (self, str module_name, LetDecl decl, Type let_type, Callable[[Expr, Type], str] let_initializer_callback)

  Emit a single top-level let declaration as a static variable.
- None emit_top_level_let_cleanup (self)

  Emit final cleanup for ARC-managed top-level lets before process exit.
- None emit_function_declaration (self, str module_name, FuncDecl decl, FuncType func_type)

  Emit a single function declaration signature.
- None emit_function_definition_header (self, str module_name, FuncDecl decl, FuncType func_type)

  Emit function definition header (signature and opening brace).
- None emit_function_definition_footer (self)

  Emit function definition footer (closing brace).
- None emit_main_wrapper (self, str entry_module, FuncType func_type)

  Emit C main() wrapper that calls the L0 entry function.
- None emit_optional_wrapper_for_defined_type (self, Type inner)

  Emit the collected optional wrapper for a just-defined user type.
- None prepare_optional_wrappers (self)

  Scan all compilation unit types and collect required optional wrappers.
- None emit_optional_wrappers (self, *bool early)

  Emit C typedef declarations for collected optional wrapper types.
- None emit_value_cleanup (self, str c_expr, Type ty)

  Emit C code to clean up an owned by-value variable.
- None emit_struct_cleanup (self, str c_ptr_expr, StructType struct_type)

  Emit cleanup code for all owned fields in a struct.
- None emit_enum_cleanup (self, str c_ptr_expr, EnumType enum_type)

  Emit cleanup code for owned fields in an enum's active variant.
- str emit_int_literal (self, int value)

  Emit C code for an integer literal.
- str emit_byte_literal (self, str value)

  Emit C code for a byte literal.
- str emit_string_literal (self, str value)

  Emit C code for an ARC string literal.
- str emit_const_string_literal (self, str value)

  Emit C code for a static string literal initializer.
- str emit_bool_literal (self, bool value)

  Emit C code for a boolean literal expression.
- str emit_const_bool_literal (self, bool value)

  Emit C code for a boolean literal in a static initializer.
- str emit_var_ref (self, str c_name)

Emit C code for a variable reference identifier.

- str emit_unary_op (self, str op, str c_operand)

  Emit C code for a unary operation.
- str emit_negated_condition (self, str c_cond)

  Emit a negated condition expression for control-flow lowering.
- str emit_binary_op (self, str op, str c_left, str c_right)

  Emit C code for a simple binary operation.
- str emit_condition_binary_op (self, str op, str c_left, str c_right)

  Emit a top-level binary condition for direct statement headers.
- str emit_checked_int_div (self, str c_left, str c_right)

  Emit C code for checked integer division runtime call.
- str emit_checked_int_mod (self, str c_left, str c_right)

  Emit C code for checked integer modulo runtime call.
- str emit_checked_int_mul (self, str c_left, str c_right)

  Emit C code for checked integer multiplication runtime call.
- str emit_checked_int_add (self, str c_left, str c_right)

  Emit C code for checked integer addition runtime call.
- str emit_checked_int_sub (self, str c_left, str c_right)

  Emit C code for checked integer subtraction runtime call.
- str emit_function_call (self, str c_func_name, str c_args)

  Emit C code for a function call.
- str emit_field_access (self, str c_obj, str field_name, bool is_pointer)

  Emit C code for field access using '.
- str emit_paren_expr (self, str c_inner)

  Emit C code for a parenthesized expression.
- str emit_cast (self, str c_type, str c_inner)

  Emit C code for a type cast.
- str emit_checked_ptr_access (self, str c_ptr_expr, str c_ptr_type, str c_required_size, str c_required_align, str access_mode="_RT_ACCESS_READ")

  Emit a runtime-checked pointer access expression.
- str emit_checked_ptr_index_access (self, str c_base_expr, str c_index_expr, str c_ptr_type, str c_element↩ _size, str c_required_align, str access_mode="_RT_ACCESS_READ")

  Emit a runtime-checked indexed pointer access expression.
- str emit_checked_ptr_access_for_base (self, str c_ptr_expr, str c_base_type, str access_mode="_RT_↩ ACCESS_READ")

  Emit a runtime-checked pointer to a complete object of `c_base_type`.
- str emit_drop_begin_expr (self, str c_ptr_expr, str c_ptr_type, str c_required_size, str c_required_align)

  Emit the expression that validates and begins a generated drop.
- None emit_drop_finish_call (self, str c_ptr_expr)

  Emit the runtime call that completes a generated drop.
- str emit_checked_narrow_cast (self, str c_dst_type, str c_inner)

  Emit C code for a checked narrowing cast runtime call.
- str emit_unwrap_ptr (self, str c_dst_type, str c_inner, str type_str)

  Emit C code for unwrapping a pointer-shaped optional runtime check.
- str emit_unwrap_opt (self, str c_src_type, str c_inner, str type_str)

  Emit C code for unwrapping a value-optional runtime check.
- str emit_null_check_eq (self, str c_expr)

  Emit C code for null equality check (opt == null).
- str emit_null_check_ne (self, str c_expr)

  Emit C code for null inequality check (opt != null).
- str emit_pointer_null_check (self, str c_expr, str op)

Emit C code for pointer null comparison.

- str emit_condition_pointer_null_check (self, str c_expr, str op)

  Emit a top-level pointer null comparison for direct statement headers.
- str emit_optional_has_value (self, str c_expr)

  Emit C code for reading an optional wrapper's has-value flag.
- str emit_optional_value (self, str c_expr)

  Emit C code for reading an optional wrapper's payload value.
- str emit_enum_tag_access (self, str c_expr)

  Emit C code for reading an enum tag.
- str emit_enum_payload_field_access (self, str c_expr, str variant, str field)

  Emit C code for reading a field from an enum payload.
- str emit_string_equals_call (self, str lhs, str rhs)

  Emit the runtime string-equality helper call.
- str emit_string_concat_call (self, str lhs, str rhs)

  Emit the runtime string-concatenation helper call.
- str emit_string_compare_call (self, str op, str lhs, str rhs)

  Emit the runtime string-compare helper call wrapped in a relational check.
- str emit_condition_string_compare_call (self, str op, str lhs, str rhs)

  Emit a top-level string relational check for direct statement headers.
- str emit_discard_expr (self, str c_expr)

  Emit a statement-context discard wrapper for an expression.
- str emit_deref_lvalue (self, str ptr_expr)

  Emit C code for a dereference lvalue: (∗ptr).
- str emit_field_lvalue (self, str obj, str field, bool is_pointer)

  Emit C code for a field access lvalue.
- str emit_index_lvalue (self, str base, str index)

  Emit C code for an index lvalue: base[idx].
- str emit_struct_constructor (self, str c_struct_name, List[Tuple[str, str]] field_inits)

  Emit C code for a struct compound literal constructor.
- str emit_struct_static_initializer (self, List[Tuple[str, str]] field_inits)

  Emit a brace-only struct initializer for static storage duration.
- str emit_struct_constructor_for_type (self, StructType struct_type, List[Tuple[str, str]] field_inits)

  Emit a C struct constructor for an L0 struct type.
- str emit_struct_static_initializer_for_type (self, StructType struct_type, List[Tuple[str, str]] field_inits)

  Emit a static-storage struct initializer for an L0 struct type.
- str emit_variant_constructor (self, str c_enum_name, str variant_name, str tag_value, List[Tuple[str, str]] payload_inits)

  Emit C code for an enum variant tagged union literal.
- str emit_variant_static_initializer (self, str variant_name, str tag_value, List[Tuple[str, str]] payload_inits)

  Emit a brace-only enum tagged-union initializer for static storage duration.
- str emit_variant_constructor_for_type (self, EnumType enum_type, str variant_name, List[Tuple[str, str]] payload_inits)

  Emit a tagged union constructor for a given L0 enum type.
- str emit_variant_static_initializer_for_type (self, EnumType enum_type, str variant_name, List[Tuple[str, str]] payload_inits)

  Emit a static-storage tagged union initializer for a given L0 enum type.
- str emit_pattern_binding_init (self, str scrutinee, str variant, str field)

  Emit C code for accessing a variant field during pattern matching.
- None emit_expr_stmt (self, str c_expr)

  Emit an expression as a statement.
- None emit_return_stmt (self, Optional[str] c_value)

Emit a C return statement.

- None emit_exit_switch (self)

  Emit a C break statement to exit a switch block.
- None emit_label (self, str label)

  Emit a C label followed by a null statement.
- None emit_goto (self, str label)

  Emit a C goto statement.
- None emit_block_start (self)

  Emit an opening brace and increase indentation.
- None emit_block_end (self)

  Emit a closing brace and decrease indentation.
- None emit_while_header (self, str c_cond)

  Emit a C while loop header.
- None emit_if_header (self, str c_cond)

  Emit a C if statement header.
- None emit_else (self)

  Emit a C else keyword.
- None emit_for_loop_start (self)

  Emit a decorative comment and opening brace for a for loop block.
- None emit_for_loop_end (self)

  Emit closing brace for a for loop block.
- None emit_let_decl (self, str c_type, str c_var_name, str c_init)

  Emit a C local variable declaration with initializer.
- None emit_assignment (self, str c_target, str c_value)

  Emit a simple C assignment statement.
- None emit_pointer_assignment (self, str c_ptr_name, str c_value)

  Emit a C assignment through a pointer.
- None emit_checked_pointer_assignment (self, str c_ptr_name, str c_base_type, str c_value)

  Emit an assignment through a runtime-checked object pointer.
- None emit_temp_decl (self, str c_type, str c_temp_name, str c_value)

  Emit a C temporary variable declaration with initializer.
- None emit_string_retain (self, str c_expr)

  Emit an ARC string retain runtime call.
- None emit_string_release (self, str c_expr)

  Emit an ARC string release runtime call.
- None emit_comment (self, str comment)

  Emit a C block comment.
- None emit_match_scrutinee_decl (self, str c_type, str c_expr)

  Emit the scrutinee declaration for a match/case statement.
- None emit_switch_start (self, str c_expr)

  Emit a C switch statement header and opening brace.
- None emit_match_switch_start (self, str scrutinee_name)

  Emit a match switch over an enum tag.
- None emit_switch_end (self)

  Emit a C switch statement closing brace.
- None emit_case_label (self, str c_tag_value)

  Emit a C case label.
- None emit_default_label (self)

  Emit a C default case label.
- None emit_null_assignment (self, str c_var)

  Emit a NULL assignment to a variable.

- None emit_alloc_obj (self, str c_ptr_type, str c_base_type, str c_temp_name)

  Emit a heap allocation runtime call.
- None emit_struct_init (self, str c_temp_name, str c_base_type, str c_init_str)

  Emit an object initialization through a pointer.
- None emit_struct_init_from_fields (self, str c_temp_name, Type base_type, List[Tuple[str, str]] field_inits)

  Emit struct initialization using positional field values.
- None emit_enum_variant_init (self, str c_temp_name, EnumType enum_type, str variant_name, List[Tuple[str, str]] payload_inits)

  Emit enum variant initialization for a heap-allocated enum.
- None emit_zero_init (self, str c_temp_name, str c_base_type)

  Emit zero-initialization through a pointer.
- None emit_try_check_niche (self, str c_tmp, str ret_none)

  Emit a NULL check for a niche-optimized optional.
- None emit_try_check_value (self, str c_tmp, str ret_none)

  Emit a has_value check for a value-optional.
- str emit_try_extract_value (self, str c_tmp)

  Emit C code to extract the inner value from an optional.

Public Attributes

- analysis

Static Public Attributes

- Set C_KEYWORDS
- Optional analysis = None
- Optional current_module = None
- CCodeBuilder out = field(default_factory=CCodeBuilder)

Protected Member Functions

- bool _is_niche_nullable (self, NullableType t)

  Check if a nullable type uses niche optimization (pointer-shaped).
- str _opt_key_for_type (self, Type t)

  Generate a unique key for an optional wrapper type name.
- str _opt_wrapper_name_for_inner (self, Type inner)

  Generate C typedef name for optional wrapper of given inner type.
- None _collect_opt_wrappers_from_type (self, Type t)

  Recursively collect optional wrapper types needed.
- bool _is_early_inner (self, Type inner)

  Check if an inner type wrapper can be emitted before user definitions.
- None _emit_optional_wrapper (self, str name, Type inner)

  Emit one collected optional wrapper typedef.
- None _emit_cleanup_by_type (self, str c_expr, Type ty)

  Emit recursive cleanup code for any ARC-managed data inside a type.
- None _emit_enum_value_cleanup (self, str c_expr, EnumType enum_type)

  Emit C cleanup code for an enum by-value variable.
- None _emit_enum_cleanup_switch (self, EnumType enum_type, str c_tag_expr, Callable[[str, str], str] field↩ _expr_for_variant_field, ∗bool missing_info_is_ice)

Emit a cleanup switch over the active variant of an enum value.

- List[Tuple[str, Type]] _iter_variant_cleanup_fields (self, EnumType enum_type, str variant_name, List[Type] variant_field_types)

  Pair variant field names with their resolved field types.
- bool _enum_has_arc_data (self, EnumInfo enum_info)

  Report whether any enum payload field requires ARC-aware cleanup.
- Optional[StructInfo] _get_struct_info (self, StructType struct_type, *bool strict)

  Look up resolved metadata for a struct type.
- Optional[EnumInfo] _get_enum_info (self, EnumType enum_type, *bool strict)

  Look up resolved metadata for an enum type.
- None _emit_field_cleanup (self, str field_expr, Type field_type)

  Emit recursive cleanup for a field.
- tuple[str, int] _string_token_to_c_bytes_and_len (self, str value)

  Decode an L0 string token payload and encode as C string-literal bytes.

Protected Attributes

- _active_line_directive

Static Protected Attributes

- int _tmp_counter = 0
- Optional _active_line_directive = None
- Dict _opt_wrappers = field(default_factory=dict)
- Set _opt_emitted = field(default_factory=set)

7.33.1 Detailed Description

C-specific code emitter.

Responsibilities:

- Emit C syntax (knows C keywords, syntax, conventions).
- Name mangling for C.
- Type emission (L0 types → C types).
- Statement/expression emission to C.
- Cleanup code emission (HOW to clean, not when/why).

Does NOT:

- Make decisions about what to emit.
- Perform semantic analysis.
- Manage scopes or lifetimes (queries backend for this).

Attributes

- `C_KEYWORDS`: Set of C keywords and restricted identifiers to be mangled.
- `analysis`: Full front-end analysis result.
- `current_module`: Name of the module currently being emitted.
- `out`: CCodeBuilder instance for output generation.

7.33.2 Member Function Documentation

7.33.2.1 _collect_opt_wrappers_from_type()

```
None l0_c_emitter.CEmitter._collect_opt_wrappers_from_type (
              self,
             Type t )  [protected]
```

Recursively collect optional wrapper types needed.

7.33.2.2 _emit_cleanup_by_type()

```
None l0_c_emitter.CEmitter._emit_cleanup_by_type (
              self,
             str c_expr,
             Type ty )  [protected]
```

Emit recursive cleanup code for any ARC-managed data inside a type.

Parameters

c_expr	C expression evaluating to the by-value object to clean up.
ty	The L0 type describing `c_expr`.

See also

`emit_value_cleanup`: Public entry point for by-value cleanup emission. `_emit_enum_value_↩cleanup`: Handles enum-specific cleanup lowering.

7.33.2.3 _emit_enum_cleanup_switch()

```
None l0_c_emitter.CEmitter._emit_enum_cleanup_switch (
              self,
             EnumType enum_type,
             str c_tag_expr,
             Callable[[str, str], str] field_expr_for_variant_field,
             *bool missing_info_is_ice )  [protected]
```

Emit a cleanup switch over the active variant of an enum value.

Parameters

enum_type	Enum type whose payload cleanup is being emitted.
c_tag_expr	C expression yielding the enum tag to switch on.
field_expr_for_variant_field	Callback that maps variant and field names to the corresponding C lvalue expression.
missing_info_is_ice	Whether missing enum metadata should raise an internal compiler error.

See also

`_emit_enum_value_cleanup`: Uses this helper for by-value enum cleanup. `_iter_variant_↩cleanup_fields`: Supplies the field names paired with types.

7.33.2.4 _emit_enum_value_cleanup()

```
None l0_c_emitter.CEmitter._emit_enum_value_cleanup (
             self,
            str c_expr,
            EnumType enum_type )  [protected]
```

Emit C cleanup code for an enum by-value variable.

7.33.2.5 _emit_field_cleanup()

```
None l0_c_emitter.CEmitter._emit_field_cleanup (
             self,
            str field_expr,
            Type field_type )  [protected]
```

Emit recursive cleanup for a field.

7.33.2.6 _emit_optional_wrapper()

```
None l0_c_emitter.CEmitter._emit_optional_wrapper (
             self,
            str name,
            Type inner )  [protected]
```

Emit one collected optional wrapper typedef.

Parameters

name	Wrapper typedef name.
inner	Inner wrapped type.

7.33.2.7 _enum_has_arc_data()

```
bool l0_c_emitter.CEmitter._enum_has_arc_data (
             self,
            EnumInfo enum_info )  [protected]
```

Report whether any enum payload field requires ARC-aware cleanup.

Parameters

enum_info	Resolved enum metadata to inspect.

Returns

`True` if any variant payload contains ARC-managed data.

See also

`_emit_enum_cleanup_switch`: Skips switch emission when this is false.

7.33.2.8 _get_enum_info()

```
Optional[EnumInfo] l0_c_emitter.CEmitter._get_enum_info (
              self,
             EnumType enum_type,
             *bool strict )  [protected]
```

Look up resolved metadata for an enum type.

Parameters

enum_type	Enum type whose metadata is needed.
strict	Whether a missing entry should raise an internal compiler error.

Returns

The matching `EnumInfo` when available, otherwise `None`.

See also

`_emit_enum_cleanup_switch`: Uses enum metadata to enumerate payload fields.

7.33.2.9 _get_struct_info()

```
Optional[StructInfo] l0_c_emitter.CEmitter._get_struct_info (
              self,
             StructType struct_type,
             *bool strict )  [protected]
```

Look up resolved metadata for a struct type.

Parameters

struct_type	Struct type whose metadata is needed.
strict	Whether a missing entry should raise an internal compiler error.

Returns

The matching `StructInfo` when available, otherwise `None`.

See also

`emit_struct_cleanup`: Requires strict struct metadata during emission.

7.33.2.10 _is_early_inner()

```
bool l0_c_emitter.CEmitter._is_early_inner (
             self,
            Type inner )  [protected]
```

Check if an inner type wrapper can be emitted before user definitions.

7.33.2.11 _is_niche_nullable()

```
bool l0_c_emitter.CEmitter._is_niche_nullable (
             self,
            NullableType t )  [protected]
```

Check if a nullable type uses niche optimization (pointer-shaped).

7.33.2.12 _iter_variant_cleanup_fields()

```
List[Tuple[str, Type]] l0_c_emitter.CEmitter._iter_variant_cleanup_fields (
             self,
            EnumType enum_type,
            str variant_name,
            List[Type] variant_field_types )  [protected]
```

Pair variant field names with their resolved field types.

Parameters

enum_type	Enum type owning the variant.
variant_name	Variant whose fields are being cleaned up.
variant_field_types	Resolved payload field types for the variant.

Returns

A list of "(field_name, field_type)" pairs, or an empty list when the AST declaration cannot be matched to the resolved payload types.

See also

`find_variant_decl`: Retrieves the source declaration used for field names. `_emit_enum_↩cleanup_switch`: Consumes these pairs to emit cleanup code.

7.33.2.13 _opt_key_for_type()

```
str l0_c_emitter.CEmitter._opt_key_for_type (
              self,
             Type t )  [protected]
```

Generate a unique key for an optional wrapper type name.

7.33.2.14 _opt_wrapper_name_for_inner()

```
str l0_c_emitter.CEmitter._opt_wrapper_name_for_inner (
              self,
             Type inner )  [protected]
```

Generate C typedef name for optional wrapper of given inner type.

7.33.2.15 _string_token_to_c_bytes_and_len()

```
tuple[str, int] l0_c_emitter.CEmitter._string_token_to_c_bytes_and_len (
              self,
             str value )  [protected]
```

Decode an L0 string token payload and encode as C string-literal bytes.

7.33.2.16 emit_alloc_obj()

```
None l0_c_emitter.CEmitter.emit_alloc_obj (
              self,
             str c_ptr_type,
             str c_base_type,
             str c_temp_name )
```

Emit a heap allocation runtime call.

Parameters

c_ptr_type	C pointer type string.
c_base_type	C base object type string.
c_temp_name	Name of temporary to hold the pointer.

7.33.2.17 emit_assignment()

```
None l0_c_emitter.CEmitter.emit_assignment (
              self,
             str c_target,
             str c_value )
```

Emit a simple C assignment statement.

Parameters

c_target	C lvalue expression.
c_value	C expression for the value.

7.33.2.18 emit_binary_op()

```
str l0_c_emitter.CEmitter.emit_binary_op (
              self,
             str op,
             str c_left,
             str c_right )
```

Emit C code for a simple binary operation.

Parameters

op	C binary operator string.
c_left	C expression for left operand.
c_right	C expression for right operand.

Returns

C binary expression string.

7.33.2.19 emit_block_end()

```
None l0_c_emitter.CEmitter.emit_block_end (
              self )
```

Emit a closing brace and decrease indentation.

7.33.2.20 emit_block_start()

```
None l0_c_emitter.CEmitter.emit_block_start (
              self )
```

Emit an opening brace and increase indentation.

7.33.2.21 emit_bool_literal()

```
str l0_c_emitter.CEmitter.emit_bool_literal (
              self,
             bool value )
```

Emit C code for a boolean literal expression.

Parameters

value	The boolean value.

Returns

"1" for true, "0" for false.

7.33.2.22 emit_byte_literal()

```
str l0_c_emitter.CEmitter.emit_byte_literal (
              self,
             str value )
```

Emit C code for a byte literal.

Parameters

value	The byte literal character.

Returns

C casted character literal.

7.33.2.23 emit_case_label()

```
None l0_c_emitter.CEmitter.emit_case_label (
              self,
             str c_tag_value )
```

Emit a C case label.

Parameters

c_tag_value	The constant C tag identifier or literal.

7.33.2.24 emit_cast()

```
str l0_c_emitter.CEmitter.emit_cast (
              self,
             str c_type,
             str c_inner )
```

Emit C code for a type cast.

Parameters

c_type	Target C type string.
c_inner	Expression to cast.

Returns

C cast expression string.

7.33.2.25 emit_checked_int_add()

```
str l0_c_emitter.CEmitter.emit_checked_int_add (
              self,
             str c_left,
             str c_right )
```

Emit C code for checked integer addition runtime call.

7.33.2.26 emit_checked_int_div()

```
str l0_c_emitter.CEmitter.emit_checked_int_div (
              self,
             str c_left,
             str c_right )
```

Emit C code for checked integer division runtime call.

7.33.2.27 emit_checked_int_mod()

```
str l0_c_emitter.CEmitter.emit_checked_int_mod (
              self,
             str c_left,
             str c_right )
```

Emit C code for checked integer modulo runtime call.

7.33.2.28 emit_checked_int_mul()

```
str l0_c_emitter.CEmitter.emit_checked_int_mul (
              self,
             str c_left,
             str c_right )
```

Emit C code for checked integer multiplication runtime call.

7.33.2.29 emit_checked_int_sub()

```
str l0_c_emitter.CEmitter.emit_checked_int_sub (
              self,
             str c_left,
             str c_right )
```

Emit C code for checked integer subtraction runtime call.

7.33.2.30 emit_checked_narrow_cast()

```
str l0_c_emitter.CEmitter.emit_checked_narrow_cast (
              self,
             str c_dst_type,
             str c_inner )
```

Emit C code for a checked narrowing cast runtime call.

7.33.2.31 emit_checked_pointer_assignment()

```
None l0_c_emitter.CEmitter.emit_checked_pointer_assignment (
              self,
             str c_ptr_name,
             str c_base_type,
             str c_value )
```

Emit an assignment through a runtime-checked object pointer.

7.33.2.32 emit_checked_ptr_access()

```
str l0_c_emitter.CEmitter.emit_checked_ptr_access (
              self,
             str c_ptr_expr,
             str c_ptr_type,
             str c_required_size,
             str c_required_align,
             str  access_mode = "_RT_ACCESS_READ" )
```

Emit a runtime-checked pointer access expression.

Declares one static per-call-site cache slot so the runtime validates repeated accesses from the same site with a single range check.

7.33.2.33 emit_checked_ptr_access_for_base()

```
str l0_c_emitter.CEmitter.emit_checked_ptr_access_for_base (
              self,
             str c_ptr_expr,
             str c_base_type,
             str  access_mode = "_RT_ACCESS_READ" )
```

Emit a runtime-checked pointer to a complete object of `c_base_type`.

7.33.2.34 emit_checked_ptr_index_access()

```
str l0_c_emitter.CEmitter.emit_checked_ptr_index_access (
              self,
             str c_base_expr,
             str c_index_expr,
             str c_ptr_type,
             str c_element_size,
             str c_required_align,
             str  access_mode = "_RT_ACCESS_READ" )
```

Emit a runtime-checked indexed pointer access expression.

7.33.2.35 emit_comment()

```
None l0_c_emitter.CEmitter.emit_comment (
              self,
             str comment )
```

Emit a C block comment.

Parameters

comment	The comment text.

7.33.2.36 emit_condition_binary_op()

```
str l0_c_emitter.CEmitter.emit_condition_binary_op (
              self,
             str op,
             str c_left,
             str c_right )
```

Emit a top-level binary condition for direct statement headers.

7.33.2.37 emit_condition_pointer_null_check()

```
str l0_c_emitter.CEmitter.emit_condition_pointer_null_check (
              self,
             str c_expr,
             str op )
```

Emit a top-level pointer null comparison for direct statement headers.

7.33.2.38 emit_condition_string_compare_call()

```
str l0_c_emitter.CEmitter.emit_condition_string_compare_call (
              self,
             str op,
             str lhs,
             str rhs )
```

Emit a top-level string relational check for direct statement headers.

7.33.2.39 emit_const_bool_literal()

```
str l0_c_emitter.CEmitter.emit_const_bool_literal (
              self,
             bool value )
```

Emit C code for a boolean literal in a static initializer.

Parameters

value	The boolean value.

Returns

"true" or "false".

7.33.2.40 emit_const_string_literal()

```
str l0_c_emitter.CEmitter.emit_const_string_literal (
              self,
             str value )
```

Emit C code for a static string literal initializer.

Parameters

value	The L0 string token payload.

Returns

C L0_STRING_CONST macro expression.

7.33.2.41 emit_default_label()

```
None l0_c_emitter.CEmitter.emit_default_label (
              self )
```

Emit a C default case label.

7.33.2.42 emit_deref_lvalue()

```
str l0_c_emitter.CEmitter.emit_deref_lvalue (
              self,
             str ptr_expr )
```

Emit C code for a dereference lvalue: (∗ptr).

7.33.2.43 emit_discard_expr()

```
str l0_c_emitter.CEmitter.emit_discard_expr (
              self,
             str c_expr )
```

Emit a statement-context discard wrapper for an expression.

7.33.2.44 emit_drop_begin_expr()

```
str l0_c_emitter.CEmitter.emit_drop_begin_expr (
              self,
             str c_ptr_expr,
             str c_ptr_type,
             str c_required_size,
             str c_required_align )
```

Emit the expression that validates and begins a generated drop.

7.33.2.45 emit_drop_finish_call()

```
None l0_c_emitter.CEmitter.emit_drop_finish_call (
              self,
             str c_ptr_expr )
```

Emit the runtime call that completes a generated drop.

7.33.2.46 emit_else()

```
None l0_c_emitter.CEmitter.emit_else (
              self )
```

Emit a C else keyword.

7.33.2.47 emit_enum()

```
None l0_c_emitter.CEmitter.emit_enum (
              self,
             str module_name,
             EnumDecl decl,
             EnumInfo enum_info )
```

Emit an L0 enum as a C tagged union.

Parameters

module_name	Name of the module.
decl	EnumDecl AST node.
enum_info	Resolved type information for the enum.

Exceptions

InternalCompilerError	If variant information is missing.

7.33.2.48 emit_enum_cleanup()

```
None l0_c_emitter.CEmitter.emit_enum_cleanup (
              self,
             str c_ptr_expr,
             EnumType enum_type )
```

Emit cleanup code for owned fields in an enum's active variant.

Parameters

c_ptr_expr	C expression evaluating to a pointer to the enum.
enum_type	The L0 enum type.

7.33.2.49 emit_enum_payload_field_access()

```
str l0_c_emitter.CEmitter.emit_enum_payload_field_access (
              self,
             str c_expr,
             str variant,
             str field )
```

Emit C code for reading a field from an enum payload.

7.33.2.50 emit_enum_tag()

```
str l0_c_emitter.CEmitter.emit_enum_tag (
              self,
             EnumType enum_type,
             str variant_name )
```

Emit C tag enum value for an enum variant.

Parameters

enum_type	The L0 enum type.
variant_name	The name of the variant.

Returns

The mangled C enum tag identifier.

7.33.2.51 emit_enum_tag_access()

```
str l0_c_emitter.CEmitter.emit_enum_tag_access (
              self,
             str c_expr )
```

Emit C code for reading an enum tag.

7.33.2.52 emit_enum_variant_init()

```
None l0_c_emitter.CEmitter.emit_enum_variant_init (
              self,
             str c_temp_name,
             EnumType enum_type,
             str variant_name,
             List[Tuple[str, str]]  payload_inits )
```

Emit enum variant initialization for a heap-allocated enum.

Parameters

c_temp_name	Name of the pointer variable.
enum_type	L0 enum type.
variant_name	Name of the active variant.
payload_inits	List of (field_name, c_value) tuples for payload.

7.33.2.53 emit_exit_switch()

```
None l0_c_emitter.CEmitter.emit_exit_switch (
              self )
```

Emit a C break statement to exit a switch block.

7.33.2.54 emit_expr_stmt()

```
None l0_c_emitter.CEmitter.emit_expr_stmt (
              self,
             str c_expr )
```

Emit an expression as a statement.

Parameters

c_expr	C expression string.

7.33.2.55 emit_field_access()

```
str l0_c_emitter.CEmitter.emit_field_access (
```

```
             self,
            str c_obj,
            str field_name,
            bool is_pointer )
```

Emit C code for field access using '.

' or '→'.

Parameters

c_obj	C expression for the object.
field_name	The field identifier.
is_pointer	True if the object is a pointer.

Returns

C field access expression string.

7.33.2.56 emit_field_lvalue()

```
str l0_c_emitter.CEmitter.emit_field_lvalue (
             self,
            str obj,
            str field,
            bool is_pointer )
```

Emit C code for a field access lvalue.

7.33.2.57 emit_for_loop_end()

```
None l0_c_emitter.CEmitter.emit_for_loop_end (
             self )
```

Emit closing brace for a for loop block.

7.33.2.58 emit_for_loop_start()

```
None l0_c_emitter.CEmitter.emit_for_loop_start (
             self )
```

Emit a decorative comment and opening brace for a for loop block.

7.33.2.59 emit_forward_decls()

```
None l0_c_emitter.CEmitter.emit_forward_decls (
             self )
```

Emit forward declarations for all structs and enums in compilation unit.

7.33.2.60 emit_function_call()

```
str l0_c_emitter.CEmitter.emit_function_call (
             self,
            str c_func_name,
            str c_args )
```

Emit C code for a function call.

Parameters

c_func_name	C identifier or expression for the function.
c_args	Comma-separated C expression string for arguments.

Returns

C function call expression string.

7.33.2.61 emit_function_declaration()

```
None l0_c_emitter.CEmitter.emit_function_declaration (
              self,
             str module_name,
             FuncDecl decl,
             FuncType func_type )
```

Emit a single function declaration signature.

Parameters

module_name	Name of the module.
decl	FuncDecl AST node.
func_type	Resolved signature of the function.

7.33.2.62 emit_function_definition_footer()

```
None l0_c_emitter.CEmitter.emit_function_definition_footer (
              self )
```

Emit function definition footer (closing brace).

7.33.2.63 emit_function_definition_header()

```
None l0_c_emitter.CEmitter.emit_function_definition_header (
              self,
             str module_name,
             FuncDecl decl,
             FuncType func_type )
```

Emit function definition header (signature and opening brace).

Parameters

module_name	Name of the module.
decl	FuncDecl AST node.
func_type	Resolved signature of the function.

7.33.2.64 emit_goto()

```
None l0_c_emitter.CEmitter.emit_goto (
             self,
            str label )
```

Emit a C goto statement.

Parameters

label	Target C label identifier.

7.33.2.65 emit_header()

```
None l0_c_emitter.CEmitter.emit_header (
             self )
```

Emit C header boilerplate, SipHash implementation, and L0 runtime.

7.33.2.66 emit_if_header()

```
None l0_c_emitter.CEmitter.emit_if_header (
             self,
            str c_cond )
```

Emit a C if statement header.

Parameters

c_cond	C expression for the condition.

7.33.2.67 emit_index_lvalue()

```
str l0_c_emitter.CEmitter.emit_index_lvalue (
             self,
            str base,
            str index )
```

Emit C code for an index lvalue: base[idx].

7.33.2.68 emit_int_literal()

```
str l0_c_emitter.CEmitter.emit_int_literal (
             self,
            int value )
```

Emit C code for an integer literal.

Parameters

value	The integer value.

Returns

C literal string (handles INT32_MIN edge case).

7.33.2.69 emit_label()

```
None l0_c_emitter.CEmitter.emit_label (
              self,
             str label )
```

Emit a C label followed by a null statement.

Parameters

label	C label identifier.

7.33.2.70 emit_let_decl()

```
None l0_c_emitter.CEmitter.emit_let_decl (
              self,
             str c_type,
             str c_var_name,
             str c_init )
```

Emit a C local variable declaration with initializer.

Parameters

c_type	C type string.
c_var_name	C identifier.
c_init	C initializer expression string.

7.33.2.71 emit_let_declaration()

```
None l0_c_emitter.CEmitter.emit_let_declaration (
              self,
             str module_name,
             LetDecl decl,
             Type let_type,
             Callable[[Expr, Type], str] let_initializer_callback )
```

Emit a single top-level let declaration as a static variable.

Parameters

module_name	Module containing the let.
decl	Let declaration AST node.
let_type	Resolved type of the let.
let_initializer_callback	Callback to emit initializer expression string.

7.33.2.72 emit_line_directive()

```
None l0_c_emitter.CEmitter.emit_line_directive (
              self,
             Node node,
             str current_module )
```

Emit #line directive for debugging generated C.

Parameters

node	AST node with span information.
current_module	Name of the current module.

7.33.2.73 emit_main_wrapper()

```
None l0_c_emitter.CEmitter.emit_main_wrapper (
              self,
             str entry_module,
             FuncType func_type )
```

Emit C main() wrapper that calls the L0 entry function.

Parameters

entry_module	Name of the entry module.
func_type	Resolved signature of the L0 main function.

7.33.2.74 emit_match_scrutinee_decl()

```
None l0_c_emitter.CEmitter.emit_match_scrutinee_decl (
              self,
             str c_type,
             str c_expr )
```

Emit the scrutinee declaration for a match/case statement.

Parameters

c_type	C type string of scrutinee.
c_expr	C expression for scrutinee value.

7.33.2.75 emit_match_switch_start()

```
None l0_c_emitter.CEmitter.emit_match_switch_start (
              self,
             str scrutinee_name )
```

Emit a match switch over an enum tag.

Parameters

scrutinee_name	Identifier of the scrutinee variable.

7.33.2.76 emit_module_comment()

```
None l0_c_emitter.CEmitter.emit_module_comment (
              self,
             str module_name )
```

Emit a module metadata comment.

Parameters

module_name	The name of the module.

7.33.2.77 emit_module_separator()

```
None l0_c_emitter.CEmitter.emit_module_separator (
              self,
             str module_name )
```

Emit a decorative module separator.

Parameters

module_name	The name of the module.

7.33.2.78 emit_negated_condition()

```
str l0_c_emitter.CEmitter.emit_negated_condition (
              self,
             str c_cond )
```

Emit a negated condition expression for control-flow lowering.

7.33.2.79 emit_none_value_for_nullable()

```
str l0_c_emitter.CEmitter.emit_none_value_for_nullable (
              self,
             NullableType t )
```

Emit C code for the L0 'null' value of a nullable type.

Parameters

t	The NullableType.

Returns

C code string representing the 'none' state.

7.33.2.80 emit_null_assignment()

```
None l0_c_emitter.CEmitter.emit_null_assignment (
              self,
             str c_var )
```

Emit a NULL assignment to a variable.

Parameters

c_var	C lvalue expression.

7.33.2.81 emit_null_check_eq()

```
str l0_c_emitter.CEmitter.emit_null_check_eq (
              self,
             str c_expr )
```

Emit C code for null equality check (opt == null).

7.33.2.82 emit_null_check_ne()

```
str l0_c_emitter.CEmitter.emit_null_check_ne (
              self,
             str c_expr )
```

Emit C code for null inequality check (opt != null).

7.33.2.83 emit_null_literal()

```
str l0_c_emitter.CEmitter.emit_null_literal (
              self,
             Optional[Type] expected_type,
             *bool  for_initializer = False )
```

Emit a null literal appropriate for the expected type.

Parameters

expected_type	The type expected in this context.
for_initializer	If True, uses C initializer syntax ({0}).

Returns

C code for the null value.

Exceptions

InternalCompilerError	If expected type is not nullable or a pointer.

7.33.2.84 emit_optional_has_value()

```
str l0_c_emitter.CEmitter.emit_optional_has_value (
              self,
             str c_expr )
```

Emit C code for reading an optional wrapper's has-value flag.

7.33.2.85 emit_optional_value()

```
str l0_c_emitter.CEmitter.emit_optional_value (
              self,
             str c_expr )
```

Emit C code for reading an optional wrapper's payload value.

7.33.2.86 emit_optional_wrapper_for_defined_type()

```
None l0_c_emitter.CEmitter.emit_optional_wrapper_for_defined_type (
              self,
             Type inner )
```

Emit the collected optional wrapper for a just-defined user type.

Parameters

inner	Just-defined struct or enum type.

7.33.2.87 emit_optional_wrappers()

```
None l0_c_emitter.CEmitter.emit_optional_wrappers (
              self,
             *bool early )
```

Emit C typedef declarations for collected optional wrapper types.

Parameters

early	If True, emit wrappers for builtins only. If False, emit wrappers for user-defined structs/enums.

7.33.2.88 emit_ord()

```
str l0_c_emitter.CEmitter.emit_ord (
              self,
             str c_enum_expr )
```

Emit C code for ord(enum_value) intrinsic.

Parameters

c_enum_expr	C expression evaluating to an enum value.

Returns

A C expression that extracts the tag field and casts to l0_int.

7.33.2.89 emit_paren_expr()

```
str l0_c_emitter.CEmitter.emit_paren_expr (
              self,
             str c_inner )
```

Emit C code for a parenthesized expression.

Parameters

c_inner	The inner C expression string.

Returns

Parenthesized expression string.

7.33.2.90 emit_pattern_binding_init()

```
str l0_c_emitter.CEmitter.emit_pattern_binding_init (
              self,
             str scrutinee,
             str variant,
             str field )
```

Emit C code for accessing a variant field during pattern matching.

Parameters

scrutinee	Name of the scrutinee variable.
variant	Name of the variant being matched.
field	Name of the payload field being extracted.

Returns

C field access expression string.

7.33.2.91 emit_pointer_assignment()

```
None l0_c_emitter.CEmitter.emit_pointer_assignment (
              self,
             str c_ptr_name,
             str c_value )
```

Emit a C assignment through a pointer.

Parameters

c_ptr_name	C expression evaluating to a pointer.
c_value	C expression for the value.

7.33.2.92 emit_pointer_null_check()

```
str l0_c_emitter.CEmitter.emit_pointer_null_check (
              self,
             str c_expr,
             str op )
```

Emit C code for pointer null comparison.

Parameters

c_expr	C expression for the pointer.
op	Comparison operator (e.g., "==", "!=").

Returns

C comparison expression string.

7.33.2.93 emit_pointer_type()

```
str l0_c_emitter.CEmitter.emit_pointer_type (
              self,
             Type base_type )
```

Emit C pointer type for a base type.

Parameters

base_type	The L0 type to point to.

Returns

C type string (e.g., "l0_int*").

7.33.2.94 emit_return_stmt()

```
None l0_c_emitter.CEmitter.emit_return_stmt (
             self,
            Optional[str] c_value )
```

Emit a C return statement.

Parameters

c_value	Optional C expression string to return.

7.33.2.95 emit_section_comment()

```
None l0_c_emitter.CEmitter.emit_section_comment (
             self,
            str text )
```

Emit a decorative section comment.

Parameters

text	The comment text.

7.33.2.96 emit_sizeof_type()

```
str l0_c_emitter.CEmitter.emit_sizeof_type (
             self,
            Type typ )
```

Emit C code for sizeof a given L0 type.

Parameters

typ	The type to measure.

Returns

A C expression like "((l0_int)sizeof(l0_int))".

7.33.2.97 emit_some_value_for_nullable()

```
str l0_c_emitter.CEmitter.emit_some_value_for_nullable (
              self,
             NullableType t,
             str c_inner_expr )
```

Emit C code for wrapping a value in 'some' for a nullable type.

Parameters

t	The NullableType.
c_inner_expr	C expression for the inner value.

Returns

C code string representing the 'some' state.

7.33.2.98 emit_string_compare_call()

```
str l0_c_emitter.CEmitter.emit_string_compare_call (
              self,
             str op,
             str lhs,
             str rhs )
```

Emit the runtime string-compare helper call wrapped in a relational check.

7.33.2.99 emit_string_concat_call()

```
str l0_c_emitter.CEmitter.emit_string_concat_call (
              self,
             str lhs,
             str rhs )
```

Emit the runtime string-concatenation helper call.

7.33.2.100 emit_string_equals_call()

```
str l0_c_emitter.CEmitter.emit_string_equals_call (
              self,
             str lhs,
             str rhs )
```

Emit the runtime string-equality helper call.

7.33.2.101 emit_string_literal()

```
str l0_c_emitter.CEmitter.emit_string_literal (
            self,
            str value )
```

Emit C code for an ARC string literal.

Parameters

value	The L0 string token payload.

Returns

C L0_STRING_CONST expression.

7.33.2.102 emit_string_release()

```
None l0_c_emitter.CEmitter.emit_string_release (
            self,
            str c_expr )
```

Emit an ARC string release runtime call.

Parameters

c_expr	C expression evaluating to an l0_string.

7.33.2.103 emit_string_retain()

```
None l0_c_emitter.CEmitter.emit_string_retain (
            self,
            str c_expr )
```

Emit an ARC string retain runtime call.

Parameters

c_expr	C expression evaluating to an l0_string.

7.33.2.104 emit_struct()

```
None l0_c_emitter.CEmitter.emit_struct (
            self,
            str module_name,
            StructDecl decl,
            StructInfo struct_info )
```

Emit a complete C struct definition.

Parameters

module_name	Name of the module.
decl	StructDecl AST node.
struct_info	Resolved type information for the struct.

7.33.2.105 emit_struct_cleanup()

```
None l0_c_emitter.CEmitter.emit_struct_cleanup (
              self,
             str c_ptr_expr,
             StructType struct_type )
```

Emit cleanup code for all owned fields in a struct.

Parameters

c_ptr_expr	C expression evaluating to a pointer to the struct.
struct_type	The L0 struct type.

7.33.2.106 emit_struct_constructor()

```
str l0_c_emitter.CEmitter.emit_struct_constructor (
              self,
             str c_struct_name,
             List[Tuple[str, str]] field_inits )
```

Emit C code for a struct compound literal constructor.

Parameters

c_struct_name	Mangled C struct name.
field_inits	List of (field_name, c_value) tuples.

Returns

C compound literal: (struct name){ .f1 = v1, .f2 = v2 }.

7.33.2.107 emit_struct_constructor_for_type()

```
str l0_c_emitter.CEmitter.emit_struct_constructor_for_type (
              self,
             StructType struct_type,
             List[Tuple[str, str]] field_inits )
```

Emit a C struct constructor for an L0 struct type.

7.33.2.108 emit_struct_init()

```
None l0_c_emitter.CEmitter.emit_struct_init (
              self,
             str c_temp_name,
             str c_base_type,
             str c_init_str )
```

Emit an object initialization through a pointer.

Parameters

c_temp_name	Name of the pointer variable.
c_base_type	C base object type string.
c_init_str	C compound initializer body.

7.33.2.109 emit_struct_init_from_fields()

```
None l0_c_emitter.CEmitter.emit_struct_init_from_fields (
              self,
             str c_temp_name,
             Type base_type,
             List[Tuple[str, str]]  field_inits )
```

Emit struct initialization using positional field values.

Parameters

c_temp_name	Name of the pointer variable.
base_type	L0 base object type.
field_inits	List of (field_name, c_value) tuples.

7.33.2.110 emit_struct_static_initializer()

```
str l0_c_emitter.CEmitter.emit_struct_static_initializer (
              self,
             List[Tuple[str, str]] field_inits )
```

Emit a brace-only struct initializer for static storage duration.

7.33.2.111 emit_struct_static_initializer_for_type()

```
str l0_c_emitter.CEmitter.emit_struct_static_initializer_for_type (
              self,
             StructType struct_type,
             List[Tuple[str, str]] field_inits )
```

Emit a static-storage struct initializer for an L0 struct type.

7.33.2.112 emit_switch_end()

```
None l0_c_emitter.CEmitter.emit_switch_end (
              self )
```

Emit a C switch statement closing brace.

7.33.2.113 emit_switch_start()

```
None l0_c_emitter.CEmitter.emit_switch_start (
              self,
             str c_expr )
```

Emit a C switch statement header and opening brace.

Parameters

c_expr	C expression to switch on.

7.33.2.114 emit_temp_decl()

```
None l0_c_emitter.CEmitter.emit_temp_decl (
              self,
             str c_type,
             str c_temp_name,
             str c_value )
```

Emit a C temporary variable declaration with initializer.

Parameters

c_type	C type string.
c_temp_name	C identifier for temporary.
c_value	C initializer expression string.

7.33.2.115 emit_top_level_let_cleanup()

```
None l0_c_emitter.CEmitter.emit_top_level_let_cleanup (
              self )
```

Emit final cleanup for ARC-managed top-level lets before process exit.

7.33.2.116 emit_try_check_niche()

```
None l0_c_emitter.CEmitter.emit_try_check_niche (
              self,
             str c_tmp,
             str ret_none )
```

Emit a NULL check for a niche-optimized optional.

Parameters

c_tmp	C identifier of the temporary holding the optional.
ret_none	C expression for the 'none' return value.

7.33.2.117 emit_try_check_value()

```
None l0_c_emitter.CEmitter.emit_try_check_value (
              self,
             str c_tmp,
             str ret_none )
```

Emit a has_value check for a value-optional.

Parameters

c_tmp	C identifier of the temporary holding the optional.
ret_none	C expression for the 'none' return value.

7.33.2.118 emit_try_extract_value()

```
str l0_c_emitter.CEmitter.emit_try_extract_value (
              self,
             str c_tmp )
```

Emit C code to extract the inner value from an optional.

Parameters

c_tmp	C identifier of the temporary holding the optional.

Returns

C expression string for the extracted value.

7.33.2.119 emit_type()

```
str l0_c_emitter.CEmitter.emit_type (
              self,
             Type typ )
```

Convert an L0 Type to its C representation.

Parameters

typ	The L0 Type to convert.

Returns

A C type string (e.g., "l0_int", "struct l0_main_Point*").

Exceptions

InternalCompilerError	If the type kind is unknown or unsupported.

7.33.2.120 emit_unary_op()

```
str l0_c_emitter.CEmitter.emit_unary_op (
              self,
             str op,
             str c_operand )
```

Emit C code for a unary operation.

Parameters

op	C unary operator string.
c_operand	C expression for the operand.

Returns

C unary expression string.

7.33.2.121 emit_unreachable_comment()

```
None l0_c_emitter.CEmitter.emit_unreachable_comment (
              self )
```

Emit an unreachable code comment for debugging.

7.33.2.122 emit_unreachable_marker()

```
None l0_c_emitter.CEmitter.emit_unreachable_marker (
              self,
             str  reason = "unreachable" )
```

Emit a runtime panic call for unreachable code paths.

Parameters

reason	Description of why the path is unreachable.

7.33.2.123 emit_unwrap_opt()

```
str l0_c_emitter.CEmitter.emit_unwrap_opt (
              self,
             str c_src_type,
             str c_inner,
             str type_str )
```

Emit C code for unwrapping a value-optional runtime check.

7.33.2.124 emit_unwrap_ptr()

```
str l0_c_emitter.CEmitter.emit_unwrap_ptr (
              self,
             str c_dst_type,
             str c_inner,
             str type_str )
```

Emit C code for unwrapping a pointer-shaped optional runtime check.

7.33.2.125 emit_value_cleanup()

```
None l0_c_emitter.CEmitter.emit_value_cleanup (
              self,
             str c_expr,
             Type ty )
```

Emit C code to clean up an owned by-value variable.

Parameters

c_expr	C expression evaluating to the value.
ty	The L0 Type of the value.

7.33.2.126 emit_var_ref()

```
str l0_c_emitter.CEmitter.emit_var_ref (
              self,
             str c_name )
```

Emit C code for a variable reference identifier.

Parameters

c_name	Mangled C identifier.

Returns

The identifier string.

7.33.2.127 emit_variant_constructor()

```
str l0_c_emitter.CEmitter.emit_variant_constructor (
              self,
             str c_enum_name,
             str variant_name,
             str tag_value,
             List[Tuple[str, str]]  payload_inits )
```

Emit C code for an enum variant tagged union literal.

Parameters

c_enum_name	Mangled C enum name.
variant_name	Name of the variant.
tag_value	C tag value.
payload_inits	List of (field_name, c_value) tuples for payload.

Returns

C tagged union literal string.

7.33.2.128 emit_variant_constructor_for_type()

```
str l0_c_emitter.CEmitter.emit_variant_constructor_for_type (
              self,
             EnumType enum_type,
             str variant_name,
             List[Tuple[str, str]]  payload_inits )
```

Emit a tagged union constructor for a given L0 enum type.

7.33.2.129 emit_variant_static_initializer()

```
str l0_c_emitter.CEmitter.emit_variant_static_initializer (
              self,
             str variant_name,
             str tag_value,
             List[Tuple[str, str]] payload_inits )
```

Emit a brace-only enum tagged-union initializer for static storage duration.

7.33.2.130 emit_variant_static_initializer_for_type()

```
str l0_c_emitter.CEmitter.emit_variant_static_initializer_for_type (
              self,
             EnumType enum_type,
             str variant_name,
             List[Tuple[str, str]] payload_inits )
```

Emit a static-storage tagged union initializer for a given L0 enum type.

7.33.2.131 emit_while_header()

```
None l0_c_emitter.CEmitter.emit_while_header (
             self,
            str c_cond )
```

Emit a C while loop header.

Parameters

c_cond	C expression for the loop condition.

7.33.2.132 emit_widen_int()

```
str l0_c_emitter.CEmitter.emit_widen_int (
             self,
            str c_expr,
            BuiltinType src_type,
            BuiltinType dst_type )
```

Emit C code for implicit integer widening.

Parameters

c_expr	C expression evaluating to the source value.
src_type	The smaller source type.
dst_type	The larger destination type.

Returns

C code string for the widening cast.

Exceptions

InternalCompilerError	If the widening path is unsupported.

7.33.2.133 emit_zero_init()

```
None l0_c_emitter.CEmitter.emit_zero_init (
             self,
            str c_temp_name,
            str c_base_type )
```

Emit zero-initialization through a pointer.

Parameters

c_temp_name	Name of the pointer variable.
c_base_type	C base object type string.

7.33.2.134 find_variant_decl()

```
Optional[EnumVariant] l0_c_emitter.CEmitter.find_variant_decl (
              self,
             str module_name,
             str enum_name,
             str  variant_name )
```

Look up an enum variant's AST declaration.

Parameters

module_name	Name of the module containing the enum.
enum_name	Name of the enum.
variant_name	Name of the variant to find.

Returns

The EnumVariant node if found, otherwise None.

7.33.2.135 fresh_tmp()

```
str l0_c_emitter.CEmitter.fresh_tmp (
              self,
             str  kind = "tmp" )
```

Generate a unique temporary variable name.

Parameters

kind	Category of temporary (e.g., "tmp", "ptr", "try").

Returns

Unique C identifier like "l0_tmp_1", "l0_ptr_2", etc.

7.33.2.136 get_output()

```
str l0_c_emitter.CEmitter.get_output (
              self )
```

Get the generated C code.

Returns

The complete generated C code string.

7.33.2.137 ice()

```
NoReturn l0_c_emitter.CEmitter.ice (
               self,
              str message,
              Optional[object]  node = None )
```

Raise an internal compiler error with context.

Parameters

message	Descriptive error message.
node	Optional AST node to provide source location information.

Exceptions

InternalCompilerError	Always raised with provided context.

7.33.2.138 is_niche_nullable()

```
bool l0_c_emitter.CEmitter.is_niche_nullable (
               self,
              NullableType t )
```

Public check for niche-optimized (pointer-shaped) nullable types.

Parameters

t	The NullableType to check.

Returns

True if the type is represented as a nullable pointer in C.

7.33.2.139 mangle_enum_name()

```
str l0_c_emitter.CEmitter.mangle_enum_name (
               self,
              str module_name,
              str enum_name )
```

Mangle an enum name.

Parameters

module_name	Module name.
enum_name	L0 enum name.

Returns

Mangled C enum name.

7.33.2.140 mangle_function_name()

```
str l0_c_emitter.CEmitter.mangle_function_name (
              self,
             str module_name,
             str func_name )
```

Mangle a function name.

Parameters

module_name	Module name.
func_name	L0 function name.

Returns

Mangled C function name.

7.33.2.141 mangle_identifier()

```
str l0_c_emitter.CEmitter.mangle_identifier (
              self,
             str name )
```

Mangle an identifier if it conflicts with C keywords or L0 names.

Used for local variables, parameters, and pattern variables. Appends '__v' suffix to avoid C keyword conflicts. Also mangles names starting with '_' or 'l0_'/'L0_' to avoid clashes with runtime names.

Parameters

name	The L0 identifier to mangle.

Returns

The safe C identifier.

7.33.2.142 mangle_let_name()

```
str l0_c_emitter.CEmitter.mangle_let_name (
              self,
             str module_name,
             str let_name )
```

Mangle a top-level let name to C identifier.

Parameters

module_name	Module name.
let_name	L0 constant name.

Returns

Mangled C identifier.

7.33.2.143 mangle_struct_name()

```
str l0_c_emitter.CEmitter.mangle_struct_name (
              self,
             str module_name,
             str struct_name )
```

Mangle a struct name to avoid C namespace collisions.

Parameters

module_name	Module name.
struct_name	L0 struct name.

Returns

Mangled C struct name (e.g., "l0_module_Point").

7.33.2.144 prepare_optional_wrappers()

```
None l0_c_emitter.CEmitter.prepare_optional_wrappers (
              self )
```

Scan all compilation unit types and collect required optional wrappers.

7.33.2.145 restore_active_line_directive()

```
None l0_c_emitter.CEmitter.restore_active_line_directive (
              self )
```

Re-emit the current source line directive after generated helper lines.

7.33.2.146 set_analysis()

```
None l0_c_emitter.CEmitter.set_analysis (
              self,
             AnalysisResult analysis )
```

Initialize emitter with analysis data.

Parameters

analysis	The AnalysisResult containing the compilation products.

7.33.3 Member Data Documentation

7.33.3.1 _active_line_directive [1/2]

```
Optional l0_c_emitter.CEmitter._active_line_directive = None  [static], [protected]
```

7.33.3.2 _active_line_directive [2/2]

```
l0_c_emitter.CEmitter._active_line_directive  [protected]
```

7.33.3.3 _opt_emitted

```
Set l0_c_emitter.CEmitter._opt_emitted = field(default_factory=set)  [static], [protected]
```

7.33.3.4 _opt_wrappers

```
Dict l0_c_emitter.CEmitter._opt_wrappers = field(default_factory=dict)  [static], [protected]
```

7.33.3.5 _tmp_counter

```
int l0_c_emitter.CEmitter._tmp_counter = 0  [static], [protected]
```

7.33.3.6 analysis [1/2]

```
Optional l0_c_emitter.CEmitter.analysis = None  [static]
```

7.33.3.7 analysis [2/2]

```
l0_c_emitter.CEmitter.analysis
```

7.33.3.8 C_KEYWORDS

```
Set l0_c_emitter.CEmitter.C_KEYWORDS  [static]
```

Initial value:

```
=  field(default_factory=lambda: {
        # C89/C99 keywords
        'auto', 'break', 'case', 'char', 'const', 'continue', 'default', 'do',
        'double', 'else', 'enum', 'extern', 'float', 'for', 'goto', 'if',
        'inline', 'int', 'long', 'register', 'restrict', 'return', 'short',
        'signed', 'sizeof', 'static', 'struct', 'switch', 'typedef', 'union',
        'unsigned', 'void', 'volatile', 'while',
        # C23 additions
        'alignas', 'alignof', 'atomic', 'bool', 'complex', 'imaginary',
        # Other identifiers to avoid
        'NULL', 'null', 'bool', 'true', 'false', 'asm', 'offsetof', 'typeof',
    })
```

7.33.3.9 current_module

```
Optional l0_c_emitter.CEmitter.current_module = None  [static]
```

7.33.3.10 out

```
CCodeBuilder l0_c_emitter.CEmitter.out = field(default_factory=CCodeBuilder)  [static]
```

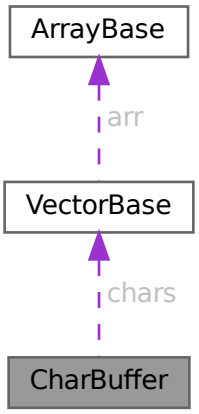
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_c_emitter.py

7.34 CharBuffer Struct Reference

CharBuffer allows for efficient building of strings by accumulating individual characters.

Collaboration diagram for CharBuffer:



Public Attributes

- chars: VectorBase*

7.34.1 Detailed Description

CharBuffer allows for efficient building of strings by accumulating individual characters.

7.34.2 Member Data Documentation

7.34.2.1 chars

chars: `chars: VectorBase*`

The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/text.l0

7.35 CivilDate Struct Reference

Represents a civil date without time components.

Public Attributes

- year: int
- month: int
- day: int
- weekday: int
- yearday: int

7.35.1 Detailed Description

Represents a civil date without time components.

7.35.2 Member Data Documentation

7.35.2.1 day

day: `day: int`

7.35.2.2 month

month: `month: int`

7.35.2.3 weekday

weekday: `weekday: int`

7.35.2.4 year

year: `year: int`

7.35.2.5 yearday

yearday: `yearday: int`

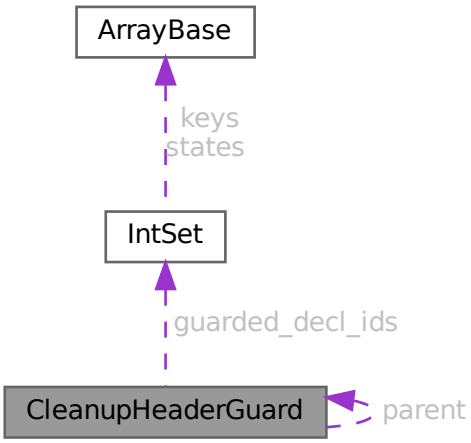
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/time.l0

7.36 CleanupHeaderGuard Struct Reference

Linked stack entry for guarded `with` header locals during cleanup-block checking.

Collaboration diagram for CleanupHeaderGuard:



Public Attributes

- parent: CleanupHeaderGuard∗?
- guarded_decl_ids: IntSet∗

7.36.1 Detailed Description

Linked stack entry for guarded `with` header locals during cleanup-block checking.

7.36.2 Member Data Documentation

7.36.2.1 guarded_decl_ids

guarded_decl_ids: `guarded_decl_ids: IntSet∗`

7.36.2.2 parent

parent: `parent: CleanupHeaderGuard*?`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/expr_types.l0

7.37 CliOptions Struct Reference

Represents the CliOptions structure.

Public Attributes

- mode: CliMode
- mode_explicit: bool
- verbosity: int
- log: bool
- project_roots: StringVector*
- sys_roots: StringVector*
- interface_paths: StringVector*
- output: string?
- keep_c: bool
- c_compiler: string?
- c_options: string?
- c_sources: StringVector*
- runtime_include: string?
- runtime_lib: string?
- no_line_directives: bool
- trace_arc: bool
- trace_memory: bool
- unchecked: bool
- check_basic: bool
- all_modules: bool
- include_eof: bool
- entry_target: string?
- extra_target_count: int
- program_args: StringVector*

7.37.1 Detailed Description

Represents the CliOptions structure.

7.37.2 Member Data Documentation

7.37.2.1 all_modules

all_modules: `all_modules: bool`

7.37.2.2 c_compiler

c_compiler: `c_compiler: string?`

7.37.2.3 c_options

c_options: `c_options: string?`

7.37.2.4 c_sources

c_sources: `c_sources: StringVector*`

7.37.2.5 check_basic

check_basic: `check_basic: bool`

7.37.2.6 entry_target

entry_target: `entry_target: string?`

7.37.2.7 extra_target_count

extra_target_count: `extra_target_count: int`

7.37.2.8 include_eof

include_eof: `include_eof: bool`

7.37.2.9 interface_paths

interface_paths: `interface_paths: StringVector*`

7.37.2.10 keep_c

keep_c: `keep_c: bool`

7.37.2.11 log

log: `log: bool`

7.37.2.12 mode

mode: `mode: CliMode`

7.37.2.13 mode_explicit

mode_explicit: `mode_explicit: bool`

7.37.2.14 no_line_directives

no_line_directives: `no_line_directives: bool`

7.37.2.15 output

output: `output: string?`

7.37.2.16 program_args

program_args: `program_args: StringVector*`

7.37.2.17 project_roots

project_roots: `project_roots: StringVector*`

7.37.2.18 runtime_include

runtime_include: `runtime_include: string?`

7.37.2.19 runtime_lib

runtime_lib: `runtime_lib: string?`

7.37.2.20 sys_roots

sys_roots: `sys_roots: StringVector*`

7.37.2.21 trace_arc

trace_arc: `trace_arc: bool`

7.37.2.22 trace_memory

trace_memory: `trace_memory: bool`

7.37.2.23 unchecked

unchecked: `unchecked: bool`

7.37.2.24 verbosity

verbosity: `verbosity: int`

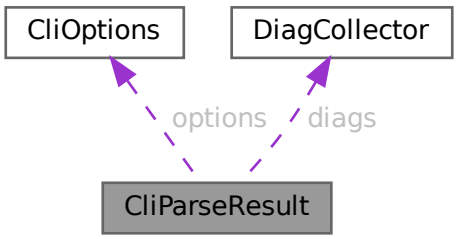
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/cli_args.l0

7.38 CliParseResult Struct Reference

Represents the CliParseResult structure.

Collaboration diagram for CliParseResult:



Public Attributes

- options: CliOptions∗?
- diags: DiagCollector∗
- help_requested: bool
- version_requested: bool
- verbosity: int
- rich_log: bool

7.38.1 Detailed Description

Represents the CliParseResult structure.

7.38.2 Member Data Documentation

7.38.2.1 diags

diags: `diags: DiagCollector*`

7.38.2.2 help_requested

help_requested: `help_requested: bool`

7.38.2.3 options

options: `options: CliOptions*?`

7.38.2.4 rich_log

rich_log: `rich_log: bool`

7.38.2.5 verbosity

verbosity: `verbosity: int`

7.38.2.6 version_requested

version_requested: `version_requested: bool`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/cli_args.l0

7.39 CodegenOptions Struct Reference

Represents backend and emitter options that affect generated C text.

Public Attributes

- emit_line_directives: bool
- trace_arc: bool
- trace_memory: bool
- rt_unchecked: bool
- rt_check_basic: bool

7.39.1 Detailed Description

Represents backend and emitter options that affect generated C text.

7.39.2 Member Data Documentation

7.39.2.1 emit_line_directives

emit_line_directives: `emit_line_directives: bool`

7.39.2.2 rt_check_basic

rt_check_basic: `rt_check_basic: bool`

7.39.2.3 rt_unchecked

rt_unchecked: `rt_unchecked: bool`

7.39.2.4 trace_arc

trace_arc: `trace_arc: bool`

7.39.2.5 trace_memory

trace_memory: `trace_memory: bool`

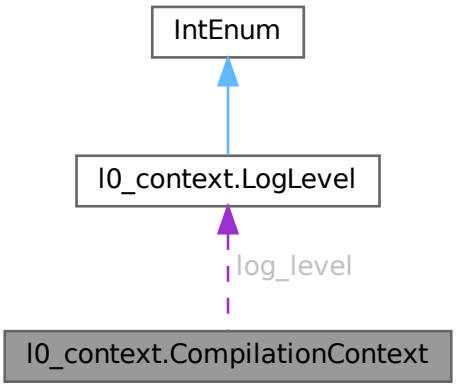
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/codegen_options.l0

7.40 l0_context.CompilationContext Class Reference

Cross-cutting compiler options affecting multiple compilation stages.

Collaboration diagram for l0_context.CompilationContext:



Static Public Member Functions

- 'CompilationContext' default ()

  Create a CompilationContext with default settings.

Static Public Attributes

- bool emit_line_directives = True
- bool trace_arc = False
- bool trace_memory = False
- bool rt_unchecked = False
- bool rt_check_basic = False
- bool log_rich_format = False
- LogLevel log_level = LogLevel.WARNING

7.40.1 Detailed Description

Cross-cutting compiler options affecting multiple compilation stages.

Attributes

- `emit_line_directives`: If True, emit #line directives in generated C code for better debugging and error messages.
- `trace_arc`: If True, generated C enables ARC tracing in runtime (`L0_TRACE_ARC`).
- `trace_memory`: If True, generated C enables memory tracing in runtime (`L0_TRACE_MEMORY`).
- `rt_unchecked`: If True, generated C disables runtime pointer validation and allocation tracking (`L0_RT_UNCHECKED`).
- `rt_check_basic`: If True, generated C enables basic checked runtime pointer validation (`L0_RT_CHECK_BASIC`).
- `log_rich_format`: If True, emit logs in rich format: may include log level, timestamps, etc.
- `log_level`: Current logging level.

7.40.2 Member Function Documentation

7.40.2.1 default()

```
'CompilationContext' l0_context.CompilationContext.default ( )  [static]
```

Create a CompilationContext with default settings.

Returns

A new CompilationContext instance with default values.

7.40.3 Member Data Documentation

7.40.3.1 emit_line_directives

```
bool l0_context.CompilationContext.emit_line_directives = True  [static]
```

7.40.3.2 log_level

```
LogLevel l0_context.CompilationContext.log_level = LogLevel.WARNING  [static]
```

7.40.3.3 log_rich_format

```
bool l0_context.CompilationContext.log_rich_format = False  [static]
```

7.40.3.4 rt_check_basic

```
bool l0_context.CompilationContext.rt_check_basic = False  [static]
```

7.40.3.5 rt_unchecked

```
bool l0_context.CompilationContext.rt_unchecked = False  [static]
```

7.40.3.6 trace_arc

```
bool l0_context.CompilationContext.trace_arc = False  [static]
```

7.40.3.7 trace_memory

```
bool l0_context.CompilationContext.trace_memory = False  [static]
```

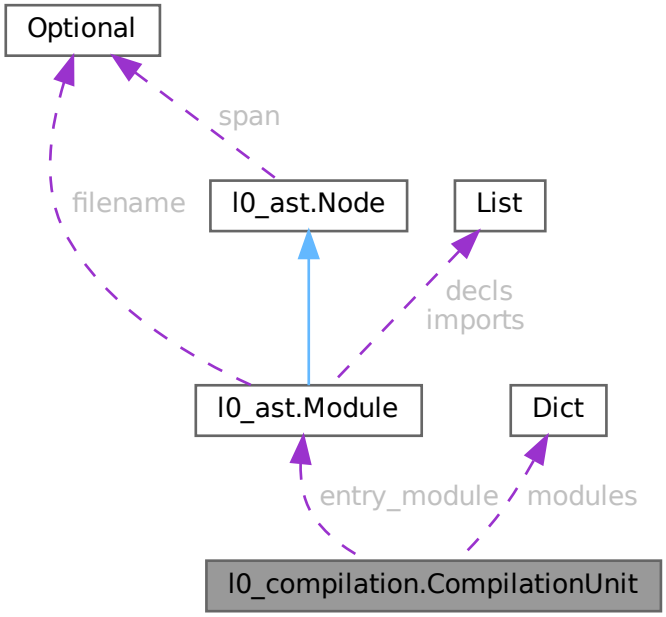
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_context.py

7.41 l0_compilation.CompilationUnit Class Reference

A closed set of modules starting from an entry module.

Collaboration diagram for l0_compilation.CompilationUnit:



Public Member Functions

- str entry_name (self)

  Get the name of the entry module.
- bool __contains__ (self, str module_name)

  Check if a module is part of this compilation unit.

Static Public Attributes

- Module entry_module
- Dict modules [str, Module]

7.41.1 Detailed Description

A closed set of modules starting from an entry module.

Attributes

- `entry_module`: The root module of the compilation (e.g., 'app.main').
- `modules`: Mapping of module names to Module objects for all transitively imported modules, including the entry module.

7.41.2 Member Function Documentation

7.41.2.1 __contains__()

```
bool l0_compilation.CompilationUnit.__contains__ (
              self,
             str module_name )
```

Check if a module is part of this compilation unit.

7.41.2.2 entry_name()

```
str l0_compilation.CompilationUnit.entry_name (
              self )
```

Get the name of the entry module.

7.41.3 Member Data Documentation

7.41.3.1 entry_module

```
Module l0_compilation.CompilationUnit.entry_module  [static]
```

7.41.3.2 modules

```
Dict l0_compilation.CompilationUnit.modules [str, Module]  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_compilation.py

7.42 CompilerWorkspace Struct Reference

Invocation-owned native build/run workspace.

Public Attributes

- root: string
- children: StringVector∗

7.42.1 Detailed Description

Invocation-owned native build/run workspace.

7.42.2 Member Data Documentation

7.42.2.1 children

children: `children: StringVector*`

7.42.2.2 root

root: `root: string`

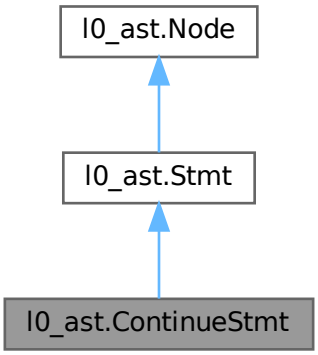
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/compiler_filesystem.l0

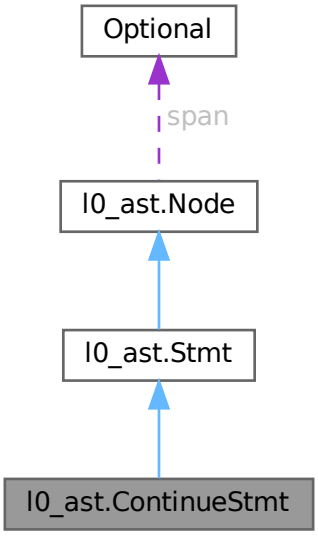
7.43 l0_ast.ContinueStmt Class Reference

A continue statement.

Inheritance diagram for l0_ast.ContinueStmt:



Collaboration diagram for l0_ast.ContinueStmt:



Additional Inherited Members

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.43.1 Detailed Description

A continue statement.

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.44 DateTime Struct Reference

Represents a broken-down date and time.

Public Attributes

- year: int
- month: int
- day: int
- hour: int
- minute: int
- second: int
- nanosecond: int
- weekday: int
- yearday: int
- utc_offset_sec: int
- is_dst: bool

7.44.1 Detailed Description

Represents a broken-down date and time.

7.44.2 Member Data Documentation

7.44.2.1 day

day: `day: int`

7.44.2.2 hour

hour: `hour: int`

7.44.2.3 is_dst

is_dst: `is_dst: bool`

7.44.2.4 minute

minute: `minute: int`

7.44.2.5 month

month: `month: int`

7.44.2.6 nanosecond

nanosecond: `nanosecond: int`

7.44.2.7 second

second: `second: int`

7.44.2.8 utc_offset_sec

utc_offset_sec: `utc_offset_sec: int`

7.44.2.9 weekday

weekday: `weekday: int`

7.44.2.10 year

year: `year: int`

7.44.2.11 yearday

yearday: `yearday: int`

The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/time.l0

7.45 DiagCollector Struct Reference

Represents the DiagCollector structure.

Public Attributes

- items: DiagnosticVector*
- error_count: int

7.45.1 Detailed Description

Represents the DiagCollector structure.

7.45.2 Member Data Documentation

7.45.2.1 error_count

error_count: `error_count: int`

7.45.2.2 items

items: `items: DiagnosticVector*`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/util/diag.l0

7.46 Diagnostic Struct Reference

Diagnostic represents an error, warning, or note message produced during compilation.

Public Attributes

- phase: string
- code: string
- severity: DiagnosticSeverity
- message: string
- filename: string
- line: int
- column: int
- line_end: int
- column_end: int

7.46.1 Detailed Description

Diagnostic represents an error, warning, or note message produced during compilation.

It includes information about the phase of compilation, a code for the specific issue, the severity level, a human-readable message, and the location in the source code.

7.46.2 Member Data Documentation

7.46.2.1 code

code: `code: string`

7.46.2.2 column

column: `column: int`

7.46.2.3 column_end

column_end: `column_end: int`

7.46.2.4 filename

filename: `filename: string`

7.46.2.5 line

line: `line: int`

7.46.2.6 line_end

line_end: `line_end: int`

7.46.2.7 message

message: `message: string`

7.46.2.8 phase

phase: `phase: string`

7.46.2.9 severity

severity: `severity: DiagnosticSeverity`

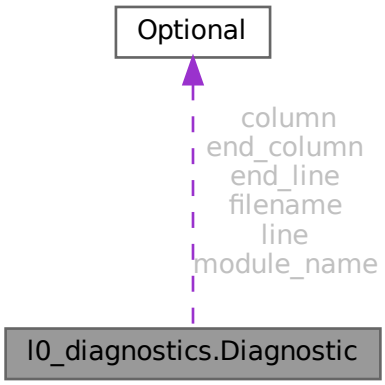
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/util/diag.l0

7.47 l0_diagnostics.Diagnostic Class Reference

Represents a compiler diagnostic (error or warning).

Collaboration diagram for l0_diagnostics.Diagnostic:



Public Member Functions

- str format (self)

  Format the diagnostic as a one-line string header.

Static Public Attributes

- str kind
- str message
- Optional module_name = None
- Optional filename = None
- Optional line = None
- Optional column = None
- Optional end_line = None
- Optional end_column = None

7.47.1 Detailed Description

Represents a compiler diagnostic (error or warning).

Attributes

- `kind`: The kind of diagnostic ("error" or "warning").
- `message`: The diagnostic message.
- `module_name`: Optional name of the module where the diagnostic occurred.
- `filename`: Optional file path where the diagnostic occurred.
- `line`: 1-based line number of the start of the diagnostic.
- `column`: 1-based column number of the start of the diagnostic.
- `end_line`: 1-based line number of the end of the diagnostic.
- `end_column`: 1-based column number of the end of the diagnostic.

7.47.2 Member Function Documentation

7.47.2.1 format()

```
str l0_diagnostics.Diagnostic.format (
              self )
```

Format the diagnostic as a one-line string header.

Returns

A formatted string containing location, kind, and message.

7.47.3 Member Data Documentation

7.47.3.1 column

```
Optional l0_diagnostics.Diagnostic.column = None  [static]
```

7.47.3.2 end_column

```
Optional l0_diagnostics.Diagnostic.end_column = None  [static]
```

7.47.3.3 end_line

```
Optional l0_diagnostics.Diagnostic.end_line = None  [static]
```

7.47.3.4 filename

```
Optional l0_diagnostics.Diagnostic.filename = None  [static]
```

7.47.3.5 kind

```
str l0_diagnostics.Diagnostic.kind  [static]
```

7.47.3.6 line

```
Optional l0_diagnostics.Diagnostic.line = None  [static]
```

7.47.3.7 message

```
str l0_diagnostics.Diagnostic.message  [static]
```

7.47.3.8 module_name

```
Optional l0_diagnostics.Diagnostic.module_name = None  [static]
```

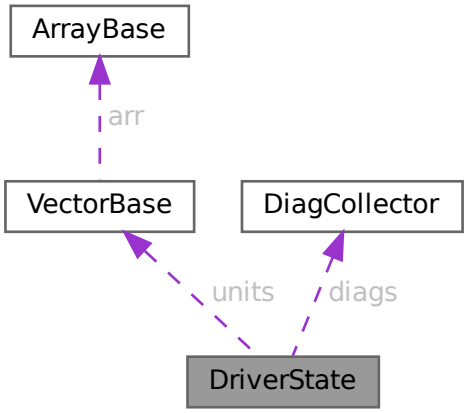
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_diagnostics.py

7.48 DriverState Struct Reference

Represents the DriverState structure.

Collaboration diagram for DriverState:



Public Attributes

- entry_module: string
- units: VectorBase∗
- loaded: StringVector∗
- diags: DiagCollector∗
- source_names: StringVector∗
- source_texts: StringVector∗

7.48.1 Detailed Description

Represents the DriverState structure.

7.48.2 Member Data Documentation

7.48.2.1 diags

diags: `diags: DiagCollector∗`

7.48.2.2 entry_module

entry_module: `entry_module: string`

7.48.2.3 loaded

loaded: `loaded: StringVector*`

7.48.2.4 source_names

source_names: `source_names: StringVector*`

7.48.2.5 source_texts

source_texts: `source_texts: StringVector*`

7.48.2.6 units

units: `units: VectorBase*`

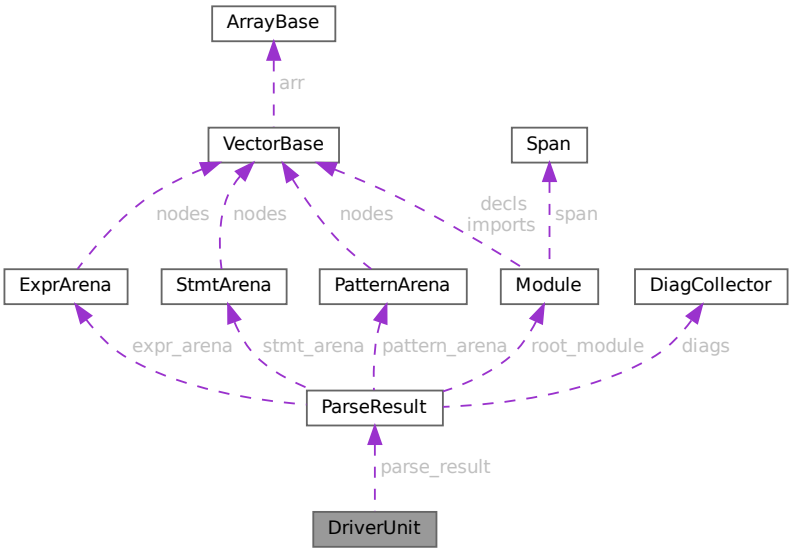
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/driver.l0

7.49 DriverUnit Struct Reference

Represents the DriverUnit structure.

Collaboration diagram for DriverUnit:



Public Attributes

- module_name: string
- path: string
- parse_result: ParseResult*

7.49.1 Detailed Description

Represents the DriverUnit structure.

7.49.2 Member Data Documentation

7.49.2.1 module_name

module_name: `module_name: string`

7.49.2.2 parse_result

parse_result: `parse_result: ParseResult*`

7.49.2.3 path

path: `path: string`

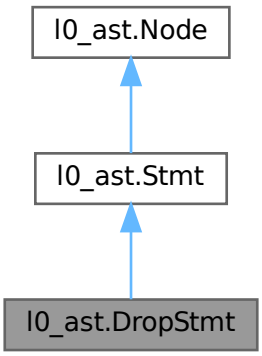
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/driver.l0

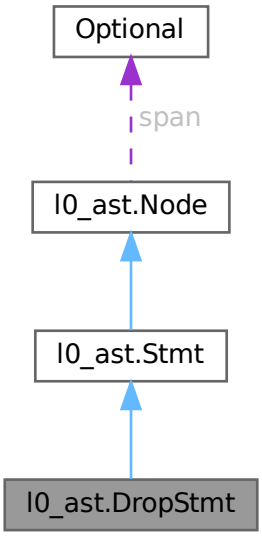
7.50 l0_ast.DropStmt Class Reference

An explicit drop statement for manual memory/resource release.

Inheritance diagram for l0_ast.DropStmt:



Collaboration diagram for l0_ast.DropStmt:



Static Public Attributes

- str name

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.50.1 Detailed Description

An explicit drop statement for manual memory/resource release.

Attributes

- `name`: The name of the variable to drop.

7.50.2 Member Data Documentation

7.50.2.1 name

```
str l0_ast.DropStmt.name  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.51 Duration Struct Reference

Represents a duration of time in seconds and nanoseconds.

Public Attributes

- sec: int
- nsec: int

7.51.1 Detailed Description

Represents a duration of time in seconds and nanoseconds.

7.51.2 Member Data Documentation

7.51.2.1 nsec

nsec: `nsec: int`

7.51.2.2 sec

sec: `sec: int`

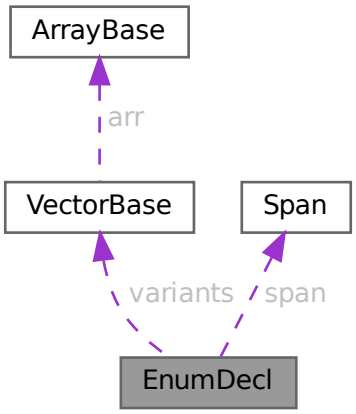
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/time.l0

7.52 EnumDecl Struct Reference

Represents the EnumDecl structure.

Collaboration diagram for EnumDecl:



Public Attributes

- name: string
- variants: VectorBase*
- span: Span

7.52.1 Detailed Description

Represents the EnumDecl structure.

7.52.2 Member Data Documentation

7.52.2.1 name

name: `name: string`

7.52.2.2 span

span: `span: Span`

7.52.2.3 variants

variants: `variants: VectorBase*`

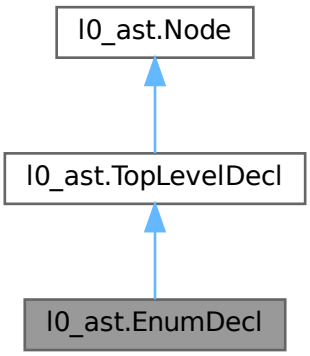
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

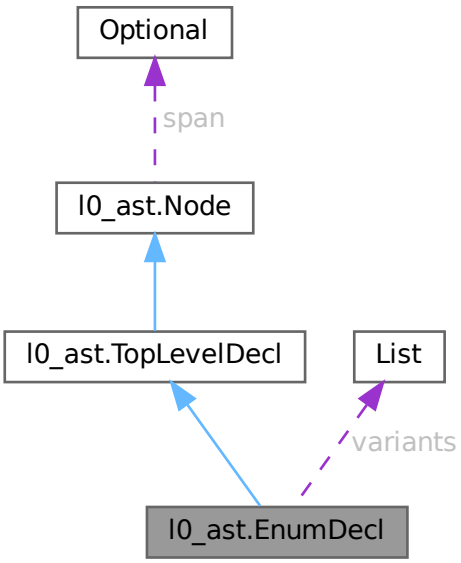
7.53 l0_ast.EnumDecl Class Reference

An enum declaration.

Inheritance diagram for l0_ast.EnumDecl:



Collaboration diagram for l0_ast.EnumDecl:



Static Public Attributes

- str name
- List variants [EnumVariant]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.53.1 Detailed Description

An enum declaration.

Attributes

- `name`: The name of the enum.
- `variants`: List of enum variant declarations.

7.53.2 Member Data Documentation

7.53.2.1 name

```
str l0_ast.EnumDecl.name  [static]
```

7.53.2.2 variants

```
List l0_ast.EnumDecl.variants [EnumVariant]  [static]
```

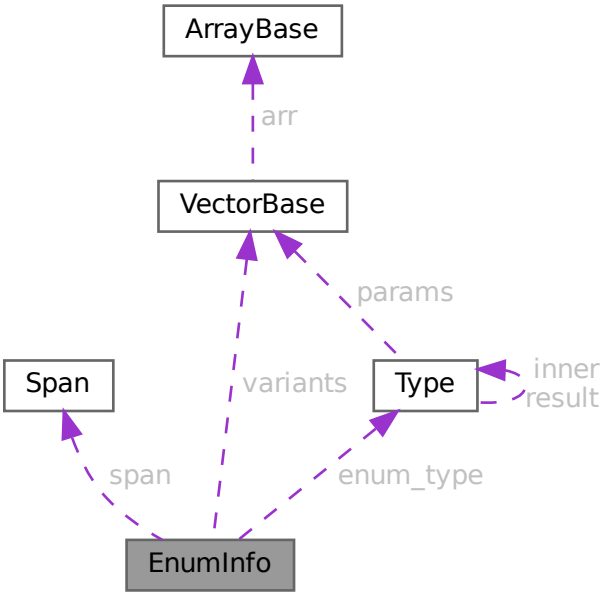
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.54 EnumInfo Struct Reference

Represents the EnumInfo structure.

Collaboration diagram for EnumInfo:



Public Attributes

- key: string
- module_name: string
- name: string
- filename: string
- span: Span
- enum_type: Type*
- variants: VectorBase*

7.54.1 Detailed Description

Represents the EnumInfo structure.

7.54.2 Member Data Documentation

7.54.2.1 enum_type

enum_type: `enum_type: Type*`

7.54.2.2 filename

filename: `filename: string`

7.54.2.3 key

key: `key: string`

7.54.2.4 module_name

module_name: `module_name: string`

7.54.2.5 name

name: `name: string`

7.54.2.6 span

span: `span: Span`

7.54.2.7 variants

variants: `variants: VectorBase*`

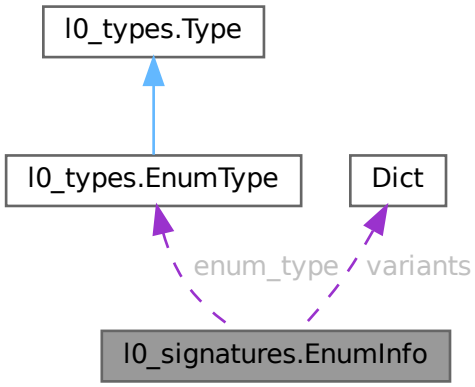
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/types.l0

7.55 l0_signatures.EnumInfo Class Reference

Resolved type information for an enum.

Collaboration diagram for l0_signatures.EnumInfo:



Static Public Attributes

- EnumType enum_type
- Dict variants [str, EnumVariantInfo]

7.55.1 Detailed Description

Resolved type information for an enum.

Attributes

- `enum_type`: The base EnumType.
- `variants`: Mapping from variant names to their variant info.

7.55.2 Member Data Documentation

7.55.2.1 enum_type

```
EnumType l0_signatures.EnumInfo.enum_type  [static]
```

7.55.2.2 variants

```
Dict l0_signatures.EnumInfo.variants [str, EnumVariantInfo]  [static]
```

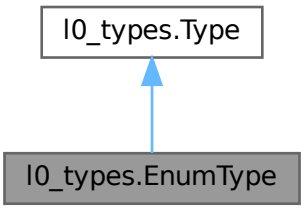
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_signatures.py

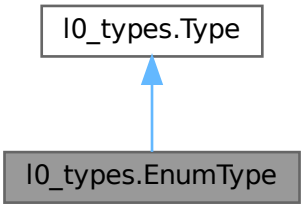
7.56 l0_types.EnumType Class Reference

A user-defined enum type.

Inheritance diagram for l0_types.EnumType:



Collaboration diagram for l0_types.EnumType:



Static Public Attributes

- str module
- str name

7.56.1 Detailed Description

A user-defined enum type.

Attributes

- `module`: The name of the module where the enum is defined.
- `name`: The name of the enum.

7.56.2 Member Data Documentation

7.56.2.1 module

```
str l0_types.EnumType.module  [static]
```

7.56.2.2 name

```
str l0_types.EnumType.name  [static]
```

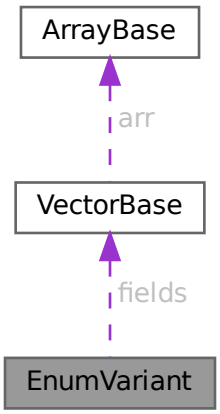
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_types.py

7.57 EnumVariant Struct Reference

Represents the EnumVariant structure.

Collaboration diagram for EnumVariant:



Public Attributes

- name: string
- fields: VectorBase∗

7.57.1 Detailed Description

Represents the EnumVariant structure.

7.57.2 Member Data Documentation

7.57.2.1 fields

fields: `fields: VectorBase*`

7.57.2.2 name

name: `name: string`

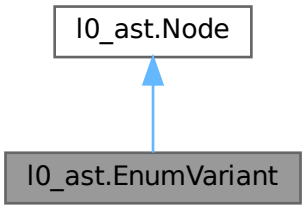
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

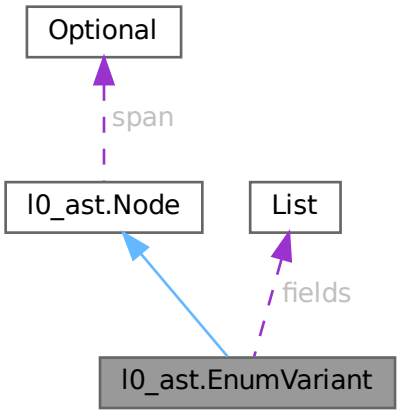
7.58 l0_ast.EnumVariant Class Reference

A variant declaration within an enum.

Inheritance diagram for l0_ast.EnumVariant:



Collaboration diagram for l0_ast.EnumVariant:



Static Public Attributes

- str name
- List fields [FieldDecl]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.58.1 Detailed Description

A variant declaration within an enum.

Attributes

- `name`: The name of the variant.
- `fields`: List of payload fields for the variant.

7.58.2 Member Data Documentation

7.58.2.1 fields

```
List l0_ast.EnumVariant.fields [FieldDecl]  [static]
```

7.58.2.2 name

```
str l0_ast.EnumVariant.name  [static]
```

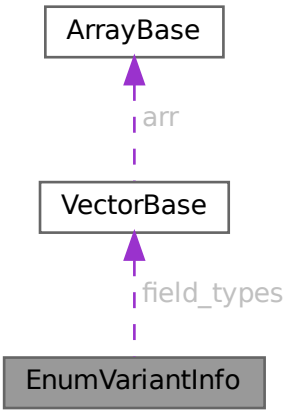
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.59 EnumVariantInfo Struct Reference

Represents the EnumVariantInfo structure.

Collaboration diagram for EnumVariantInfo:



Public Attributes

- name: string
- field_types: VectorBase∗

7.59.1 Detailed Description

Represents the EnumVariantInfo structure.

7.59.2 Member Data Documentation

7.59.2.1 field_types

field_types: `field_types: VectorBase∗`

7.59.2.2 name

name: `name: string`

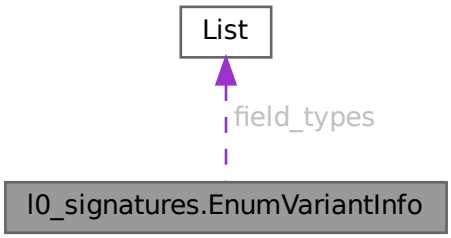
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/types.l0

7.60 l0_signatures.EnumVariantInfo Class Reference

Information about a single enum variant.

Collaboration diagram for l0_signatures.EnumVariantInfo:



Static Public Attributes

- str name
- List field_types [Type]

7.60.1 Detailed Description

Information about a single enum variant.

Attributes

- `name`: The name of the variant.
- `field_types`: List of resolved types for the variant's payload fields.

7.60.2 Member Data Documentation

7.60.2.1 field_types

```
List l0_signatures.EnumVariantInfo.field_types [Type]  [static]
```

7.60.2.2 name

```
str l0_signatures.EnumVariantInfo.name  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_signatures.py

7.61 EscapedChar Struct Reference

Represents the EscapedChar structure.

Public Attributes

- escape_str: string
- char_value: int

7.61.1 Detailed Description

Represents the EscapedChar structure.

7.61.2 Member Data Documentation

7.61.2.1 char_value

char_value: `char_value:  int`

7.61.2.2 escape_str

escape_str: `escape_str: string`

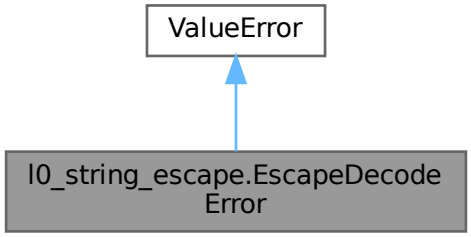
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/lexer.l0

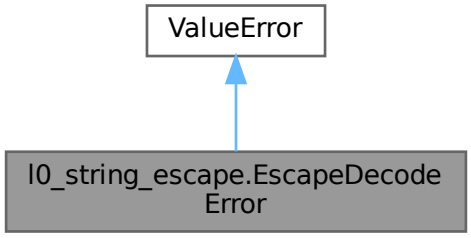
7.62 l0_string_escape.EscapeDecodeError Class Reference

Raised when an invalid escape sequence is encountered during decoding.

Inheritance diagram for l0_string_escape.EscapeDecodeError:



Collaboration diagram for l0_string_escape.EscapeDecodeError:



Static Public Attributes

- str code
- str details = ""

7.62.1 Detailed Description

Raised when an invalid escape sequence is encountered during decoding.

Attributes

- `code`: Error code identifying the kind of failure.
- `details`: Optional string providing more context.

7.62.2 Member Data Documentation

7.62.2.1 code

```
str l0_string_escape.EscapeDecodeError.code  [static]
```

7.62.2.2 details

```
str l0_string_escape.EscapeDecodeError.details = ""  [static]
```

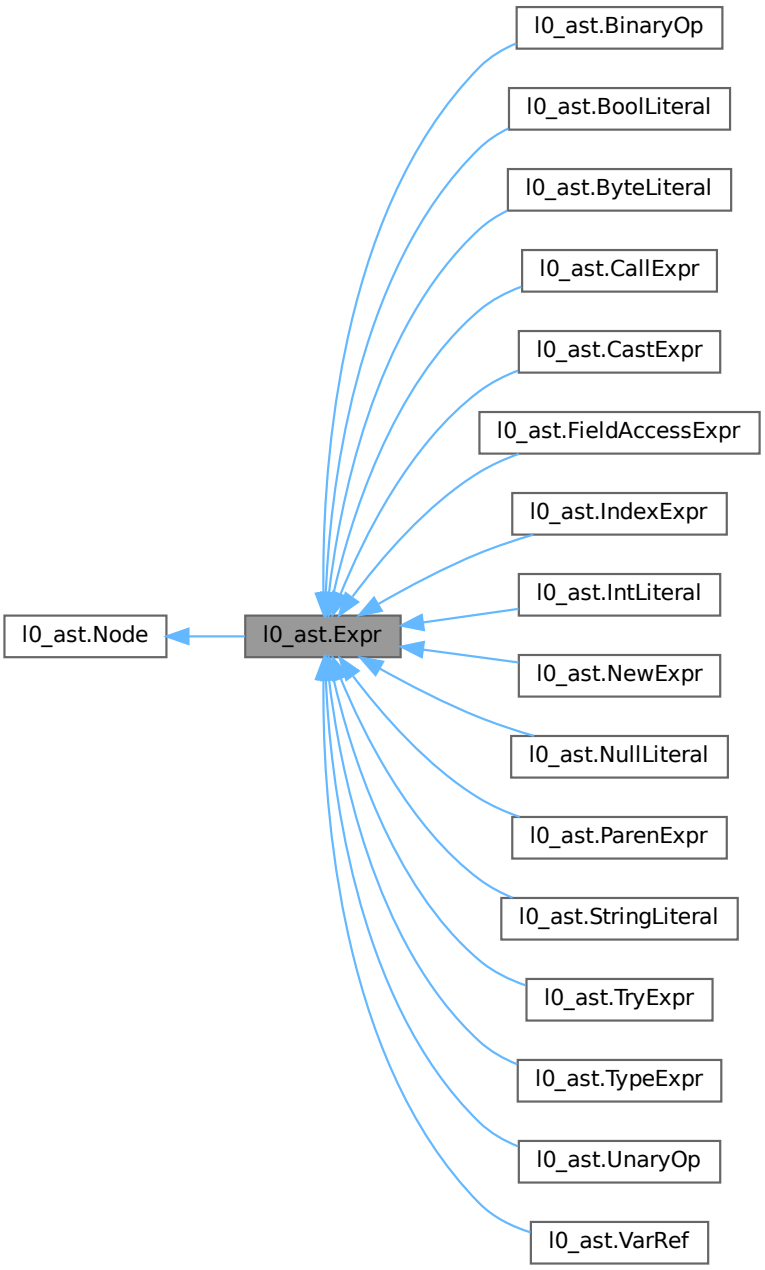
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_string_escape.py

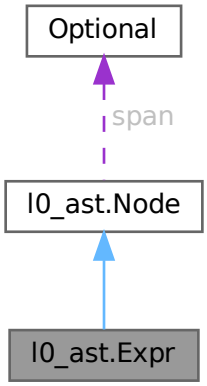
7.63 l0_ast.Expr Class Reference

Base class for all expressions.

Inheritance diagram for l0_ast.Expr:



Collaboration diagram for l0_ast.Expr:



Additional Inherited Members

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.63.1 Detailed Description

Base class for all expressions.

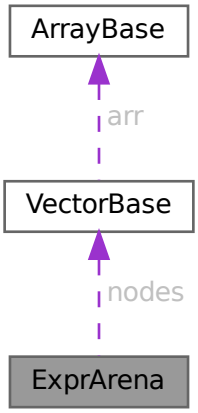
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.64 ExprArena Struct Reference

Represents the ExprArena structure.

Collaboration diagram for ExprArena:



Public Attributes

- nodes: VectorBase*

7.64.1 Detailed Description

Represents the ExprArena structure.

7.64.2 Member Data Documentation

7.64.2.1 nodes

nodes: `nodes: VectorBase*`

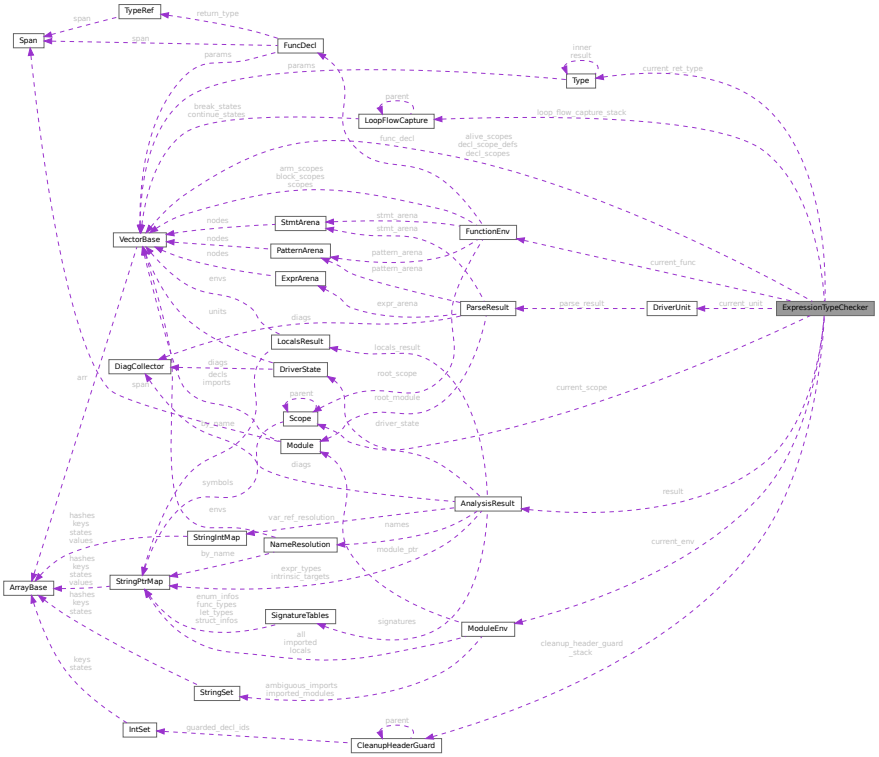
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

7.65 ExpressionTypeChecker Struct Reference

Represents the ExpressionTypeChecker structure.

Collaboration diagram for ExpressionTypeChecker:



Public Attributes

- result: AnalysisResult∗
- current_module: string
- current_unit: DriverUnit∗?
- current_env: ModuleEnv∗?
- current_func: FunctionEnv∗?
- current_ret_type: Type∗?
- current_scope: Scope∗?
- alive_scopes: VectorBase∗?
- decl_scopes: VectorBase∗?
- decl_scope_defs: VectorBase∗?
- loop_depth: int
- next_stmt_unreachable: bool
- suppress_diagnostics: bool
- suppress_liveness_diagnostics: bool
- liveness_diagnostics_only: bool
- loop_flow_capture_stack: LoopFlowCapture∗?
- cleanup_header_guard_stack: CleanupHeaderGuard∗?

7.65.1 Detailed Description

Represents the ExpressionTypeChecker structure.

7.65.2 Member Data Documentation

7.65.2.1 alive_scopes

alive_scopes: `alive_scopes : VectorBase*?`

7.65.2.2 cleanup_header_guard_stack

cleanup_header_guard_stack: `cleanup_header_guard_stack : CleanupHeaderGuard*?`

7.65.2.3 current_env

current_env: `current_env : ModuleEnv*?`

7.65.2.4 current_func

current_func: `current_func : FunctionEnv*?`

7.65.2.5 current_module

current_module: `current_module : string`

7.65.2.6 current_ret_type

current_ret_type: `current_ret_type : Type*?`

7.65.2.7 current_scope

current_scope: `current_scope : Scope*?`

7.65.2.8 current_unit

current_unit: `current_unit : DriverUnit*?`

7.65.2.9 decl_scope_defs

decl_scope_defs: `decl_scope_defs : VectorBase*?`

7.65.2.10 decl_scopes

decl_scopes: `decl_scopes: VectorBase*?`

7.65.2.11 liveness_diagnostics_only

liveness_diagnostics_only: `liveness_diagnostics_only: bool`

7.65.2.12 loop_depth

loop_depth: `loop_depth: int`

7.65.2.13 loop_flow_capture_stack

loop_flow_capture_stack: `loop_flow_capture_stack: LoopFlowCapture*?`

7.65.2.14 next_stmt_unreachable

next_stmt_unreachable: `next_stmt_unreachable: bool`

7.65.2.15 result

result: `result: AnalysisResult*`

7.65.2.16 suppress_diagnostics

suppress_diagnostics: `suppress_diagnostics: bool`

7.65.2.17 suppress_liveness_diagnostics

suppress_liveness_diagnostics: `suppress_liveness_diagnostics: bool`

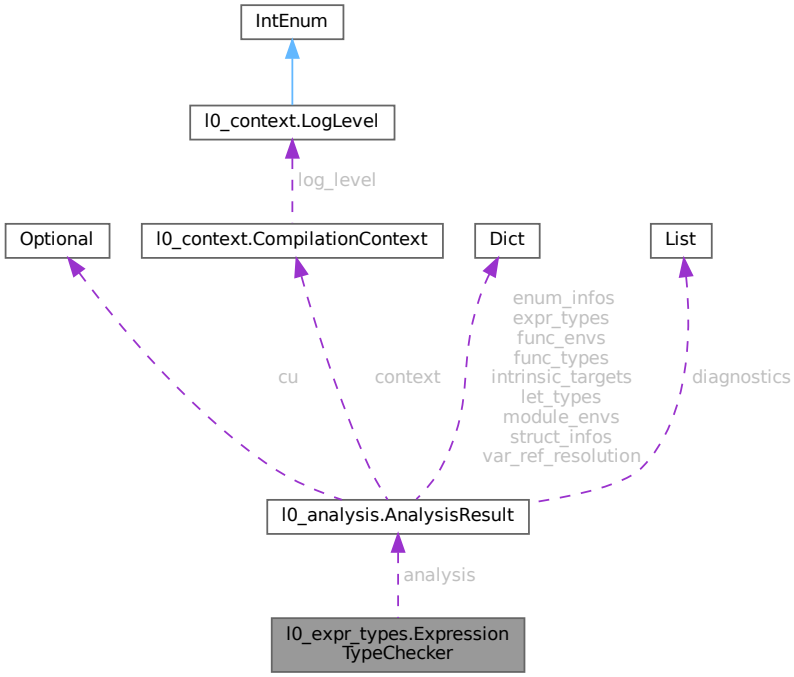
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/expr_types.l0

7.66 l0_expr_types.ExpressionTypeChecker Class Reference

Expression-level type checker for L0.

Collaboration diagram for l0_expr_types.ExpressionTypeChecker:



Public Member Functions

- None __post_init__ (self)

  Initialize checker with analysis data and cache builtin types.
- None check (self)

  Run expression/type checking for all non-extern functions.

Public Attributes

- struct_infos
- enum_infos
- func_types
- func_envs
- diagnostics
- expr_types
- module_envs
- void_type
- int_type
- byte_type
- bool_type
- string_type

Static Public Attributes

- AnalysisResult analysis

Protected Member Functions

- None _check_function (self, FunctionEnv func_env, FuncType func_type)

  Check a single function definition.
- Dict[str, Type] _make_param_scope (self, FunctionEnv func_env, FuncType func_type)

  Create a name-to-type mapping for function parameters.
- Dict[str, bool] _make_param_alive_scope (self, FunctionEnv func_env)

  Initialize liveness tracking for function parameters.
- None _push_scope (self)

  Enter a new lexical scope.
- None _pop_scope (self)

  Exit the current lexical scope.
- None _declare_local (self, str name, Type typ, Node node)

  Declare a local variable in the innermost scope.
- Optional[Type] _lookup_local (self, str name)

  Look up a local variable's type in the scope stack.
- Optional[int] _lookup_local_scope_index (self, str name)

  Return scope index where local resolves (nearest scope wins).
- bool _is_guarded_cleanup_header_ref (self, str name, int scope_index)

  True when a local resolves to a guarded maybe-uninitialized header let.
- bool _expr_contains_try (self, Expr expr)

  Check if an expression contains a '?' try operator.
- bool _stmt_contains_try (self, Stmt stmt)

  Check if a statement contains a '?' try operator.
- Optional[bool] _lookup_alive (self, str name)

  Check if a variable is currently alive (not dropped).
- None _set_alive (self, str name, bool alive)

  Set the liveness state of a variable.
- List[Dict[str, bool]] _clone_alive_scopes (self)

  Clone the current definite-liveness stack.
- None _meet_alive_scopes (self, *List[Dict[str, bool]] states)

  Keep a binding alive only when every incoming state keeps it alive.
- List[Dict[str, bool]] _meet_alive_state (self, *List[Dict[str, bool]] states)

  Return the definite-liveness meet of several states.
- StmtFlow _check_dead_liveness_block (self, Block block, *bool check_return_paths)

  Type-check an unreachable branch without exporting liveness or loop exits.
- List[List[Dict[str, bool]]] _check_update_from_backedge_states (self, Optional[Stmt] update, List[List[Dict[str, bool]]] states)

  Check a for-update statement from all reachable body backedges.
- Tuple[LoopFlowCapture, List[List[Dict[str, bool]]], StmtFlow] _check_loop_iteration (self, Block body, *Optional[Stmt] update=None, bool check_return_paths=False)

  Check one loop iteration and return captured flow states.
- Tuple[List[Dict[str, bool]], LoopFlowCapture, List[List[Dict[str, bool]]]] _loop_liveness_fixed_point (self, List[Dict[str, bool]] pre_loop, Block body, *Optional[Expr] cond=None, Optional[Stmt] update=None)

  Converge loop-head liveness and diagnose later-iteration uses.
- StmtFlow _check_block (self, Block block, *bool check_return_paths=False, bool push_new_scope=True)

  Check a block of statements.

- StmtFlow _check_stmt (self, Stmt stmt, *bool check_return_paths=False)

  Check a single statement.
- Optional[Type] _infer_type_expr (self, TypeExpr expr)

  TypeExpr is only valid as an argument to type-accepting intrinsics.
- None _check_expr_liveness (self, Expr expr)

  Replay only local-variable use checks for an already typed expression.
- Optional[Type] _infer_expr (self, Optional[Expr] expr, *Optional[Type] widening_type=None, str context_↩ code="TYP-0319", str context_descriptor="expression")

  Infer the type of an expression.
- None _store_var_resolution (self, VarRef expr, VarRefResolution resolution)

  Record stable resolution metadata outside liveness-only replay.
- Optional[Type] _infer_var_ref (self, VarRef expr)

  Infer type for a variable reference.
- Optional[Type] _infer_unary (self, UnaryOp expr)

  Infer type for a unary operation.
- Optional[Type] _infer_binary (self, BinaryOp expr)

  Infer type for a binary operation.
- Optional[Type] _binary_expect_both_int (self, BinaryOp expr, Optional[Type] left, Optional[Type] right, Optional[Type] result)

  Check that both operands of a binary op are int-assignable.
- Optional[Type] _binary_expect_both_bool (self, BinaryOp expr, Optional[Type] left, Optional[Type] right, Optional[Type] result)

  Check that both operands of a binary op are bool.
- Optional[Type] _binary_equality (self, BinaryOp expr, Optional[Type] left, Optional[Type] right)

  Infer type for equality/inequality comparison.
- Optional[Type] _try_infer_intrinsic (self, CallExpr expr)

  Handle compiler intrinsics calls.
- Type _infer_sizeof_intrinsic (self, CallExpr expr)

  Handle sizeof(T) or sizeof(expr) intrinsic.
- Optional[Type] _try_resolve_type_name (self, str name, *Optional[Node] node=None, Optional[List[str]] module_path=None)

  Try to resolve an identifier as a type name.
- None _store_sizeof_target (self, CallExpr expr, Type target_ty)

  Store the resolved sizeof target type for codegen.
- Type _infer_ord_intrinsic (self, CallExpr expr)

  Handle ord(enum_value) intrinsic - returns 0-based ordinal of enum variant.
- Optional[Type] _infer_call (self, CallExpr expr)

  Infer type for a function or constructor call.
- Optional[Type] _infer_struct_constructor (self, CallExpr expr, Symbol sym)

  Infer type for a struct constructor call.
- Optional[Type] _infer_variant_constructor (self, CallExpr expr, Symbol sym)

  Infer type for an enum variant constructor call.
- Optional[Type] _infer_index (self, IndexExpr expr)

  Infer type for an indexing expression.
- Optional[Type] _infer_field_access (self, FieldAccessExpr expr)

  Infer type for a field access expression.
- Optional[Type] _infer_cast (self, CastExpr expr)

  Infer type for a cast expression and validate compatibility.
- Optional[Type] _infer_try (self, TryExpr expr)

  Infer type for a '?' try operator expression.
- None _check_return (self, ReturnStmt stmt)

Check return statement validity and type compatibility.

- Optional[Type] _resolve_type_ref (self, tref)

  Resolve a TypeRef to a semantic Type.
- bool _can_assign (self, Type target, Type source, *allow_promotion=False)

  Check if 'source' type can be assigned to 'target' type.
- bool _is_nullable_or_ptr (self, Type t)

  Check if type is nullable or a pointer.
- None _error (self, Optional[Node] node, str message)

  Report an error diagnostic.
- None _warn (self, Optional[Node] node, str message)

  Report a warning diagnostic.
- None _diagnostic (self, Optional[Node] node, str message, str kind="info")

  Internal helper to create and append a diagnostic.
- bool _is_int_assignable (self, Optional[Type] typ)

  Check if type is 'int' or 'byte'.
- Optional[int] _get_const_int_for_explicit_cast (self, Expr expr)

  Extract a compile-time integer value from a cast operand when available.
- bool _is_const_null_for_explicit_cast (self, Expr expr)

  Check whether a cast operand is provably null at compile time.
- bool _is_bool (self, Optional[Type] typ)

  Check if type is 'bool'.
- bool _is_string (self, NullType|Type typ)

  Check if type is 'string'.
- Optional[tuple[Type, object]] _case_literal_info (self, Expr expr)

  Get type and value for a case arm literal.
- Optional[bytes] _decode_escaped_bytes (self, str text, Node node)

  Decode a string literal, reporting errors.
- bool _is_void (self, Type typ)

  Check if type is 'void'.
- bool _types_equal (self, Type a, Type b)

  Check if two types are exactly the same.
- str _describe_lvalue (self, Expr expr)

  Generate a human-readable description of an lvalue expression.
- bool _reject_name_qualifier (self, Node node, str name, Optional[List[str]] name_qualifier, Optional[List[str]] module_path)

  Check for and reject unsupported qualified name syntax (::).
- Optional[str] _validate_match_variant_pattern (self, VariantPattern pattern, EnumType scrutinee_ty, Optional[EnumInfo] enum_info)

  Validate one match variant and return its canonical enum name.
- Optional[Type] _infer_new (self, NewExpr expr)

  Infer type for a 'new' heap allocation expression.

Protected Attributes

- _current_func_env
- _current_func_type
- _local_scopes
- _alive_scopes
- _return_paths
- _breakable_loop_depth
- _next_stmt_unreachable

- _cleanup_header_ref_guard_stack
- _loop_flow_capture_stack
- _suppress_liveness_diagnostics
- _suppress_diagnostics
- _liveness_diagnostics_only

7.66.1 Detailed Description

Expression-level type checker for L0.

This checker implements:

- Expression typing: literals, variables, operators, calls, constructors, casts, dereference, indexing, field access, try-operator, sizeof.
- Structs and enums: construction, field access, variant patterns, ord.
- Function calls: argument validation and signature matching.
- Type compatibility: int, bool, string, pointers, nullability.
- Statement checking: let (with inference), return, if, while, match, drop.
- Control flow: return path analysis, unreachable code detection.
- Memory safety: flow-sensitive liveness for dropped variables.

Attributes

- `analysis`: The AnalysisResult to populate with types and diagnostics.

7.66.2 Member Function Documentation

7.66.2.1 __post_init__()

```
None l0_expr_types.ExpressionTypeChecker.__post_init__ (
              self )
```

Initialize checker with analysis data and cache builtin types.

7.66.2.2 _binary_equality()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._binary_equality (
             self,
            BinaryOp expr,
            Optional[Type] left,
            Optional[Type]  right )  [protected]
```

Infer type for equality/inequality comparison.

7.66.2.3 _binary_expect_both_bool()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._binary_expect_both_bool (
              self,
             BinaryOp expr,
             Optional[Type] left,
             Optional[Type] right,
             Optional[Type]  result )  [protected]
```

Check that both operands of a binary op are bool.

7.66.2.4 _binary_expect_both_int()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._binary_expect_both_int (
              self,
             BinaryOp expr,
             Optional[Type] left,
             Optional[Type] right,
             Optional[Type]  result )  [protected]
```

Check that both operands of a binary op are int-assignable.

7.66.2.5 _can_assign()

```
bool l0_expr_types.ExpressionTypeChecker._can_assign (
              self,
             Type target,
             Type source,
             * allow_promotion = False )  [protected]
```

Check if 'source' type can be assigned to 'target' type.

7.66.2.6 _case_literal_info()

```
Optional[tuple[Type, object]] l0_expr_types.ExpressionTypeChecker._case_literal_info (
              self,
             Expr expr )  [protected]
```

Get type and value for a case arm literal.

7.66.2.7 _check_block()

```
StmtFlow l0_expr_types.ExpressionTypeChecker._check_block (
              self,
             Block block,
             *bool  check_return_paths = False,
             bool  push_new_scope = True )  [protected]
```

Check a block of statements.

Parameters

block	The Block node.
check_return_paths	Whether to track return paths in this block.
push_new_scope	Whether to push a new lexical scope for this block.

7.66.2.8 _check_dead_liveness_block()

```
StmtFlow l0_expr_types.ExpressionTypeChecker._check_dead_liveness_block (
              self,
             Block block,
             *bool check_return_paths )  [protected]
```

Type-check an unreachable branch without exporting liveness or loop exits.

7.66.2.9 _check_expr_liveness()

```
None l0_expr_types.ExpressionTypeChecker._check_expr_liveness (
              self,
             Expr expr )  [protected]
```

Replay only local-variable use checks for an already typed expression.

7.66.2.10 _check_function()

```
None l0_expr_types.ExpressionTypeChecker._check_function (
              self,
             FunctionEnv func_env,
             FuncType func_type )  [protected]
```

Check a single function definition.

Parameters

func_env	The function's environment containing AST and module info.
func_type	The function's resolved signature.

7.66.2.11 _check_loop_iteration()

```
Tuple[LoopFlowCapture, List[List[Dict[str, bool]]], StmtFlow] l0_expr_types.ExpressionType↩
Checker._check_loop_iteration (
              self,
             Block body,
             *Optional[Stmt]  update = None,
             bool  check_return_paths = False )  [protected]
```

Check one loop iteration and return captured flow states.

7.66.2.12 _check_return()

```
None l0_expr_types.ExpressionTypeChecker._check_return (
              self,
             ReturnStmt stmt )  [protected]
```

Check return statement validity and type compatibility.

7.66.2.13 _check_stmt()

```
StmtFlow l0_expr_types.ExpressionTypeChecker._check_stmt (
              self,
             Stmt stmt,
             *bool  check_return_paths = False )  [protected]
```

Check a single statement.

Parameters

stmt	The Stmt node.
check_return_paths	Whether to update return path tracking.

7.66.2.14 _check_update_from_backedge_states()

```
List[List[Dict[str, bool]]] l0_expr_types.ExpressionTypeChecker._check_update_from_backedge_↩
states (
              self,
             Optional[Stmt] update,
             List[List[Dict[str, bool]]] states )  [protected]
```

Check a for-update statement from all reachable body backedges.

7.66.2.15 _clone_alive_scopes()

```
List[Dict[str, bool]] l0_expr_types.ExpressionTypeChecker._clone_alive_scopes (
              self )  [protected]
```

Clone the current definite-liveness stack.

7.66.2.16 _declare_local()

```
None l0_expr_types.ExpressionTypeChecker._declare_local (
              self,
             str name,
             Type typ,
             Node node )  [protected]
```

Declare a local variable in the innermost scope.

Parameters

name	The name of the variable.
typ	The resolved Type of the variable.
node	The AST node where the declaration occurs.

7.66.2.17 _decode_escaped_bytes()

```
Optional[bytes] l0_expr_types.ExpressionTypeChecker._decode_escaped_bytes (
              self,
             str text,
             Node node )  [protected]
```

Decode a string literal, reporting errors.

7.66.2.18 _describe_lvalue()

```
str l0_expr_types.ExpressionTypeChecker._describe_lvalue (
              self,
             Expr expr )  [protected]
```

Generate a human-readable description of an lvalue expression.

7.66.2.19 _diagnostic()

```
None l0_expr_types.ExpressionTypeChecker._diagnostic (
              self,
             Optional[Node] node,
             str message,
             str  kind = "info" )  [protected]
```

Internal helper to create and append a diagnostic.

7.66.2.20 _error()

```
None l0_expr_types.ExpressionTypeChecker._error (
              self,
             Optional[Node] node,
             str message )  [protected]
```

Report an error diagnostic.

7.66.2.21 _expr_contains_try()

```
bool l0_expr_types.ExpressionTypeChecker._expr_contains_try (
              self,
             Expr expr )  [protected]
```

Check if an expression contains a '?' try operator.

7.66.2.22 _get_const_int_for_explicit_cast()

```
Optional[int] l0_expr_types.ExpressionTypeChecker._get_const_int_for_explicit_cast (
              self,
             Expr expr )  [protected]
```

Extract a compile-time integer value from a cast operand when available.

7.66.2.23 _infer_binary()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_binary (
              self,
             BinaryOp expr )  [protected]
```

Infer type for a binary operation.

7.66.2.24 _infer_call()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_call (
              self,
             CallExpr expr )  [protected]
```

Infer type for a function or constructor call.

7.66.2.25 _infer_cast()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_cast (
              self,
             CastExpr expr )  [protected]
```

Infer type for a cast expression and validate compatibility.

7.66.2.26 _infer_expr()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_expr (
              self,
             Optional[Expr] expr,
             *Optional[Type]  widening_type = None,
             str  context_code = "TYP-0319",
             str  context_descriptor = "expression" )  [protected]
```

Infer the type of an expression.

Parameters

expr	The expression to type-check.
widening_type	Optional expected type for context-sensitive checking.
context_code	Diagnostic code for type mismatch.
context_descriptor	Description of context for diagnostic.

Returns

The resolved Type of the expression, or None on error.

7.66.2.27 _infer_field_access()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_field_access (
             self,
            FieldAccessExpr expr )  [protected]
```

Infer type for a field access expression.

7.66.2.28 _infer_index()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_index (
             self,
            IndexExpr expr )  [protected]
```

Infer type for an indexing expression.

7.66.2.29 _infer_new()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_new (
             self,
            NewExpr expr )  [protected]
```

Infer type for a 'new' heap allocation expression.

7.66.2.30 _infer_ord_intrinsic()

```
Type l0_expr_types.ExpressionTypeChecker._infer_ord_intrinsic (
             self,
            CallExpr expr )  [protected]
```

Handle ord(enum_value) intrinsic - returns 0-based ordinal of enum variant.

7.66.2.31 _infer_sizeof_intrinsic()

```
Type l0_expr_types.ExpressionTypeChecker._infer_sizeof_intrinsic (
             self,
            CallExpr expr )  [protected]
```

Handle sizeof(T) or sizeof(expr) intrinsic.

7.66.2.32 _infer_struct_constructor()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_struct_constructor (
             self,
            CallExpr expr,
            Symbol sym )  [protected]
```

Infer type for a struct constructor call.

7.66.2.33 _infer_try()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_try (
             self,
            TryExpr expr )  [protected]
```

Infer type for a '?' try operator expression.

7.66.2.34 _infer_type_expr()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_type_expr (
             self,
            TypeExpr expr )  [protected]
```

TypeExpr is only valid as an argument to type-accepting intrinsics.

7.66.2.35 _infer_unary()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_unary (
             self,
            UnaryOp expr )  [protected]
```

Infer type for a unary operation.

7.66.2.36 _infer_var_ref()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_var_ref (
             self,
            VarRef expr )  [protected]
```

Infer type for a variable reference.

7.66.2.37 _infer_variant_constructor()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._infer_variant_constructor (
             self,
            CallExpr expr,
            Symbol sym )  [protected]
```

Infer type for an enum variant constructor call.

7.66.2.38 _is_bool()

```
bool l0_expr_types.ExpressionTypeChecker._is_bool (
             self,
            Optional[Type] typ )  [protected]
```

Check if type is 'bool'.

7.66.2.39 _is_const_null_for_explicit_cast()

```
bool l0_expr_types.ExpressionTypeChecker._is_const_null_for_explicit_cast (
             self,
            Expr expr )  [protected]
```

Check whether a cast operand is provably null at compile time.

7.66.2.40 _is_guarded_cleanup_header_ref()

```
bool l0_expr_types.ExpressionTypeChecker._is_guarded_cleanup_header_ref (
             self,
            str name,
            int scope_index )  [protected]
```

True when a local resolves to a guarded maybe-uninitialized header let.

7.66.2.41 _is_int_assignable()

```
bool l0_expr_types.ExpressionTypeChecker._is_int_assignable (
             self,
            Optional[Type] typ )  [protected]
```

Check if type is 'int' or 'byte'.

7.66.2.42 _is_nullable_or_ptr()

```
bool l0_expr_types.ExpressionTypeChecker._is_nullable_or_ptr (
             self,
            Type t )  [protected]
```

Check if type is nullable or a pointer.

7.66.2.43 _is_string()

```
bool l0_expr_types.ExpressionTypeChecker._is_string (
             self,
            NullType | Type typ )  [protected]
```

Check if type is 'string'.

7.66.2.44 _is_void()

```
bool l0_expr_types.ExpressionTypeChecker._is_void (
             self,
            Type typ )  [protected]
```

Check if type is 'void'.

7.66.2.45 _lookup_alive()

```
Optional[bool] l0_expr_types.ExpressionTypeChecker._lookup_alive (
             self,
            str name )  [protected]
```

Check if a variable is currently alive (not dropped).

7.66.2.46 _lookup_local()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._lookup_local (
             self,
            str name )  [protected]
```

Look up a local variable's type in the scope stack.

7.66.2.47 _lookup_local_scope_index()

```
Optional[int] l0_expr_types.ExpressionTypeChecker._lookup_local_scope_index (
             self,
            str name )  [protected]
```

Return scope index where local resolves (nearest scope wins).

7.66.2.48 _loop_liveness_fixed_point()

```
Tuple[List[Dict[str, bool]], LoopFlowCapture, List[List[Dict[str, bool]]]] l0_expr_types.↩
ExpressionTypeChecker._loop_liveness_fixed_point (
             self,
            List[Dict[str, bool]] pre_loop,
            Block body,
            *Optional[Expr]  cond = None,
            Optional[Stmt]  update = None )  [protected]
```

Converge loop-head liveness and diagnose later-iteration uses.

7.66.2.49 _make_param_alive_scope()

```
Dict[str, bool] l0_expr_types.ExpressionTypeChecker._make_param_alive_scope (
             self,
            FunctionEnv func_env )  [protected]
```

Initialize liveness tracking for function parameters.

7.66.2.50 _make_param_scope()

```
Dict[str, Type] l0_expr_types.ExpressionTypeChecker._make_param_scope (
              self,
             FunctionEnv func_env,
             FuncType func_type )  [protected]
```

Create a name-to-type mapping for function parameters.

7.66.2.51 _meet_alive_scopes()

```
None l0_expr_types.ExpressionTypeChecker._meet_alive_scopes (
              self,
             *List[Dict[str, bool]] states )  [protected]
```

Keep a binding alive only when every incoming state keeps it alive.

7.66.2.52 _meet_alive_state()

```
List[Dict[str, bool]] l0_expr_types.ExpressionTypeChecker._meet_alive_state (
              self,
             *List[Dict[str, bool]] states )  [protected]
```

Return the definite-liveness meet of several states.

7.66.2.53 _pop_scope()

```
None l0_expr_types.ExpressionTypeChecker._pop_scope (
              self )  [protected]
```

Exit the current lexical scope.

7.66.2.54 _push_scope()

```
None l0_expr_types.ExpressionTypeChecker._push_scope (
              self )  [protected]
```

Enter a new lexical scope.

7.66.2.55 _reject_name_qualifier()

```
bool l0_expr_types.ExpressionTypeChecker._reject_name_qualifier (
              self,
             Node node,
             str name,
             Optional[List[str]] name_qualifier,
             Optional[List[str]] module_path )  [protected]
```

Check for and reject unsupported qualified name syntax (::).

7.66.2.56 _resolve_type_ref()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._resolve_type_ref (
              self,
              tref )  [protected]
```

Resolve a TypeRef to a semantic Type.

7.66.2.57 _set_alive()

```
None l0_expr_types.ExpressionTypeChecker._set_alive (
              self,
             str name,
             bool alive )  [protected]
```

Set the liveness state of a variable.

7.66.2.58 _stmt_contains_try()

```
bool l0_expr_types.ExpressionTypeChecker._stmt_contains_try (
              self,
             Stmt stmt )  [protected]
```

Check if a statement contains a '?' try operator.

7.66.2.59 _store_sizeof_target()

```
None l0_expr_types.ExpressionTypeChecker._store_sizeof_target (
              self,
             CallExpr expr,
             Type target_ty )  [protected]
```

Store the resolved sizeof target type for codegen.

7.66.2.60 _store_var_resolution()

```
None l0_expr_types.ExpressionTypeChecker._store_var_resolution (
              self,
             VarRef expr,
             VarRefResolution resolution )  [protected]
```

Record stable resolution metadata outside liveness-only replay.

7.66.2.61 _try_infer_intrinsic()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._try_infer_intrinsic (
              self,
             CallExpr expr )  [protected]
```

Handle compiler intrinsics calls.

7.66.2.62 _try_resolve_type_name()

```
Optional[Type] l0_expr_types.ExpressionTypeChecker._try_resolve_type_name (
              self,
             str name,
             *Optional[Node]  node = None,
             Optional[List[str]]  module_path = None )  [protected]
```

Try to resolve an identifier as a type name.

7.66.2.63 _types_equal()

```
bool l0_expr_types.ExpressionTypeChecker._types_equal (
              self,
             Type a,
             Type b )  [protected]
```

Check if two types are exactly the same.

7.66.2.64 _validate_match_variant_pattern()

```
Optional[str] l0_expr_types.ExpressionTypeChecker._validate_match_variant_pattern (
              self,
             VariantPattern pattern,
             EnumType scrutinee_ty,
             Optional[EnumInfo] enum_info )  [protected]
```

Validate one match variant and return its canonical enum name.

7.66.2.65 _warn()

```
None l0_expr_types.ExpressionTypeChecker._warn (
              self,
             Optional[Node] node,
             str message )  [protected]
```

Report a warning diagnostic.

7.66.2.66 check()

```
None l0_expr_types.ExpressionTypeChecker.check (
              self )
```

Run expression/type checking for all non-extern functions.

This is the main entry point for expression analysis. It should be called after top-level signature resolution is complete and local scopes have been built.

7.66.3 Member Data Documentation

7.66.3.1 _alive_scopes

l0_expr_types.ExpressionTypeChecker._alive_scopes [protected]

7.66.3.2 _breakable_loop_depth

l0_expr_types.ExpressionTypeChecker._breakable_loop_depth [protected]

7.66.3.3 _cleanup_header_ref_guard_stack

l0_expr_types.ExpressionTypeChecker._cleanup_header_ref_guard_stack [protected]

7.66.3.4 _current_func_env

l0_expr_types.ExpressionTypeChecker._current_func_env [protected]

7.66.3.5 _current_func_type

l0_expr_types.ExpressionTypeChecker._current_func_type [protected]

7.66.3.6 _liveness_diagnostics_only

l0_expr_types.ExpressionTypeChecker._liveness_diagnostics_only [protected]

7.66.3.7 _local_scopes

l0_expr_types.ExpressionTypeChecker._local_scopes [protected]

7.66.3.8 _loop_flow_capture_stack

l0_expr_types.ExpressionTypeChecker._loop_flow_capture_stack [protected]

7.66.3.9 _next_stmt_unreachable

l0_expr_types.ExpressionTypeChecker._next_stmt_unreachable [protected]

7.66.3.10 _return_paths

l0_expr_types.ExpressionTypeChecker._return_paths [protected]

7.66.3.11 _suppress_diagnostics

l0_expr_types.ExpressionTypeChecker._suppress_diagnostics [protected]

7.66.3.12 _suppress_liveness_diagnostics

l0_expr_types.ExpressionTypeChecker._suppress_liveness_diagnostics [protected]

7.66.3.13 analysis

AnalysisResult l0_expr_types.ExpressionTypeChecker.analysis [static]

7.66.3.14 bool_type

l0_expr_types.ExpressionTypeChecker.bool_type

7.66.3.15 byte_type

l0_expr_types.ExpressionTypeChecker.byte_type

7.66.3.16 diagnostics

l0_expr_types.ExpressionTypeChecker.diagnostics

7.66.3.17 enum_infos

l0_expr_types.ExpressionTypeChecker.enum_infos

7.66.3.18 expr_types

l0_expr_types.ExpressionTypeChecker.expr_types

7.66.3.19 func_envs

l0_expr_types.ExpressionTypeChecker.func_envs

7.66.3.20 func_types

l0_expr_types.ExpressionTypeChecker.func_types

7.66.3.21 int_type

```
l0_expr_types.ExpressionTypeChecker.int_type
```

7.66.3.22 module_envs

```
l0_expr_types.ExpressionTypeChecker.module_envs
```

7.66.3.23 string_type

```
l0_expr_types.ExpressionTypeChecker.string_type
```

7.66.3.24 struct_infos

```
l0_expr_types.ExpressionTypeChecker.struct_infos
```

7.66.3.25 void_type

```
l0_expr_types.ExpressionTypeChecker.void_type
```

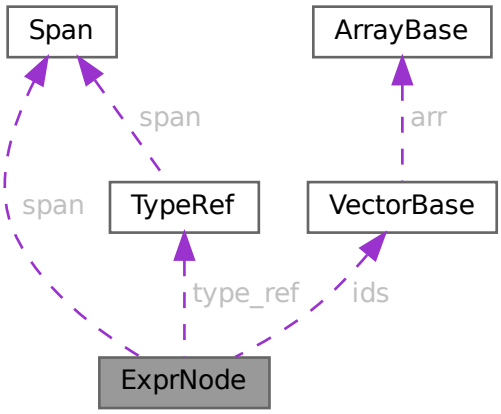
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_expr_types.py

7.67 ExprNode Struct Reference

Represents the ExprNode structure.

Collaboration diagram for ExprNode:



Public Attributes

- kind: ExprKind
- span: Span
- int_value: int
- bool_value: bool
- text_value: string
- expr_a: ExprId
- expr_b: ExprId
- expr_c: ExprId
- ids: VectorBase*?
- type_ref: TypeRef*?
- module_path: StringVector*?
- name_qualifier: StringVector*?

7.67.1 Detailed Description

Represents the ExprNode structure.

7.67.2 Member Data Documentation

7.67.2.1 bool_value

bool_value: `bool_value: bool`

7.67.2.2 expr_a

expr_a: `expr_a: ExprId`

7.67.2.3 expr_b

expr_b: `expr_b: ExprId`

7.67.2.4 expr_c

expr_c: `expr_c: ExprId`

7.67.2.5 ids

ids: `ids: VectorBase*?`

7.67.2.6 int_value

int_value: `int_value: int`

7.67.2.7 kind

kind: `kind: ExprKind`

7.67.2.8 module_path

module_path: `module_path: StringVector*?`

7.67.2.9 name_qualifier

name_qualifier: `name_qualifier: StringVector*?`

7.67.2.10 span

span: `span: Span`

7.67.2.11 text_value

text_value: `text_value: string`

7.67.2.12 type_ref

type_ref: `type_ref: TypeRef*?`

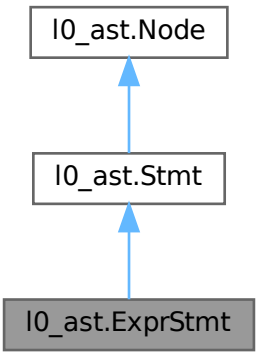
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

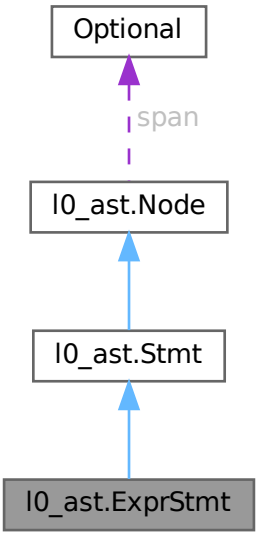
7.68 l0_ast.ExprStmt Class Reference

A statement consisting of a single expression.

Inheritance diagram for l0_ast.ExprStmt:



Collaboration diagram for l0_ast.ExprStmt:



Static Public Attributes

- str expr : "Expr"

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.68.1 Detailed Description

A statement consisting of a single expression.

Attributes

- `expr`: The expression to execute.

7.68.2 Member Data Documentation

7.68.2.1 expr

```
str l0_ast.ExprStmt.expr :  "Expr"  [static]
```

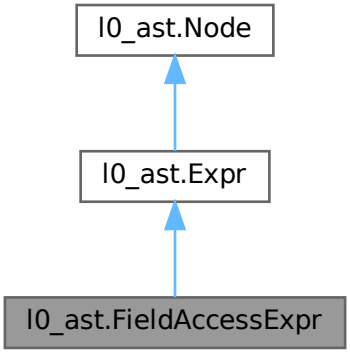
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

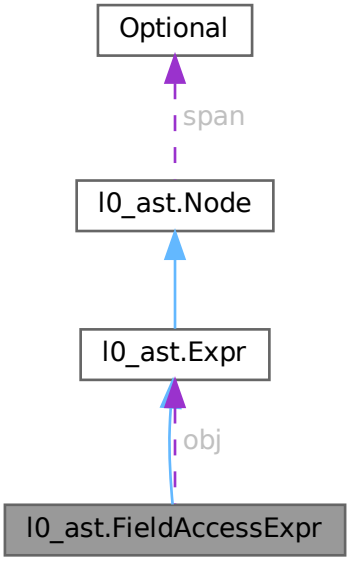
7.69 l0_ast.FieldAccessExpr Class Reference

A field access expression.

Inheritance diagram for l0_ast.FieldAccessExpr:



Collaboration diagram for l0_ast.FieldAccessExpr:



Static Public Attributes

- Expr obj
- str field

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.69.1 Detailed Description

A field access expression.

Attributes

- `obj`: The object expression.
- `field`: The name of the field to access.

7.69.2 Member Data Documentation

7.69.2.1 field

```
str l0_ast.FieldAccessExpr.field  [static]
```

7.69.2.2 obj

```
Expr l0_ast.FieldAccessExpr.obj  [static]
```

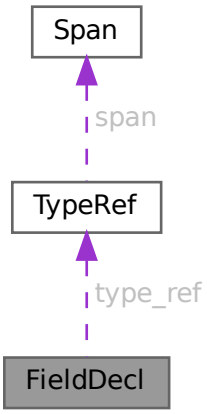
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.70 FieldDecl Struct Reference

Represents the FieldDecl structure.

Collaboration diagram for FieldDecl:



Public Attributes

- name: string
- type_ref: TypeRef*

7.70.1 Detailed Description

Represents the FieldDecl structure.

7.70.2 Member Data Documentation

7.70.2.1 name

name: `name: string`

7.70.2.2 type_ref

type_ref: `type_ref: TypeRef*`

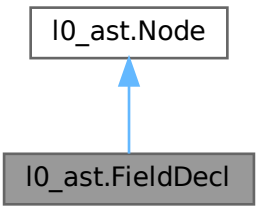
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

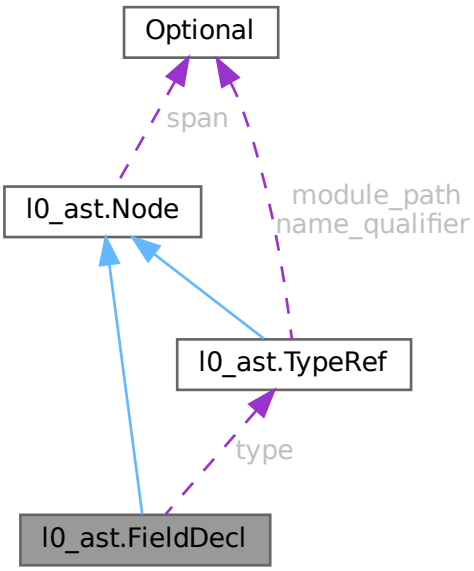
7.71 l0_ast.FieldDecl Class Reference

A field declaration within a struct.

Inheritance diagram for l0_ast.FieldDecl:



Collaboration diagram for l0_ast.FieldDecl:



Static Public Attributes

- str name
- TypeRef type

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.71.1 Detailed Description

A field declaration within a struct.

Attributes

- `name`: The name of the field.
- `type`: The type of the field.

7.71.2 Member Data Documentation

7.71.2.1 name

```
str l0_ast.FieldDecl.name  [static]
```

7.71.2.2 type

`TypeRef l0_ast.FieldDecl.type [static]`

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.72 FileInfo Struct Reference

Shared file metadata wrapper around `sys.rt:: RtFileInfo`.

Public Attributes

- exists: bool
- is_file: bool
- is_dir: bool
- size: int?
- mtime_sec: int?
- mtime_nsec: int?

7.72.1 Detailed Description

Shared file metadata wrapper around `sys.rt:: RtFileInfo`.

7.72.2 Member Data Documentation

7.72.2.1 exists

exists: `exists: bool`

7.72.2.2 is_dir

is_dir: `is_dir: bool`

7.72.2.3 is_file

is_file: `is_file: bool`

7.72.2.4 mtime_nsec

mtime_nsec: `mtime_nsec: int?`

7.72.2.5 mtime_sec

mtime_sec: `mtime_sec: int?`

7.72.2.6 size

size: `size: int?`

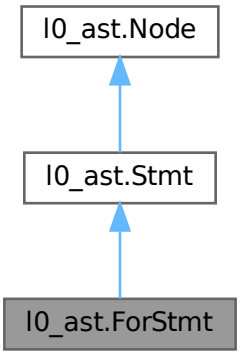
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/fs.l0

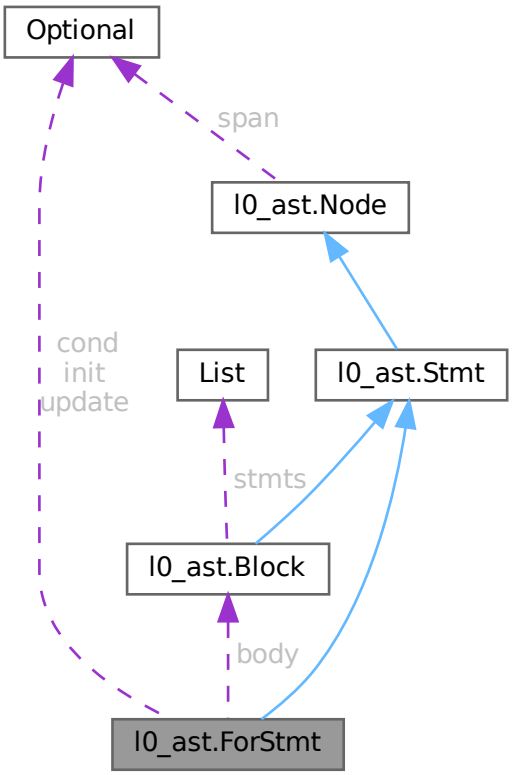
7.73 l0_ast.ForStmt Class Reference

A for loop statement.

Inheritance diagram for l0_ast.ForStmt:



Collaboration diagram for l0_ast.ForStmt:



Static Public Attributes

- Optional init [Stmt]
- Optional cond ["Expr"]
- Optional update [Stmt]
- Block body

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.73.1 Detailed Description

A for loop statement.

Attributes

- `init`: Optional initialization statement.
- `cond`: Optional condition expression.
- `update`: Optional update statement.
- `body`: The loop body block.

7.73.2 Member Data Documentation

7.73.2.1 body

```
Block l0_ast.ForStmt.body  [static]
```

7.73.2.2 cond

```
Optional l0_ast.ForStmt.cond ["Expr"]  [static]
```

7.73.2.3 init

```
Optional l0_ast.ForStmt.init [Stmt]  [static]
```

7.73.2.4 update

```
Optional l0_ast.ForStmt.update [Stmt]  [static]
```

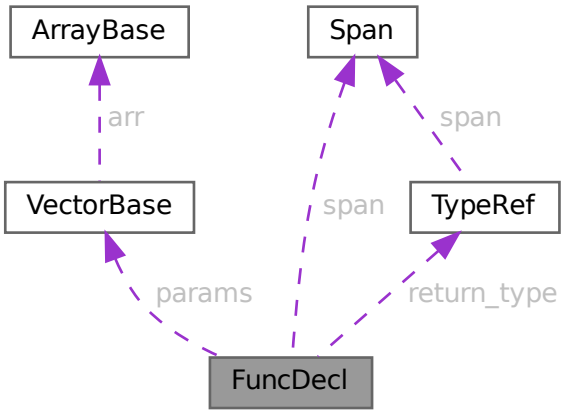
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.74 FuncDecl Struct Reference

Represents the FuncDecl structure.

Collaboration diagram for FuncDecl:



Public Attributes

- name: string
- params: VectorBase*
- return_type: TypeRef*
- body: StmtId
- is_extern: bool
- span: Span

7.74.1 Detailed Description

Represents the FuncDecl structure.

7.74.2 Member Data Documentation

7.74.2.1 body

body: `body: StmtId`

7.74.2.2 is_extern

is_extern: `is_extern: bool`

7.74.2.3 name

name: `name: string`

7.74.2.4 params

params: `params: VectorBase*`

7.74.2.5 return_type

return_type: `return_type: TypeRef*`

7.74.2.6 span

span: `span: Span`

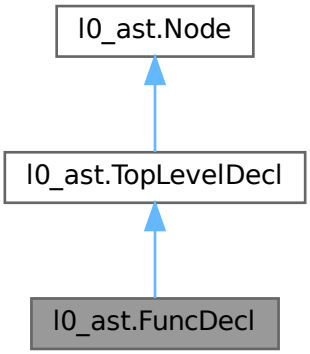
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

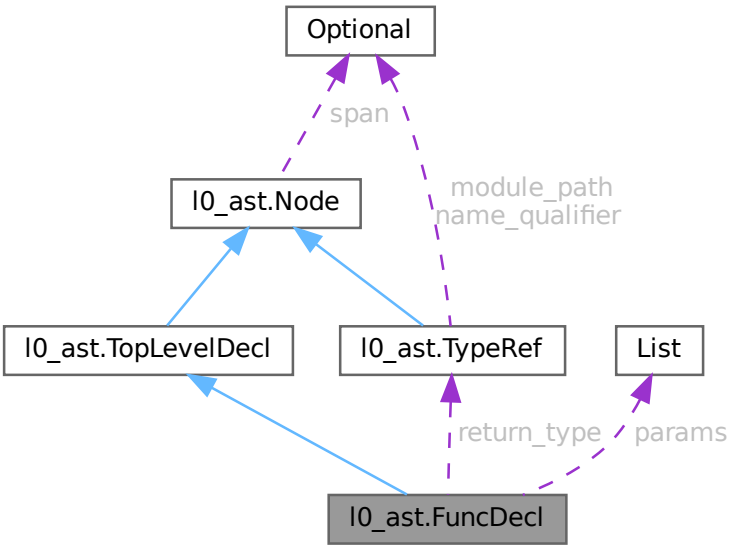
7.75 l0_ast.FuncDecl Class Reference

A function declaration.

Inheritance diagram for l0_ast.FuncDecl:



Collaboration diagram for l0_ast.FuncDecl:



Static Public Attributes

- str name

- List params [Param]
- TypeRef return_type
- str body : "Block"
- bool is_extern = False

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.75.1 Detailed Description

A function declaration.

Attributes

- `name`: The name of the function.
- `params`: List of function parameters.
- `return_type`: The return type of the function.
- `body`: The function body block.
- `is_extern`: Whether the function is declared as 'extern'.

7.75.2 Member Data Documentation

7.75.2.1 body

```
str l0_ast.FuncDecl.body :  "Block"  [static]
```

7.75.2.2 is_extern

```
bool l0_ast.FuncDecl.is_extern = False  [static]
```

7.75.2.3 name

```
str l0_ast.FuncDecl.name  [static]
```

7.75.2.4 params

```
List l0_ast.FuncDecl.params [Param]  [static]
```

7.75.2.5 return_type

```
TypeRef l0_ast.FuncDecl.return_type  [static]
```

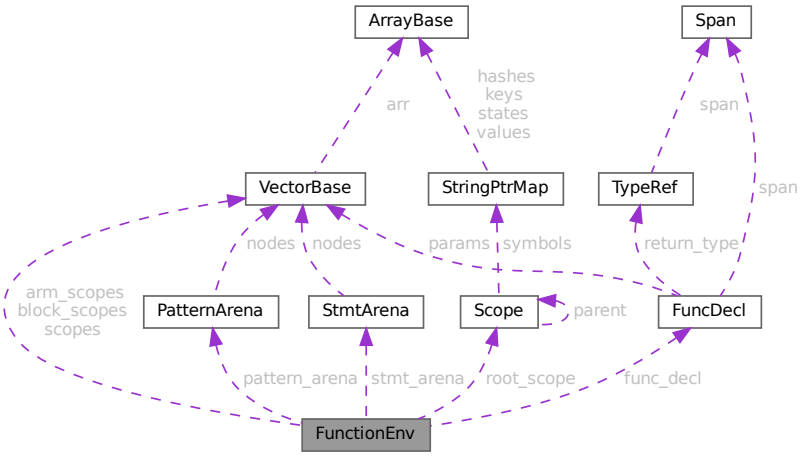
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.76 FunctionEnv Struct Reference

Represents the FunctionEnv structure.

Collaboration diagram for FunctionEnv:



Public Attributes

- key: string
- module_name: string
- func_name: string
- func_decl: FuncDecl*
- pattern_arena: PatternArena*
- stmt_arena: StmtArena*
- root_scope: Scope*
- scopes: VectorBase*
- block_scopes: VectorBase*
- arm_scopes: VectorBase*

7.76.1 Detailed Description

Represents the FunctionEnv structure.

7.76.2 Member Data Documentation

7.76.2.1 arm_scopes

arm_scopes: `arm_scopes: VectorBase*`

7.76.2.2 block_scopes

block_scopes: `block_scopes: VectorBase*`

7.76.2.3 func_decl

func_decl: `func_decl: FuncDecl*`

7.76.2.4 func_name

func_name: `func_name: string`

7.76.2.5 key

key: `key: string`

7.76.2.6 module_name

module_name: `module_name: string`

7.76.2.7 pattern_arena

pattern_arena: `pattern_arena: PatternArena*`

7.76.2.8 root_scope

root_scope: `root_scope: Scope*`

7.76.2.9 scopes

scopes: `scopes: VectorBase*`

7.76.2.10 stmt_arena

stmt_arena: `stmt_arena: StmtArena*`

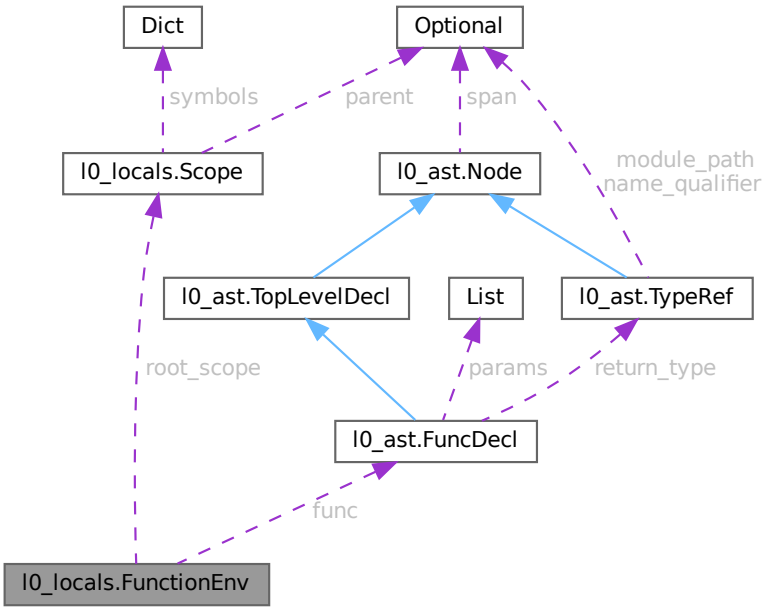
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/locals.l0

7.77 l0_locals.FunctionEnv Class Reference

Environment for a single function.

Collaboration diagram for l0_locals.FunctionEnv:



Static Public Attributes

- str module_name
- FuncDecl func
- Scope root_scope

7.77.1 Detailed Description

Environment for a single function.

Attributes

- `module_name`: Name of the module containing the function.
- `func`: The FuncDecl AST node.
- `root_scope`: The function's root lexical scope.

7.77.2 Member Data Documentation

7.77.2.1 func

```
FuncDecl l0_locals.FunctionEnv.func  [static]
```

7.77.2.2 module_name

```
str l0_locals.FunctionEnv.module_name  [static]
```

7.77.2.3 root_scope

```
Scope l0_locals.FunctionEnv.root_scope  [static]
```

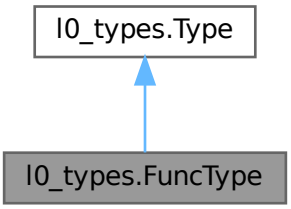
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_locals.py

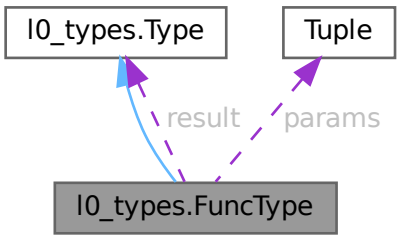
7.78 l0_types.FuncType Class Reference

A function signature type.

Inheritance diagram for l0_types.FuncType:



Collaboration diagram for l0_types.FuncType:



Static Public Attributes

- Tuple params [Type, ...]
- Type result

7.78.1 Detailed Description

A function signature type.

Attributes

- `params`: Tuple of parameter Types.
- `result`: The return Type of the function.

7.78.2 Member Data Documentation

7.78.2.1 params

```
Tuple l0_types.FuncType.params [Type, ...]  [static]
```

7.78.2.2 result

```
Type l0_types.FuncType.result  [static]
```

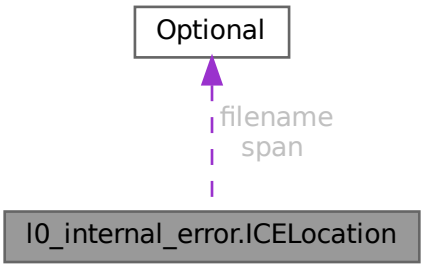
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_types.py

7.79 l0_internal_error.ICELocation Class Reference

Represents the location where an internal compiler error occurred.

Collaboration diagram for l0_internal_error.ICELocation:



Static Public Attributes

- Optional filename [str]
- Optional span [Span]

7.79.1 Detailed Description

Represents the location where an internal compiler error occurred.

Attributes

- `filename`: The path to the source file being processed.
- `span`: Optional source span information.

7.79.2 Member Data Documentation

7.79.2.1 filename

```
Optional l0_internal_error.ICELocation.filename [str]  [static]
```

7.79.2.2 span

```
Optional l0_internal_error.ICELocation.span [Span]  [static]
```

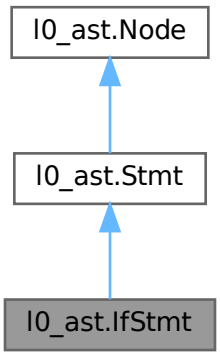
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_internal_error.py

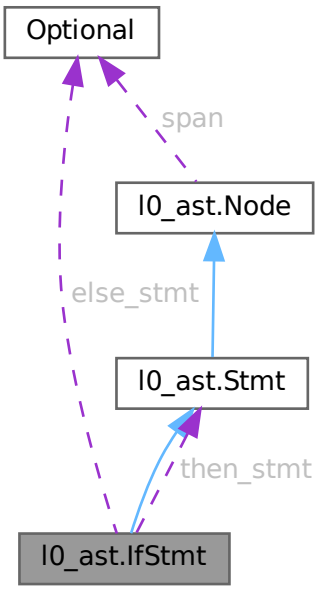
7.80 l0_ast.IfStmt Class Reference

An if-else statement.

Inheritance diagram for l0_ast.IfStmt:



Collaboration diagram for l0_ast.IfStmt:



Static Public Attributes

- str cond : "Expr"
- Stmt then_stmt
- Optional else_stmt [Stmt]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.80.1 Detailed Description

An if-else statement.

Attributes

- `cond`: The condition expression.
- `then_stmt`: The statement to execute if the condition is true.
- `else_stmt`: Optional statement to execute if the condition is false.

7.80.2 Member Data Documentation

7.80.2.1 cond

```
str l0_ast.IfStmt.cond :  "Expr"  [static]
```

7.80.2.2 else_stmt

```
Optional l0_ast.IfStmt.else_stmt [Stmt]  [static]
```

7.80.2.3 then_stmt

```
Stmt l0_ast.IfStmt.then_stmt  [static]
```

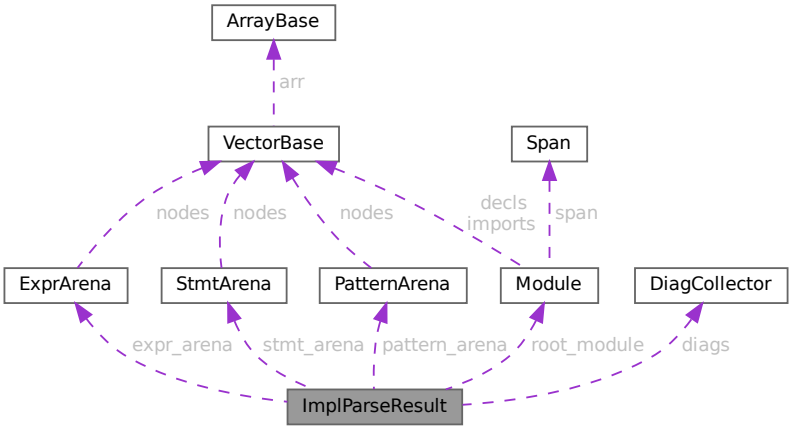
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.81 ImplParseResult Struct Reference

Represents the ImplParseResult structure.

Collaboration diagram for ImplParseResult:



Public Attributes

- root_module: Module*?
- expr_arena: ExprArena*
- stmt_arena: StmtArena*
- pattern_arena: PatternArena*
- diags: DiagCollector*

7.81.1 Detailed Description

Represents the ImplParseResult structure.

7.81.2 Member Data Documentation

7.81.2.1 diags

diags: `diags: DiagCollector*`

7.81.2.2 expr_arena

expr_arena: `expr_arena: ExprArena*`

7.81.2.3 pattern_arena

pattern_arena: `pattern_arena: PatternArena*`

7.81.2.4 root_module

root_module: `root_module: Module*?`

7.81.2.5 stmt_arena

stmt_arena: `stmt_arena: StmtArena*`

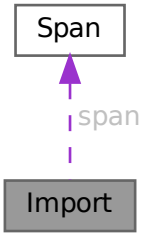
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/parser/shared.l0

7.82 Import Struct Reference

Represents the Import structure.

Collaboration diagram for Import:



Public Attributes

- name: string
- span: Span

7.82.1 Detailed Description

Represents the Import structure.

7.82.2 Member Data Documentation

7.82.2.1 name

name: `name: string`

7.82.2.2 span

span: `span: Span`

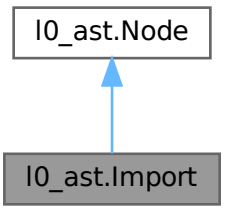
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

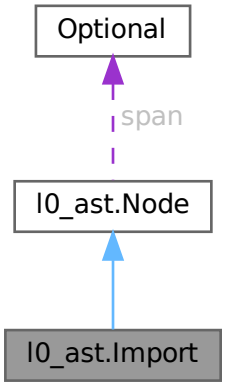
7.83 l0_ast.Import Class Reference

An import declaration.

Inheritance diagram for l0_ast.Import:



Collaboration diagram for l0_ast.Import:



Static Public Attributes

- str name

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.83.1 Detailed Description

An import declaration.

Attributes

- `name`: The qualified name of the module being imported.

7.83.2 Member Data Documentation

7.83.2.1 name

```
str l0_ast.Import.name  [static]
```

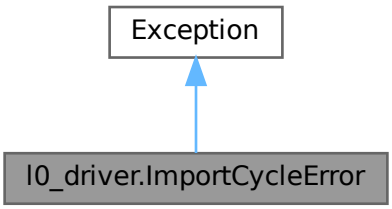
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

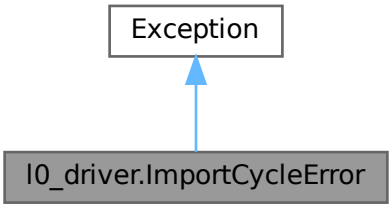
7.84 l0_driver.ImportCycleError Class Reference

Raised when a cyclic import is detected.

Inheritance diagram for l0_driver.ImportCycleError:



Collaboration diagram for l0_driver.ImportCycleError:



7.84.1 Detailed Description

Raised when a cyclic import is detected.

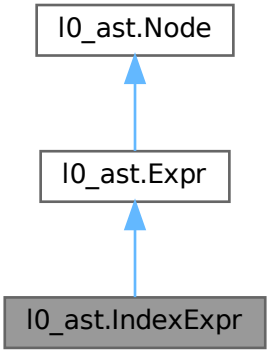
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_driver.py

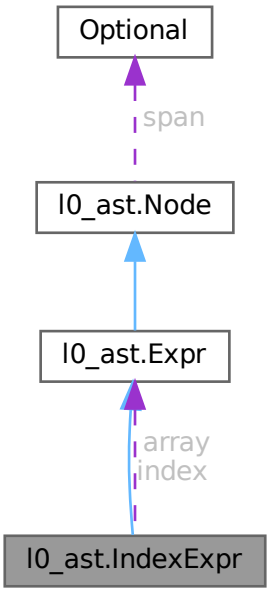
7.85 l0_ast.IndexExpr Class Reference

An array indexing expression.

Inheritance diagram for l0_ast.IndexExpr:



Collaboration diagram for l0_ast.IndexExpr:



Static Public Attributes

- Expr array
- Expr index

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.85.1 Detailed Description

An array indexing expression.

Attributes

- `array`: The array expression.
- `index`: The index expression.

7.85.2 Member Data Documentation

7.85.2.1 array

```
Expr l0_ast.IndexExpr.array  [static]
```

7.85.2.2 index

```
Expr l0_ast.IndexExpr.index  [static]
```

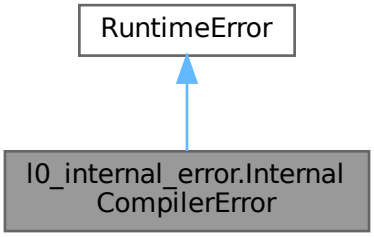
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

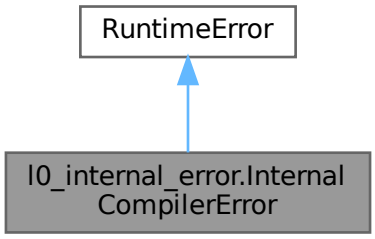
7.86 l0_internal_error.InternalCompilerError Class Reference

Exception raised when an internal compiler invariant is violated.

Inheritance diagram for l0_internal_error.InternalCompilerError:



Collaboration diagram for l0_internal_error.InternalCompilerError:



Public Member Functions

- __init__ (self, str message, ICELocation|None loc=None)
  Initialize the internal compiler error.
- str format (self)
  Format the ICE as a human-readable string.

Public Attributes

- message
- loc

7.86.1 Detailed Description

Exception raised when an internal compiler invariant is violated.

Internal compiler errors (ICEs) indicate bugs in the compiler itself, rather than mistakes in the user's source code.

Attributes

- `message`: A descriptive error message, preferably including an [ICE-nnnn] code.
- `loc`: The source location where the error originated.

7.86.2 Constructor & Destructor Documentation

7.86.2.1 __init__()

```
l0_internal_error.InternalCompilerError.__init__ (
              self,
             str message,
             ICELocation | None  loc = None )
```

Initialize the internal compiler error.

Parameters

message	The error message.
loc	Optional source location.

7.86.3 Member Function Documentation

7.86.3.1 format()

```
str l0_internal_error.InternalCompilerError.format (
              self )
```

Format the ICE as a human-readable string.

Returns

A formatted error message including location information.

7.86.4 Member Data Documentation

7.86.4.1 loc

```
l0_internal_error.InternalCompilerError.loc
```

7.86.4.2 message

```
l0_internal_error.InternalCompilerError.message
```

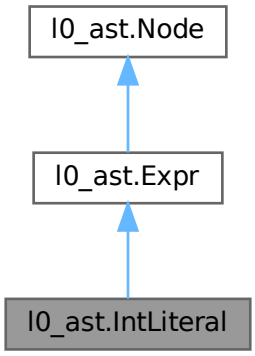
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_internal_error.py

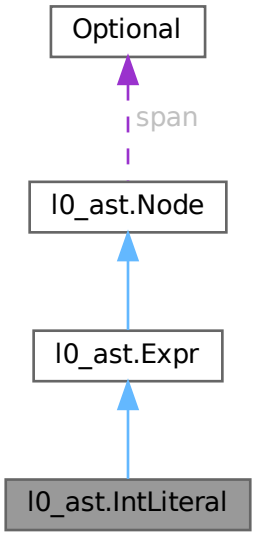
7.87 l0_ast.IntLiteral Class Reference

An integer literal expression.

Inheritance diagram for l0_ast.IntLiteral:



Collaboration diagram for l0_ast.IntLiteral:



Static Public Attributes

- int value

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.87.1 Detailed Description

An integer literal expression.

Attributes

- `value`: The integer value.

7.87.2 Member Data Documentation

7.87.2.1 value

```
int l0_ast.IntLiteral.value  [static]
```

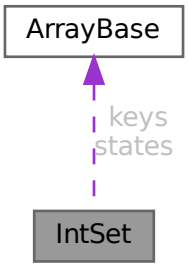
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.88 IntSet Struct Reference

IntSet is an open-addressing hash set of integers with linear probing.

Collaboration diagram for IntSet:



Public Attributes

- capacity: int
- count: int
- tomb_count: int
- states: ArrayBase∗
- keys: ArrayBase∗

7.88.1 Detailed Description

IntSet is an open-addressing hash set of integers with linear probing.

Struct-of-arrays: states, keys.

7.88.2 Member Data Documentation

7.88.2.1 capacity

capacity: `capacity: int`

7.88.2.2 count

count: `count: int`

7.88.2.3 keys

keys: `keys: ArrayBase*`

7.88.2.4 states

states: `states: ArrayBase*`

7.88.2.5 tomb_count

tomb_count: `tomb_count: int`

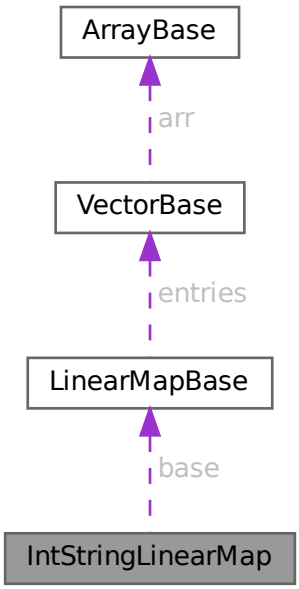
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/util/intset.l0

7.89 IntStringLinearMap Struct Reference

A linear map specialized for integer keys and string values.

Collaboration diagram for IntStringLinearMap:



Public Attributes

- base: LinearMapBase∗

7.89.1 Detailed Description

A linear map specialized for integer keys and string values.

7.89.2 Member Data Documentation

7.89.2.1 base

base: `base: LinearMapBase∗`

The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/linear_map.l0

7.90 IntStringLinearMapEntry Struct Reference

Entry for an integer-to-string linear map.

Public Attributes

- key: int
- value: string

7.90.1 Detailed Description

Entry for an integer-to-string linear map.

7.90.2 Member Data Documentation

7.90.2.1 key

key: `key: int`

7.90.2.2 value

value: `value: string`

The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/linear_map.l0

7.91 l0_opt_bool Struct Reference

Optional boolean wrapper.

```
#include <compiler/shared/runtime/l0_runtime.h>
```

Public Attributes

- l0_bool has_value
- l0_bool value

7.91.1 Detailed Description

Optional boolean wrapper.

7.91.2 Member Data Documentation

7.91.2.1 has_value

```
l0_bool l0_opt_bool::has_value
```

7.91.2.2 value

```
l0_bool l0_opt_bool::value
```

The documentation for this struct was generated from the following file:

- compiler/shared/runtime/l0_runtime.h

7.92 l0_opt_byte Struct Reference

Optional byte wrapper.

```
#include <compiler/shared/runtime/l0_runtime.h>
```

Public Attributes

- l0_bool has_value
- l0_byte value

7.92.1 Detailed Description

Optional byte wrapper.

7.92.2 Member Data Documentation

7.92.2.1 has_value

```
l0_bool l0_opt_byte::has_value
```

7.92.2.2 value

```
l0_byte l0_opt_byte::value
```

The documentation for this struct was generated from the following file:

- compiler/shared/runtime/l0_runtime.h

7.93 l0_opt_int Struct Reference

Optional integer wrapper.

`#include <compiler/shared/runtime/l0_runtime.h>`

Public Attributes

- l0_bool has_value
- l0_int value

7.93.1 Detailed Description

Optional integer wrapper.

7.93.2 Member Data Documentation

7.93.2.1 has_value

```
l0_bool l0_opt_int::has_value
```

7.93.2.2 value

```
l0_int l0_opt_int::value
```

The documentation for this struct was generated from the following file:

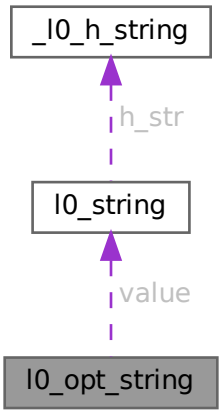
- compiler/shared/runtime/l0_runtime.h

7.94 l0_opt_string Struct Reference

Optional string wrapper.

`#include <compiler/shared/runtime/l0_runtime.h>`

Collaboration diagram for l0_opt_string:



Public Attributes

- l0_bool has_value
- l0_string value

7.94.1 Detailed Description

Optional string wrapper.

7.94.2 Member Data Documentation

7.94.2.1 has_value

```
l0_bool l0_opt_string::has_value
```

7.94.2.2 value

```
l0_string l0_opt_string::value
```

The documentation for this struct was generated from the following file:

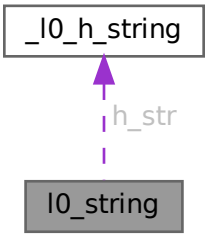
- compiler/shared/runtime/l0_runtime.h

7.95 l0_string Struct Reference

Unified L0 string type (static or heap-allocated).

```
#include <compiler/shared/runtime/l0_runtime.h>
```

Collaboration diagram for l0_string:



Public Attributes

- unsigned int kind: 1
  Kind of string: either L0_STRING_K_STATIC (0) or L0_STRING_K_HEAP (1)
- unsigned int: 0
  Align to next unsigned int boundary.
- union {
  struct {
  l0_int len
  Length in bytes (for constant inline strings)
  const char * bytes
  Pointer to character data (may be NULL for empty string)
  } s_str
  Static string structure for constant inline strings.
  _l0_h_string * h_str
  Heap-allocated string structure for dynamic strings.
  } data

7.95.1 Detailed Description

Unified L0 string type (static or heap-allocated).

7.95.2 Member Data Documentation

7.95.2.1 bytes

```
const char* l0_string::bytes
```

Pointer to character data (may be NULL for empty string)

7.95.2.2 [union]

```
union { ... } l0_string::data
```

7.95.2.3 h_str

```
_l0_h_string* l0_string::h_str
```

Heap-allocated string structure for dynamic strings.

7.95.2.4 int

```
unsigned l0_string::int
```

Align to next unsigned int boundary.

7.95.2.5 kind

```
unsigned int l0_string::kind
```

Kind of string: either L0_STRING_K_STATIC (0) or L0_STRING_K_HEAP (1)

7.95.2.6 len

```
l0_int l0_string::len
```

Length in bytes (for constant inline strings)

7.95.2.7 [struct]

```
struct { ... } l0_string::s_str
```

Static string structure for constant inline strings.

The documentation for this struct was generated from the following file:

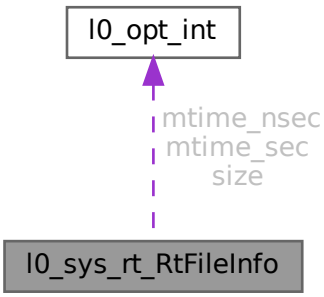
- compiler/shared/runtime/l0_runtime.h

7.96 l0_sys_rt_RtFileInfo Struct Reference

Definition for `sys.rt::RtFileInfo`.

```
#include <compiler/shared/runtime/l0_runtime.h>
```

Collaboration diagram for l0_sys_rt_RtFileInfo:



Public Attributes

- l0_bool exists
- l0_bool is_file
- l0_bool is_dir
- l0_opt_int size
- l0_opt_int mtime_sec
- l0_opt_int mtime_nsec

7.96.1 Detailed Description

Definition for `sys.rt::RtFileInfo`.

7.96.2 Member Data Documentation

7.96.2.1 exists

```
l0_bool l0_sys_rt_RtFileInfo::exists
```

7.96.2.2 is_dir

```
l0_bool l0_sys_rt_RtFileInfo::is_dir
```

7.96.2.3 is_file

```
l0_bool l0_sys_rt_RtFileInfo::is_file
```

7.96.2.4 mtime_nsec

```
l0_opt_int l0_sys_rt_RtFileInfo::mtime_nsec
```

7.96.2.5 mtime_sec

```
l0_opt_int l0_sys_rt_RtFileInfo::mtime_sec
```

7.96.2.6 size

```
l0_opt_int l0_sys_rt_RtFileInfo::size
```

The documentation for this struct was generated from the following file:

- compiler/shared/runtime/l0_runtime.h

7.97 l0_sys_rt_RtTimeParts Struct Reference

Definition for `sys.rt::RtTimeParts`.

`#include <compiler/shared/runtime/l0_runtime.h>`

Public Attributes

- l0_int sec
- l0_int nsec

7.97.1 Detailed Description

Definition for `sys.rt::RtTimeParts`.

7.97.2 Member Data Documentation

7.97.2.1 nsec

`l0_int l0_sys_rt_RtTimeParts::nsec`

7.97.2.2 sec

`l0_int l0_sys_rt_RtTimeParts::sec`

The documentation for this struct was generated from the following file:

- compiler/shared/runtime/l0_runtime.h

7.98 l0_driver.L0Driver Class Reference

Stage-1 driver for the L0 compiler.

Public Member Functions

- __init__ (self, SourceSearchPaths|None search_paths=None, CompilationContext|None context=None)
  Initialize the L0 driver.
- AnalysisResult analyze (self, str entry_module_name)
  Execute the high-level front-end pipeline.
- CompilationUnit build_compilation_unit (self, str entry_module_name)
  Build a compilation unit for an entry module.
- Module load_module (self, str module_name)
  Load a module by its qualified name.

Public Attributes

- search_paths
- context
- diagnostics

Protected Member Functions

- Module _load_single_file (self, str|Path path)

  Load a single file as a parsed module.
- Module _parse_source (self, str text, str file_path)

  Tokenize and parse source text.

7.98.1 Detailed Description

Stage-1 driver for the L0 compiler.

Responsibilities include:

- Reading source files.
- Tokenizing (lexing).
- Parsing.
- Recursively resolving imports using SourceSearchPaths.

Entry points:

- analyze(name): High-level analysis pipeline for an entry module.
- build_compilation_unit(name): Build a closed set of modules for an entry.
- load_module(name): Load by dotted module name, recursively loading imports.
- load_single_file(path): Ad hoc one-off parsing (no import resolution).

7.98.2 Constructor & Destructor Documentation

7.98.2.1 __init__()

```
l0_driver.L0Driver.__init__ (
              self,
             SourceSearchPaths | None  search_paths = None,
             CompilationContext | None  context = None )
```

Initialize the L0 driver.

Parameters

search_paths	Paths to search for modules. Defaults to an empty set of search paths.
context	Compilation context for configuration and logging. Defaults to CompilationContext.default().

7.98.3 Member Function Documentation

7.98.3.1 _load_single_file()

```
Module l0_driver.L0Driver._load_single_file (
            self,
            str | Path path )  [protected]
```

Load a single file as a parsed module.

Ignores search paths and does not recursively resolve imports. The module is cached by its declared name.

Parameters

path	Path to the source file.

Returns

The parsed Module object.

Exceptions

FileNotFoundError	If the file does not exist.
SourceEncodingError	If the file is not valid UTF-8.

7.98.3.2 _parse_source()

```
Module l0_driver.L0Driver._parse_source (
            self,
            str text,
            str file_path )  [protected]
```

Tokenize and parse source text.

Parameters

text	The source code text.
file_path	The path to the file (for diagnostics).

Returns

The parsed Module object.

7.98.3.3 analyze()

```
AnalysisResult l0_driver.L0Driver.analyze (
            self,
            str entry_module_name )
```

Execute the high-level front-end pipeline.

The pipeline consists of the following stages: 1. Build CompilationUnit for entry_module_name. 2. Run Name↩ Resolver (module-level symbols). 3. Run SignatureResolver (top-level types). 4. Run LocalScopeResolver (function/block scopes). 5. Run ExpressionTypeChecker (expression types).

Parameters

entry_module_name	The name of the module to use as the entry point.

Returns

An AnalysisResult containing the compilation unit, environment information, and all collected diagnostics.

7.98.3.4 build_compilation_unit()

```
CompilationUnit l0_driver.L0Driver.build_compilation_unit (
              self,
             str entry_module_name )
```

Build a compilation unit for an entry module.

Loads the entry module and recursively walks its imports to build the transitive closure of all required modules.

Parameters

entry_module_name	The name of the entry module.

Returns

A CompilationUnit containing the entry module and all reachable modules.

7.98.3.5 load_module()

```
Module l0_driver.L0Driver.load_module (
              self,
             str module_name )
```

Load a module by its qualified name.

Uses search paths to resolve the module name to a file, loads and parses it, and recursively loads all imported modules. Results are cached.

Parameters

module_name	The qualified (dotted) name of the module.

Returns

The loaded and parsed Module object.

Exceptions

ImportCycleError	If a cyclic import is detected.
FileNotFoundError	If the module source file cannot be found.
ValueError	If the module name declared in the file mismatch.
SourceEncodingError	If the source file is not valid UTF-8.

7.98.4 Member Data Documentation

7.98.4.1 context

`l0_driver.L0Driver.context`

7.98.4.2 diagnostics

`l0_driver.L0Driver.diagnostics`

7.98.4.3 search_paths

`l0_driver.L0Driver.search_paths`

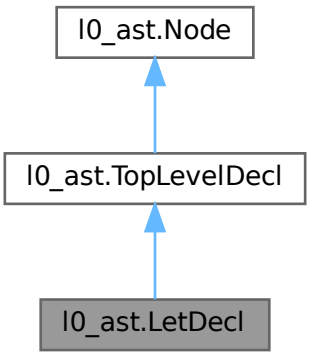
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_driver.py

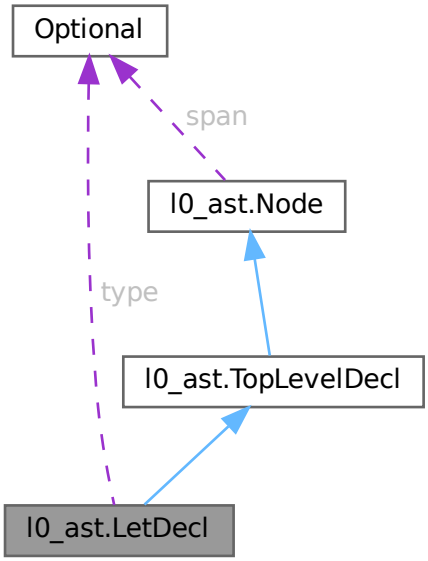
7.99 l0_ast.LetDecl Class Reference

A top-level 'let' declaration.

Inheritance diagram for l0_ast.LetDecl:



Collaboration diagram for l0_ast.LetDecl:



Static Public Attributes

- str name
- Optional type [TypeRef]
- str value : "Expr"

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.99.1 Detailed Description

A top-level 'let' declaration.

Attributes

- `name`: The name of the constant.
- `type`: Optional type annotation.
- `value`: The initialization expression.

7.99.2 Member Data Documentation

7.99.2.1 name

```
str l0_ast.LetDecl.name  [static]
```

7.99.2.2 type

```
Optional l0_ast.LetDecl.type [TypeRef]  [static]
```

7.99.2.3 value

```
str l0_ast.LetDecl.value :  "Expr"  [static]
```

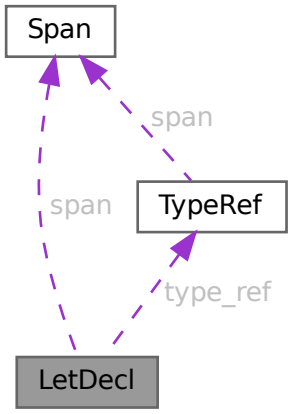
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.100 LetDecl Struct Reference

Represents the LetDecl structure.

Collaboration diagram for LetDecl:



Public Attributes

- name: string
- type_ref: TypeRef*?
- value: ExprId
- span: Span

7.100.1 Detailed Description

Represents the LetDecl structure.

7.100.2 Member Data Documentation

7.100.2.1 name

name: `name: string`

7.100.2.2 span

span: `span: Span`

7.100.2.3 type_ref

type_ref: `type_ref: TypeRef*?`

7.100.2.4 value

value: `value: ExprId`

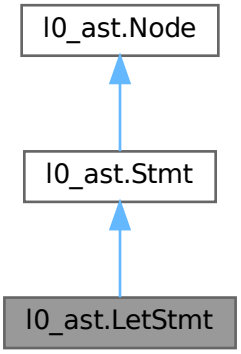
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

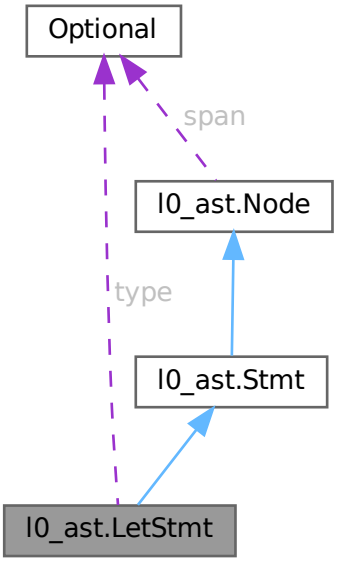
7.101 l0_ast.LetStmt Class Reference

A local variable declaration statement.

Inheritance diagram for l0_ast.LetStmt:



Collaboration diagram for l0_ast.LetStmt:



Static Public Attributes

- str name
- Optional type [TypeRef]
- str value : "Expr"

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.101.1 Detailed Description

A local variable declaration statement.

Attributes

- `name`: The name of the variable.
- `type`: Optional type annotation.
- `value`: The initialization expression.

7.101.2 Member Data Documentation

7.101.2.1 name

```
str l0_ast.LetStmt.name  [static]
```

7.101.2.2 type

```
Optional l0_ast.LetStmt.type [TypeRef]  [static]
```

7.101.2.3 value

```
str l0_ast.LetStmt.value :  "Expr"  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.102 l0_lexer.Lexer Class Reference

Tokenizes L0 source code.

Public Member Functions

- None __init__ (self, str source, str filename="<input>", Optional[List[Diagnostic]] diagnostics=None)

  Initialize the lexer.
- "Lexer" from_source (cls, str source)

  Create a lexer from a source string with default settings.
- List[Token] tokenize (self)

  Perform tokenization of the source string.

Public Attributes

- source
- filename
- length
- index
- line
- column
- diagnostics

Protected Member Functions

- None _error (self, str message, int line, int column)

  Add an error diagnostic.
- Token _lexer_error_token (self, str text, int line, int column, List[Diagnostic] diagnostics, Optional[Token] recovery=None)

  Build a deferred lexer-error wrapper token.
- Token _wrap_new_diagnostics (self, int diagnostic_start, str text, int line, int column, Token recovery)

  Move newly emitted recoverable diagnostics into a wrapper token.
- bool _at_end (self)

  Check if all input has been scanned.
- str _peek (self)

  Peek at the current character without advancing.
- str _peek_next (self)

  Peek at the next character without advancing.
- str _advance (self)

  Advance the current index and return the character.
- Token _next_token (self)

  Scan and return the next token.
- str _read_byte_literal (self, int start_col, int start_line)

  Scan a byte/character literal.
- str _read_string_literal (self)

  Scan a double-quoted string literal.
- str _read_valid_char_escape (self)

  Scan a single escape sequence.
- str _read_number (self, str c, int start_col, int start_line, bool is_negative=False)

  Scan an integer literal.
- None _skip_ws_and_comments (self)

  Skip whitespace and both line and block comments.
- str _collect_invalid_characters (self)

  Consume the rest of an invalid-character run and return its text.

Protected Attributes

- _prev_ends_expression

7.102.1 Detailed Description

Tokenizes L0 source code.

This lexer scans source text and produces a list of Token objects. It tracks line and column information for diagnostics.

Attributes

- `source`: The source code string.
- `filename`: Name of the file being tokenized.
- `diagnostics`: List of collected diagnostics.

7.102.2 Constructor & Destructor Documentation

7.102.2.1 __init__()

```
None l0_lexer.Lexer.__init__ (
              self,
             str source,
             str  filename = "<input>",
             Optional[List[Diagnostic]]  diagnostics = None )
```

Initialize the lexer.

Parameters

source	The source code text.
filename	The source filename for diagnostics.
diagnostics	Optional list to collect diagnostics into.

7.102.3 Member Function Documentation

7.102.3.1 _advance()

```
str l0_lexer.Lexer._advance (
              self )  [protected]
```

Advance the current index and return the character.

7.102.3.2 _at_end()

```
bool l0_lexer.Lexer._at_end (
              self )  [protected]
```

Check if all input has been scanned.

7.102.3.3 _collect_invalid_characters()

```
str l0_lexer.Lexer._collect_invalid_characters (
              self )  [protected]
```

Consume the rest of an invalid-character run and return its text.

The run stops at printable ASCII or whitespace, so a diagnostic span never crosses a line break.

7.102.3.4 _error()

```
None l0_lexer.Lexer._error (
              self,
             str message,
             int line,
             int column )  [protected]
```

Add an error diagnostic.

7.102.3.5 _lexer_error_token()

```
Token l0_lexer.Lexer._lexer_error_token (
             self,
            str text,
            int line,
            int column,
            List[Diagnostic] diagnostics,
            Optional[Token]  recovery = None )  [protected]
```

Build a deferred lexer-error wrapper token.

7.102.3.6 _next_token()

```
Token l0_lexer.Lexer._next_token (
             self )  [protected]
```

Scan and return the next token.

7.102.3.7 _peek()

```
str l0_lexer.Lexer._peek (
             self )  [protected]
```

Peek at the current character without advancing.

7.102.3.8 _peek_next()

```
str l0_lexer.Lexer._peek_next (
             self )  [protected]
```

Peek at the next character without advancing.

7.102.3.9 _read_byte_literal()

```
str l0_lexer.Lexer._read_byte_literal (
             self,
            int start_col,
            int start_line )  [protected]
```

Scan a byte/character literal.

7.102.3.10 _read_number()

```
str l0_lexer.Lexer._read_number (
             self,
            str c,
            int start_col,
            int start_line,
            bool  is_negative = False )  [protected]
```

Scan an integer literal.

7.102.3.11 _read_string_literal()

```
str l0_lexer.Lexer._read_string_literal (
              self )  [protected]
```

Scan a double-quoted string literal.

7.102.3.12 _read_valid_char_escape()

```
str l0_lexer.Lexer._read_valid_char_escape (
              self )  [protected]
```

Scan a single escape sequence.

7.102.3.13 _skip_ws_and_comments()

```
None l0_lexer.Lexer._skip_ws_and_comments (
              self )  [protected]
```

Skip whitespace and both line and block comments.

7.102.3.14 _wrap_new_diagnostics()

```
Token l0_lexer.Lexer._wrap_new_diagnostics (
              self,
             int diagnostic_start,
             str text,
             int line,
             int column,
             Token recovery )  [protected]
```

Move newly emitted recoverable diagnostics into a wrapper token.

7.102.3.15 from_source()

```
"Lexer" l0_lexer.Lexer.from_source (
              cls,
             str source )
```

Create a lexer from a source string with default settings.

Parameters

source	The source code text.

Returns

A new Lexer instance.

7.102.3.16 tokenize()

```
List[Token] l0_lexer.Lexer.tokenize (
              self )
```

Perform tokenization of the source string.

Returns

A list of all tokens, ending with an EOF token.

7.102.4 Member Data Documentation

7.102.4.1 _prev_ends_expression

```
l0_lexer.Lexer._prev_ends_expression  [protected]
```

7.102.4.2 column

```
l0_lexer.Lexer.column
```

7.102.4.3 diagnostics

```
l0_lexer.Lexer.diagnostics
```

7.102.4.4 filename

```
l0_lexer.Lexer.filename
```

7.102.4.5 index

```
l0_lexer.Lexer.index
```

7.102.4.6 length

```
l0_lexer.Lexer.length
```

7.102.4.7 line

```
l0_lexer.Lexer.line
```

7.102.4.8 source

`l0_lexer.Lexer.source`

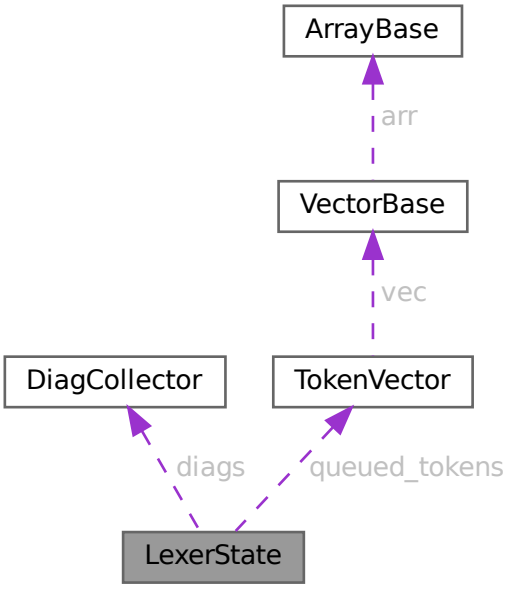
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_lexer.py

7.103 LexerState Struct Reference

Lexer state object that keeps track of the current position in the source code being lexed.

Collaboration diagram for LexerState:



Public Attributes

- source: string
- filename: string
- length: int
- index: int
- line: int
- column: int
- prev_ends_expression: bool
- diags: DiagCollector*
- pending_errors: DiagnosticVector*
- queued_tokens: TokenVector
- queued_index: int

7.103.1 Detailed Description

Lexer state object that keeps track of the current position in the source code being lexed.

7.103.2 Member Data Documentation

7.103.2.1 column

column: `column: int`

7.103.2.2 diags

diags: `diags: DiagCollector*`

7.103.2.3 filename

filename: `filename: string`

7.103.2.4 index

index: `index: int`

7.103.2.5 length

length: `length: int`

7.103.2.6 line

line: `line: int`

7.103.2.7 pending_errors

pending_errors: `pending_errors: DiagnosticVector*`

7.103.2.8 prev_ends_expression

prev_ends_expression: `prev_ends_expression: bool`

7.103.2.9 queued_index

queued_index: `queued_index: int`

7.103.2.10 queued_tokens

queued_tokens: `queued_tokens: TokenVector`

7.103.2.11 source

source: `source: string`

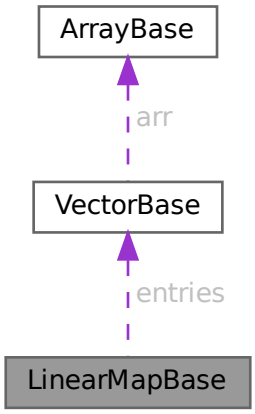
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/lexer.l0

7.104 LinearMapBase Struct Reference

Base structure for a linear map.

Collaboration diagram for LinearMapBase:



Public Attributes

- entries: VectorBase∗
- key_size : int
- value_size: int

7.104.1 Detailed Description

Base structure for a linear map.

7.104.2 Member Data Documentation

7.104.2.1 entries

entries: `entries: VectorBase*`

7.104.2.2 key_size

key_size: `key_size : int`

7.104.2.3 value_size

value_size: `value_size: int`

The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/linear_map.l0

7.105 LinearMapEntryBase Struct Reference

Linear map implementation using linear search.

Public Attributes

- hash_key: int
- hash_value: int

7.105.1 Detailed Description

Linear map implementation using linear search.

Note: This is not a hash map. Elements are stored in a vector, and lookups are done via linear search. Lookup can be done by key or by value, in O(n) time. Only efficient for very small datasets.

7.105.2 Member Data Documentation

7.105.2.1 hash_key

hash_key: `hash_key: int`

7.105.2.2 hash_value

hash_value: `hash_value: int`

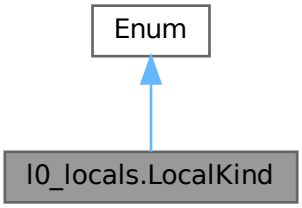
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/linear_map.l0

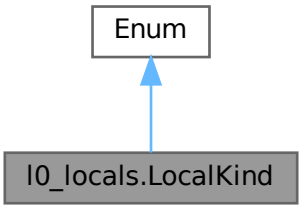
7.106 l0_locals.LocalKind Class Reference

Enumeration of the different kinds of local bindings.

Inheritance diagram for l0_locals.LocalKind:



Collaboration diagram for l0_locals.LocalKind:



Static Public Attributes

- PARAM = auto()
- LOCAL = auto()
- PATTERN_VAR = auto()

7.106.1 Detailed Description

Enumeration of the different kinds of local bindings.

Attributes

- `PARAM`: A function parameter.
- `LOCAL`: A variable declared via a 'let' statement.
- `PATTERN_VAR`: A variable bound in a 'match' variant pattern.

7.106.2 Member Data Documentation

7.106.2.1 LOCAL

```
l0_locals.LocalKind.LOCAL = auto()  [static]
```

7.106.2.2 PARAM

```
l0_locals.LocalKind.PARAM = auto()  [static]
```

7.106.2.3 PATTERN_VAR

```
l0_locals.LocalKind.PATTERN_VAR = auto()  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_locals.py

7.107 l0_locals.LocalScopeResolver Class Reference

Builds lexical scopes for functions within a compilation unit.

Public Member Functions

- None __init__ (self, Dict[str, Module] modules)
  Initialize the local scope resolver.
- Dict[Tuple[str, str], FunctionEnv] resolve (self)
  Build scopes for all non-extern functions in the modules.
- Optional[Scope] get_block_scope (self, Block block)
  Get the lexical scope associated with a block.
- Optional[Scope] get_match_arm_scope (self, MatchArm arm)
  Get the lexical scope associated with a match arm.

Public Attributes

- modules

Protected Member Functions

- FunctionEnv _build_function_env (self, str module_name, FuncDecl func)
  Build the environment and root scope for a single function.
- None _visit_block (self, Block block, Scope scope)
  Traverse a block and populate scopes.
- None _visit_stmt (self, Stmt stmt, Scope scope)
  Visit a statement and handle any scope introductions.
- None _visit_with_stmt (self, WithStmt stmt, Scope parent_scope)
  Visit a with statement and manage its header and body scopes.
- None _visit_match_arm (self, MatchArm arm, Scope parent_scope)
  Visit a match arm and create its child scope.
- None _visit_case_arm (self, CaseArm arm, Scope parent_scope)
  Visit a case arm.
- None _visit_case_else (self, CaseElse arm, Scope parent_scope)
  Visit a case else arm.
- None _bind_pattern_vars (self, Pattern pattern, Scope scope)
  Bind variables introduced by a pattern in the given scope.
- LocalSymbol _declare (self, Scope scope, str name, LocalKind kind, Optional[TypeRef] type_ref, Node decl)
  Declare a local symbol in the given scope.

7.107.1 Detailed Description

Builds lexical scopes for functions within a compilation unit.

This resolver traverses function bodies and creates nested scopes for blocks, loops, and match arms.

Attributes

- `modules`: Mapping of module names to Module AST nodes.
- `function_envs`: Mapping of (module, func) names to FunctionEnv objects.

Note

- Parameters live in the function's root scope.
- Top-level lets inside the function body also live in the root scope.
- Each nested Block inside control flow structures gets its own child scope.
- Pattern variables in a 'match' arm live in the arm's scope.

See also

`get_block_scope` and `get_match_arm_scope` methods for retrieving scopes by AST node.

7.107.2 Constructor & Destructor Documentation

7.107.2.1 __init__()

```
None l0_locals.LocalScopeResolver.__init__ (
             self,
            Dict[str, Module] modules )
```

Initialize the local scope resolver.

Parameters

modules	The modules to analyze.

7.107.3 Member Function Documentation

7.107.3.1 _bind_pattern_vars()

```
None l0_locals.LocalScopeResolver._bind_pattern_vars (
              self,
             Pattern pattern,
             Scope scope )  [protected]
```

Bind variables introduced by a pattern in the given scope.

7.107.3.2 _build_function_env()

```
FunctionEnv l0_locals.LocalScopeResolver._build_function_env (
              self,
             str module_name,
             FuncDecl func )  [protected]
```

Build the environment and root scope for a single function.

7.107.3.3 _declare()

```
LocalSymbol l0_locals.LocalScopeResolver._declare (
              self,
             Scope scope,
             str name,
             LocalKind kind,
             Optional[TypeRef] type_ref,
             Node decl )  [protected]
```

Declare a local symbol in the given scope.

7.107.3.4 _visit_block()

```
None l0_locals.LocalScopeResolver._visit_block (
              self,
             Block block,
             Scope scope )  [protected]
```

Traverse a block and populate scopes.

7.107.3.5 _visit_case_arm()

```
None l0_locals.LocalScopeResolver._visit_case_arm (
              self,
             CaseArm arm,
             Scope parent_scope )  [protected]
```

Visit a case arm.

7.107.3.6 _visit_case_else()

```
None l0_locals.LocalScopeResolver._visit_case_else (
              self,
             CaseElse arm,
             Scope parent_scope )  [protected]
```

Visit a case else arm.

7.107.3.7 _visit_match_arm()

```
None l0_locals.LocalScopeResolver._visit_match_arm (
              self,
             MatchArm arm,
             Scope parent_scope )  [protected]
```

Visit a match arm and create its child scope.

7.107.3.8 _visit_stmt()

```
None l0_locals.LocalScopeResolver._visit_stmt (
              self,
             Stmt stmt,
             Scope scope )  [protected]
```

Visit a statement and handle any scope introductions.

7.107.3.9 _visit_with_stmt()

```
None l0_locals.LocalScopeResolver._visit_with_stmt (
              self,
             WithStmt stmt,
             Scope parent_scope )  [protected]
```

Visit a with statement and manage its header and body scopes.

7.107.3.10 get_block_scope()

```
Optional[Scope] l0_locals.LocalScopeResolver.get_block_scope (
              self,
             Block block )
```

Get the lexical scope associated with a block.

Parameters

block	The Block AST node.

Returns

The associated Scope if found, otherwise None.

7.107.3.11 get_match_arm_scope()

```
Optional[Scope] l0_locals.LocalScopeResolver.get_match_arm_scope (
              self,
             MatchArm arm )
```

Get the lexical scope associated with a match arm.

Parameters

arm	The MatchArm AST node.

Returns

The associated Scope if found, otherwise None.

7.107.3.12 resolve()

```
Dict[Tuple[str, str], FunctionEnv] l0_locals.LocalScopeResolver.resolve (
              self )
```

Build scopes for all non-extern functions in the modules.

Returns

A mapping from function identifiers to their environments.

7.107.4 Member Data Documentation

7.107.4.1 modules

```
l0_locals.LocalScopeResolver.modules
```

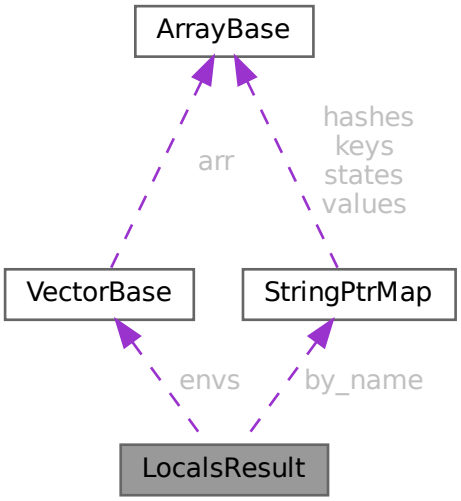
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_locals.py

7.108 LocalsResult Struct Reference

Represents the LocalsResult structure.

Collaboration diagram for LocalsResult:



Public Attributes

- envs: VectorBase∗
- by_name: StringPtrMap∗

7.108.1 Detailed Description

Represents the LocalsResult structure.

7.108.2 Member Data Documentation

7.108.2.1 by_name

by_name: `by_name: StringPtrMap∗`

7.108.2.2 envs

envs: `envs: VectorBase∗`

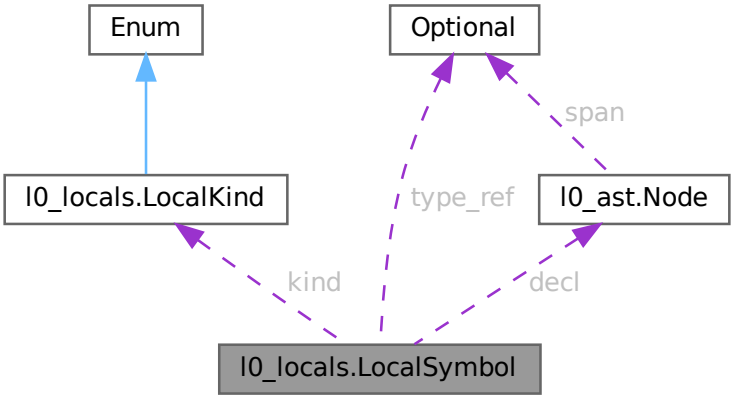
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/locals.l0

7.109 l0_locals.LocalSymbol Class Reference

Represents a single local binding.

Collaboration diagram for l0_locals.LocalSymbol:



Static Public Attributes

- str name
- LocalKind kind
- Optional type_ref [TypeRef]
- Node decl

7.109.1 Detailed Description

Represents a single local binding.

Attributes

- `name`: The name of the binding.
- `kind`: The kind of binding (parameter, local or pattern variable).
- `type_ref`: Optional type reference from the declaration.
- `decl`: The AST node that introduced the binding.

7.109.2 Member Data Documentation

7.109.2.1 decl

```
Node l0_locals.LocalSymbol.decl  [static]
```

7.109.2.2 kind

```
LocalKind l0_locals.LocalSymbol.kind  [static]
```

7.109.2.3 name

```
str l0_locals.LocalSymbol.name  [static]
```

7.109.2.4 type_ref

```
Optional l0_locals.LocalSymbol.type_ref [TypeRef]  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_locals.py

7.110 LocalSymbol Struct Reference

Represents the LocalSymbol structure.

Public Attributes

- name: string
- kind: LocalKind
- decl_ptr: void*
- decl_id: int
- binding_slot: int

7.110.1 Detailed Description

Represents the LocalSymbol structure.

7.110.2 Member Data Documentation

7.110.2.1 binding_slot

binding_slot: `binding_slot:  int`

7.110.2.2 decl_id

decl_id: `decl_id:  int`

7.110.2.3 decl_ptr

decl_ptr: `decl_ptr: void*`

7.110.2.4 kind

kind: `kind: LocalKind`

7.110.2.5 name

name: `name: string`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/locals.l0

7.111 LogConfig Struct Reference

Represents the LogConfig structure.

Public Attributes

- level: LogLevel
- rich: bool

7.111.1 Detailed Description

Represents the LogConfig structure.

7.111.2 Member Data Documentation

7.111.2.1 level

level: `level: LogLevel`

7.111.2.2 rich

rich: `rich: bool`

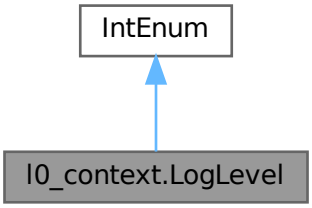
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/util/log.l0

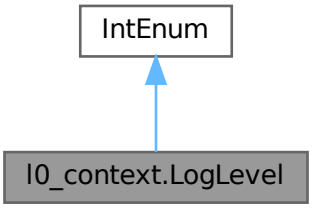
7.112 l0_context.LogLevel Class Reference

Hierarchical logging levels for the L0 compiler.

Inheritance diagram for l0_context.LogLevel:



Collaboration diagram for l0_context.LogLevel:



Static Public Attributes

- int SILENT = 0
- int ERROR = 3
- int WARNING = 6
- int INFO = 10
- int DEBUG = 30

7.112.1 Detailed Description

Hierarchical logging levels for the L0 compiler.

Attributes

- `SILENT`: No logging.
- `ERROR`: Error messages only.
- `WARNING`: Warning messages.
- `INFO`: General progress messages (-v).
- `DEBUG`: Detailed diagnostic information (-vvv).

7.112.2 Member Data Documentation

7.112.2.1 DEBUG

```
int l0_context.LogLevel.DEBUG = 30  [static]
```

7.112.2.2 ERROR

```
int l0_context.LogLevel.ERROR = 3  [static]
```

7.112.2.3 INFO

```
int l0_context.LogLevel.INFO = 10  [static]
```

7.112.2.4 SILENT

```
int l0_context.LogLevel.SILENT = 0  [static]
```

7.112.2.5 WARNING

```
int l0_context.LogLevel.WARNING = 6  [static]
```

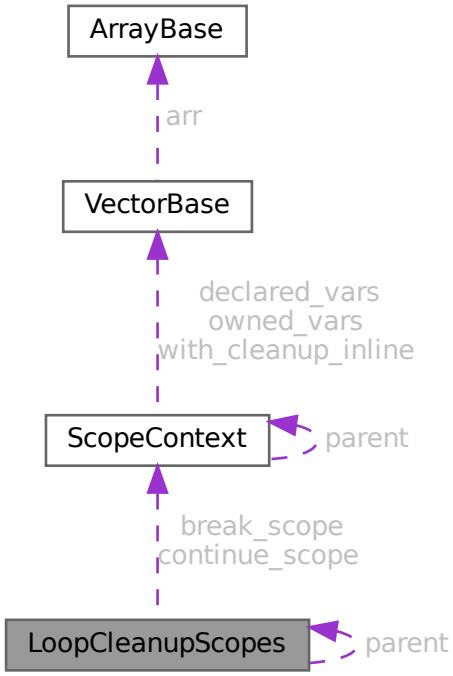
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_context.py

7.113 LoopCleanupScopes Struct Reference

Represents the cleanup targets for one active loop.

Collaboration diagram for LoopCleanupScopes:



Public Attributes

- parent: LoopCleanupScopes∗?
- continue_scope: ScopeContext∗
- break_scope: ScopeContext∗
- continue_scope_id: int
- break_scope_id: int

7.113.1 Detailed Description

Represents the cleanup targets for one active loop.

7.113.2 Member Data Documentation

7.113.2.1 break_scope

break_scope: `break_scope: ScopeContext*`

7.113.2.2 break_scope_id

break_scope_id: `break_scope_id: int`

7.113.2.3 continue_scope

continue_scope: `continue_scope: ScopeContext*`

7.113.2.4 continue_scope_id

continue_scope_id: `continue_scope_id: int`

7.113.2.5 parent

parent: `parent: LoopCleanupScopes*?`

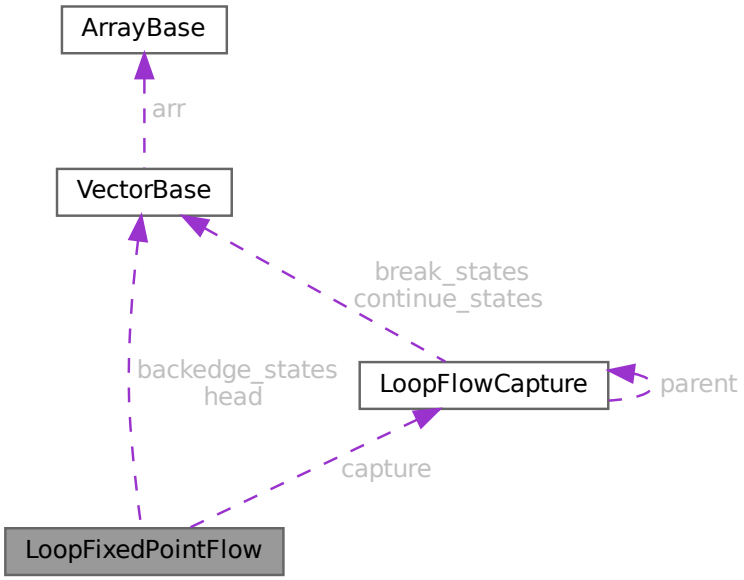
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/backend.l0

7.114 LoopFixedPointFlow Struct Reference

Result of fixed-point liveness convergence for one loop.

Collaboration diagram for LoopFixedPointFlow:



Public Attributes

- head: VectorBase∗
- capture: LoopFlowCapture∗
- backedge_states: VectorBase∗

7.114.1 Detailed Description

Result of fixed-point liveness convergence for one loop.

7.114.2 Member Data Documentation

7.114.2.1 backedge_states

backedge_states: `backedge_states: VectorBase∗`

7.114.2.2 capture

capture: `capture: LoopFlowCapture∗`

7.114.2.3 head

head: `head: VectorBase∗`

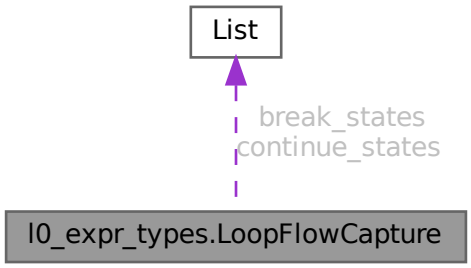
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/expr_types.l0

7.115 l0_expr_types.LoopFlowCapture Class Reference

Liveness states for loop-control exits from one loop body.

Collaboration diagram for l0_expr_types.LoopFlowCapture:



Static Public Attributes

- List break_states = field(default_factory=list)
- List continue_states = field(default_factory=list)

7.115.1 Detailed Description

Liveness states for loop-control exits from one loop body.

7.115.2 Member Data Documentation

7.115.2.1 break_states

```
List l0_expr_types.LoopFlowCapture.break_states = field(default_factory=list)  [static]
```

7.115.2.2 continue_states

```
List l0_expr_types.LoopFlowCapture.continue_states = field(default_factory=list)  [static]
```

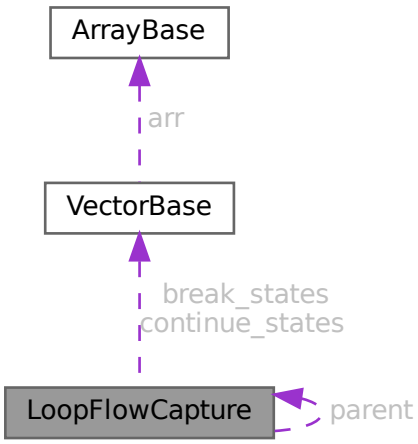
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_expr_types.py

7.116 LoopFlowCapture Struct Reference

Captured liveness states for loop-control exits from one loop body.

Collaboration diagram for LoopFlowCapture:



Public Attributes

- parent: LoopFlowCapture*?
- break_states: VectorBase*
- continue_states: VectorBase*

7.116.1 Detailed Description

Captured liveness states for loop-control exits from one loop body.

7.116.2 Member Data Documentation

7.116.2.1 break_states

break_states: `break_states: VectorBase*`

7.116.2.2 continue_states

continue_states: `continue_states: VectorBase*`

7.116.2.3 parent

parent: `parent: LoopFlowCapture*?`

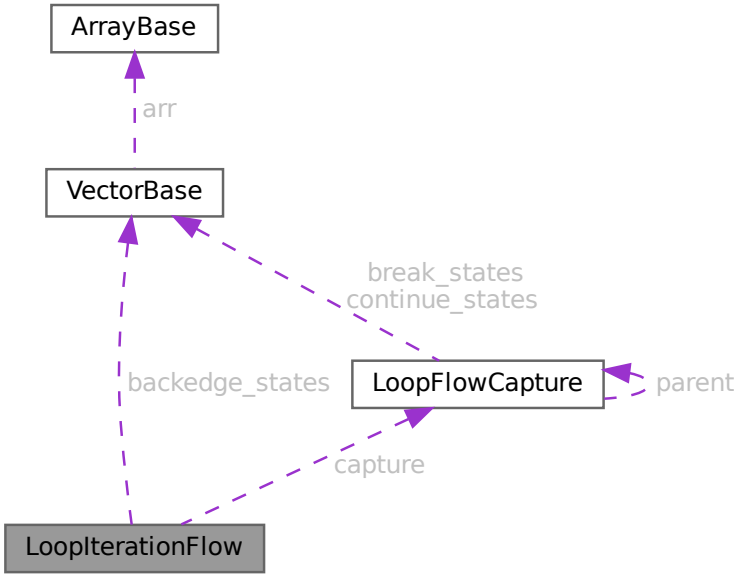
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/expr_types.l0

7.117 LoopIterationFlow Struct Reference

Result of checking one loop iteration for liveness flow.

Collaboration diagram for LoopIterationFlow:



Public Attributes

- capture: LoopFlowCapture*
- backedge_states: VectorBase*
- body_flow: StmtFlowKind

7.117.1 Detailed Description

Result of checking one loop iteration for liveness flow.

7.117.2 Member Data Documentation

7.117.2.1 backedge_states

backedge_states: `backedge_states: VectorBase*`

7.117.2.2 body_flow

body_flow: `body_flow: StmtFlowKind`

7.117.2.3 capture

capture: `capture: LoopFlowCapture*`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/expr_types.l0

7.118 LoopLabels Struct Reference

Represents one active loop label pair.

Collaboration diagram for LoopLabels:

Public Attributes

- parent: LoopLabels*?
- break_label: string
- continue_label: string

7.118.1 Detailed Description

Represents one active loop label pair.

7.118.2 Member Data Documentation

7.118.2.1 break_label

break_label: `break_label: string`

7.118.2.2 continue_label

continue_label: `continue_label: string`

7.118.2.3 parent

parent: `parent: LoopLabels*?`

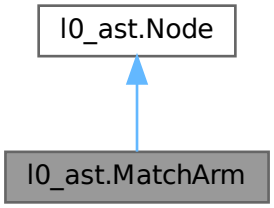
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/backend.l0

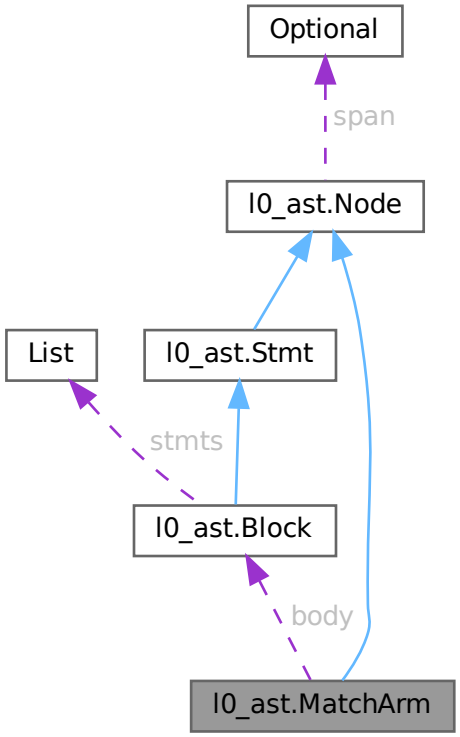
7.119 l0_ast.MatchArm Class Reference

A single arm in a match statement.

Inheritance diagram for l0_ast.MatchArm:



Collaboration diagram for l0_ast.MatchArm:



Static Public Attributes

- str pattern : "Pattern"
- Block body

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.119.1 Detailed Description

A single arm in a match statement.

Attributes

- `pattern`: The pattern to match against.
- `body`: The block to execute if the pattern matches.

7.119.2 Member Data Documentation

7.119.2.1 body

```
Block l0_ast.MatchArm.body  [static]
```

7.119.2.2 pattern

```
str l0_ast.MatchArm.pattern :  "Pattern"  [static]
```

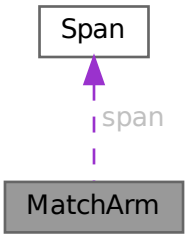
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.120 MatchArm Struct Reference

Represents the MatchArm structure.

Collaboration diagram for MatchArm:



Public Attributes

- pattern: PatternId
- body: StmtId
- span: Span

7.120.1 Detailed Description

Represents the MatchArm structure.

7.120.2 Member Data Documentation

7.120.2.1 body

body: `body: StmtId`

7.120.2.2 pattern

pattern: `pattern: PatternId`

7.120.2.3 span

span: `span: Span`

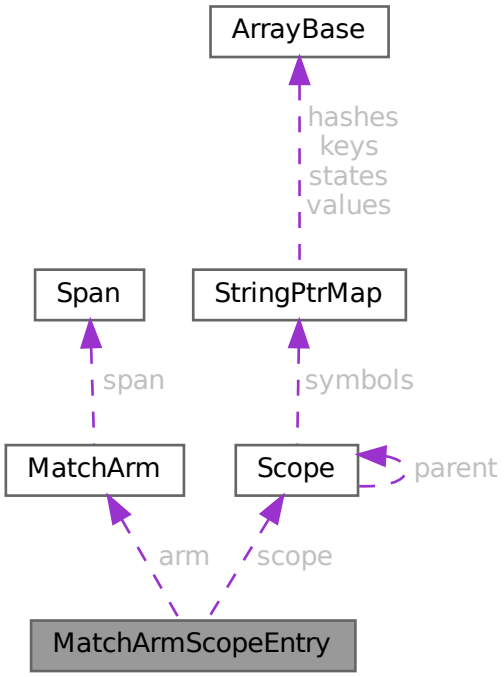
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

7.121 MatchArmScopeEntry Struct Reference

Represents the MatchArmScopeEntry structure.

Collaboration diagram for MatchArmScopeEntry:



Public Attributes

- arm: MatchArm*
- scope: Scope*

7.121.1 Detailed Description

Represents the MatchArmScopeEntry structure.

7.121.2 Member Data Documentation

7.121.2.1 arm

arm: `arm: MatchArm*`

7.121.2.2 scope

scope: `scope: Scope*`

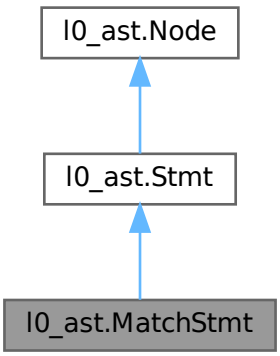
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/locals.l0

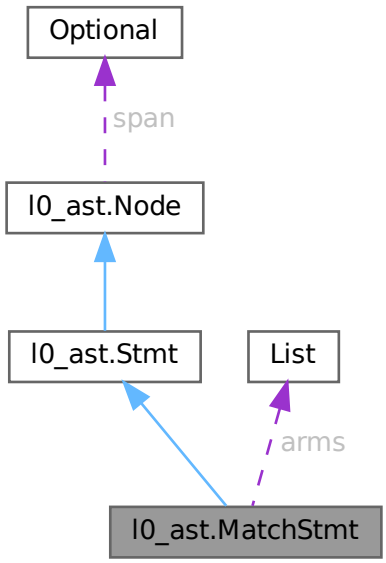
7.122 l0_ast.MatchStmt Class Reference

A match statement for pattern matching.

Inheritance diagram for l0_ast.MatchStmt:



Collaboration diagram for l0_ast.MatchStmt:



Static Public Attributes

- str expr : "Expr"
- List arms [MatchArm]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.122.1 Detailed Description

A match statement for pattern matching.

Attributes

- `expr`: The expression to match.
- `arms`: List of match arms.

7.122.2 Member Data Documentation

7.122.2.1 arms

```
List l0_ast.MatchStmt.arms [MatchArm]  [static]
```

7.122.2.2 expr

```
str l0_ast.MatchStmt.expr :  "Expr"  [static]
```

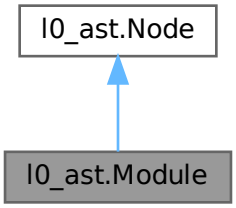
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

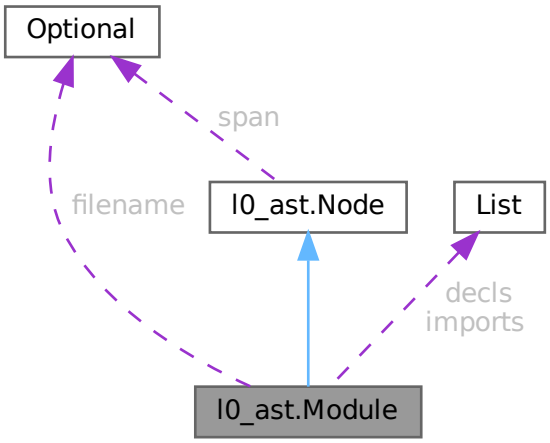
7.123 l0_ast.Module Class Reference

A complete L0 module.

Inheritance diagram for l0_ast.Module:



Collaboration diagram for l0_ast.Module:



Static Public Attributes

- str name
- List imports [Import]
- List decls [TopLevelDecl]
- Optional filename = field(default=None, repr=False, compare=False, kw_only=True)

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.123.1 Detailed Description

A complete L0 module.

Attributes

- `name`: The name of the module.
- `imports`: List of module imports.
- `decls`: List of top-level declarations.
- `filename`: Optional source file path.

7.123.2 Member Data Documentation

7.123.2.1 decls

```
List l0_ast.Module.decls [TopLevelDecl]  [static]
```

7.123.2.2 filename

```
Optional l0_ast.Module.filename = field(default=None, repr=False, compare=False, kw_only=True)
[static]
```

7.123.2.3 imports

```
List l0_ast.Module.imports [Import]  [static]
```

7.123.2.4 name

```
str l0_ast.Module.name  [static]
```

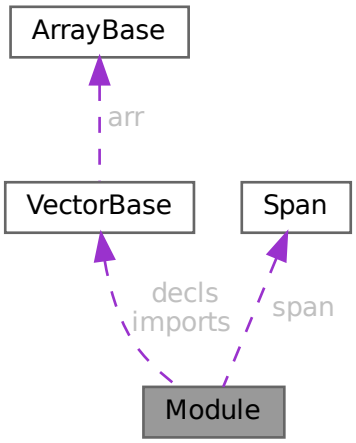
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.124 Module Struct Reference

Represents the Module structure.

Collaboration diagram for Module:



Public Attributes

- name: string
- imports: VectorBase∗
- decls: VectorBase∗
- span: Span
- filename: string

7.124.1 Detailed Description

Represents the Module structure.

7.124.2 Member Data Documentation

7.124.2.1 decls

decls: `decls: VectorBase*`

7.124.2.2 filename

filename: `filename: string`

7.124.2.3 imports

imports: `imports: VectorBase*`

7.124.2.4 name

name: `name: string`

7.124.2.5 span

span: `span: Span`

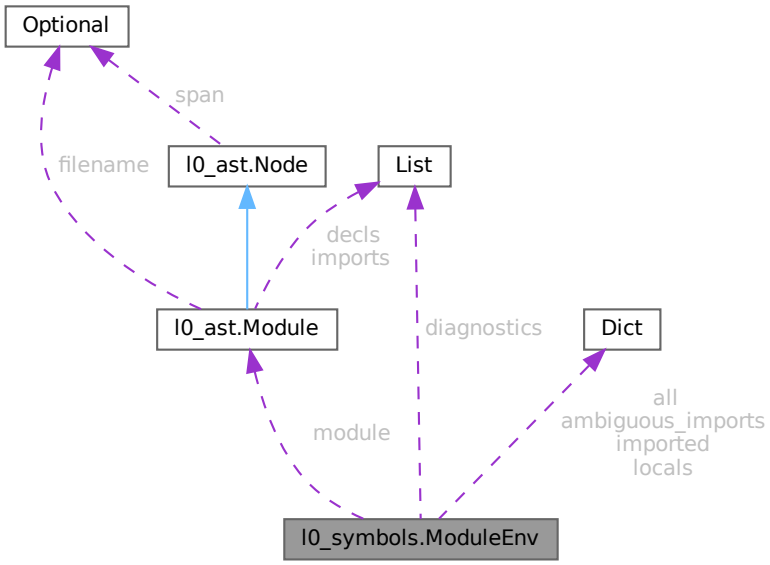
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

7.125 l0_symbols.ModuleEnv Class Reference

Per-module symbol environment.

Collaboration diagram for l0_symbols.ModuleEnv:



Public Member Functions

- str name (self)

  Get the name of the module associated with this environment.

Static Public Attributes

- Module module
- Dict locals = field(default_factory=dict)
- Dict imported = field(default_factory=dict)
- Dict all = field(default_factory=dict)
- Dict ambiguous_imports = field(default_factory=dict)
- List diagnostics = field(default_factory=list)

7.125.1 Detailed Description

Per-module symbol environment.

Attributes

- `module`: The Module AST node for this environment.
- `locals`: Mapping of symbols defined within this module.
- `imported`: Mapping of symbols imported from other modules.
- `all`: Effective view of all symbols visible in this module (locals + imported), with ambiguous names removed.
- `ambiguous_imports`: Mapping of ambiguous names to the list of modules they were imported from.
- `diagnostics`: List of diagnostics collected during name resolution.

7.125.2 Member Function Documentation

7.125.2.1 name()

```
str l0_symbols.ModuleEnv.name (
              self )
```

Get the name of the module associated with this environment.

7.125.3 Member Data Documentation

7.125.3.1 all

```
Dict l0_symbols.ModuleEnv.all = field(default_factory=dict)  [static]
```

7.125.3.2 ambiguous_imports

```
Dict l0_symbols.ModuleEnv.ambiguous_imports = field(default_factory=dict)  [static]
```

7.125.3.3 diagnostics

```
List l0_symbols.ModuleEnv.diagnostics = field(default_factory=list)  [static]
```

7.125.3.4 imported

```
Dict l0_symbols.ModuleEnv.imported = field(default_factory=dict)  [static]
```

7.125.3.5 locals

```
Dict l0_symbols.ModuleEnv.locals = field(default_factory=dict)  [static]
```

7.125.3.6 module

```
Module l0_symbols.ModuleEnv.module  [static]
```

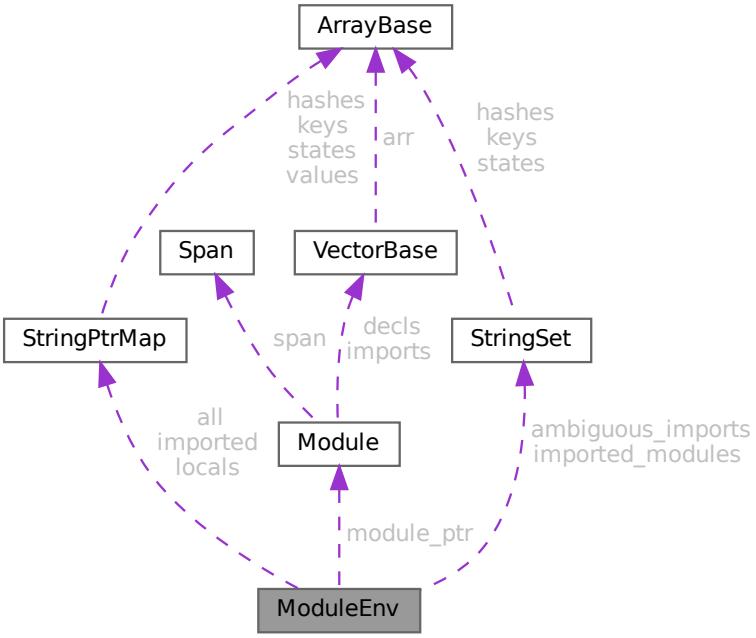
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_symbols.py

7.126 ModuleEnv Struct Reference

Represents the ModuleEnv structure.

Collaboration diagram for ModuleEnv:



Public Attributes

- module_name: string
- module_ptr: Module*
- locals: StringPtrMap*
- imported: StringPtrMap*
- all: StringPtrMap*
- ambiguous_imports: StringSet*
- imported_modules: StringSet*

7.126.1 Detailed Description

Represents the ModuleEnv structure.

7.126.2 Member Data Documentation

7.126.2.1 all

all: `all: StringPtrMap*`

7.126.2.2 ambiguous_imports

ambiguous_imports: `ambiguous_imports: StringSet*`

7.126.2.3 imported

imported: `imported: StringPtrMap*`

7.126.2.4 imported_modules

imported_modules: `imported_modules: StringSet*`

7.126.2.5 locals

locals: `locals: StringPtrMap*`

7.126.2.6 module_name

module_name: `module_name: string`

7.126.2.7 module_ptr

module_ptr: `module_ptr: Module*`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/name_resolver.l0

7.127 MonotonicTime Struct Reference

Represents monotonic time in seconds and nanoseconds.

Public Attributes

- sec: int
- nsec: int

7.127.1 Detailed Description

Represents monotonic time in seconds and nanoseconds.

7.127.2 Member Data Documentation

7.127.2.1 nsec

nsec: `nsec: int`

7.127.2.2 sec

sec: `sec: int`

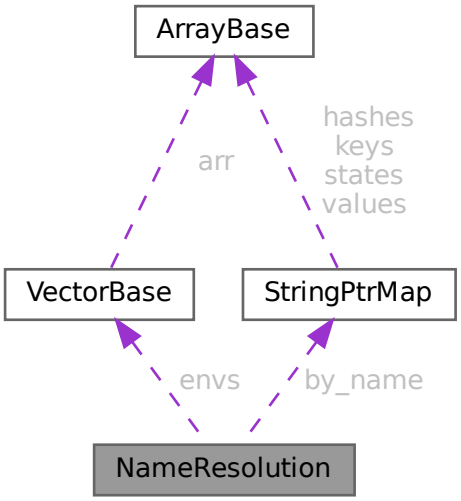
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/time.l0

7.128 NameResolution Struct Reference

Represents the NameResolution structure.

Collaboration diagram for NameResolution:



Public Attributes

- envs: VectorBase∗
- by_name: StringPtrMap∗

7.128.1 Detailed Description

Represents the NameResolution structure.

7.128.2 Member Data Documentation

7.128.2.1 by_name

by_name: `by_name: StringPtrMap∗`

7.128.2.2 envs

envs: `envs: VectorBase∗`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/name_resolver.l0

7.129 l0_name_resolver.NameResolver Class Reference

Stage-1 name resolver for module-level symbols.

Public Member Functions

- __init__ (self, CompilationUnit cu)
  Initialize the name resolver.
- Dict[str, ModuleEnv] resolve (self)
  Resolve names for all modules in the compilation unit.

Public Attributes

- cu

Protected Member Functions

- None _collect_locals (self, ModuleEnv env)
  Collect local symbols from module declarations.
- None _define_local (self, ModuleEnv env, str name, Symbol sym)
  Define a local symbol, checking for duplicates.
- None _open_imports (self, ModuleEnv env)
  Process imports for a module environment.
- bool _extern_signatures_compatible (self, Symbol local, Symbol imported)
  Check if two extern function signatures are compatible.

7.129.1 Detailed Description

Stage-1 name resolver for module-level symbols.

This resolver:

- Builds a ModuleEnv for each module in a CompilationUnit.
- Collects top-level symbols (functions, structs, enums, variants, aliases).
- Processes imports by injecting imported symbols into the importer's scope.
- Detects duplicate definitions and import conflicts.

Attributes

- `cu`: The compilation unit to resolve.
- `module_envs`: Mapping of module names to their built environments.
- `diagnostics`: List of collected diagnostics.

7.129.2 Constructor & Destructor Documentation

7.129.2.1 __init__()

```
l0_name_resolver.NameResolver.__init__ (
            self,
            CompilationUnit cu )
```

Initialize the name resolver.

Parameters

cu	The compilation unit to process.

7.129.3 Member Function Documentation

7.129.3.1 _collect_locals()

```
None l0_name_resolver.NameResolver._collect_locals (
              self,
             ModuleEnv env )  [protected]
```

Collect local symbols from module declarations.

Parameters

env	The module environment to populate.

7.129.3.2 _define_local()

```
None l0_name_resolver.NameResolver._define_local (
              self,
             ModuleEnv env,
             str name,
             Symbol sym )  [protected]
```

Define a local symbol, checking for duplicates.

Parameters

env	The environment to define the symbol in.
name	The name of the symbol.
sym	The Symbol object.

7.129.3.3 _extern_signatures_compatible()

```
bool l0_name_resolver.NameResolver._extern_signatures_compatible (
              self,
             Symbol local,
             Symbol imported )  [protected]
```

Check if two extern function signatures are compatible.

7.129.3.4 _open_imports()

```
None l0_name_resolver.NameResolver._open_imports (
              self,
             ModuleEnv env )  [protected]
```

Process imports for a module environment.

Parameters

env	The module environment whose imports should be opened.

Note

Apply the following rules when opening imports:

- For each imported module, add its locals into `env.imported` and `env.all`.
- If a name collides with a local, keep the local and issue a warning.
- If a name is imported from multiple modules, mark it ambiguous, remove it from `env.all` and issue a warning; later resolution will see no binding.

7.129.3.5 resolve()

```
Dict[str, ModuleEnv] l0_name_resolver.NameResolver.resolve (
              self )
```

Resolve names for all modules in the compilation unit.

Returns

A mapping from module names to their populated ModuleEnv objects.

7.129.4 Member Data Documentation

7.129.4.1 cu

```
l0_name_resolver.NameResolver.cu
```

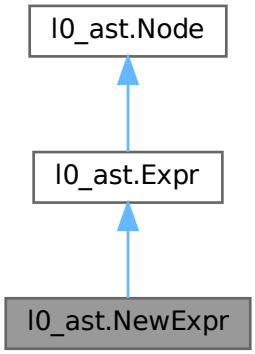
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_name_resolver.py

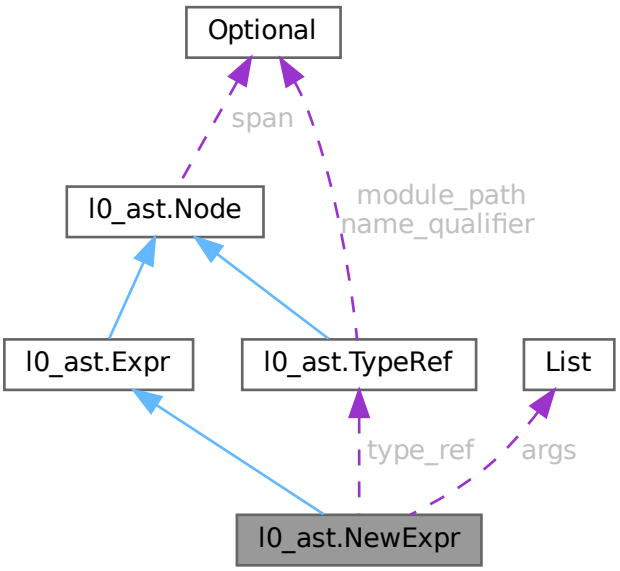
7.130 l0_ast.NewExpr Class Reference

A heap allocation or constructor expression.

Inheritance diagram for l0_ast.NewExpr:



Collaboration diagram for l0_ast.NewExpr:



Static Public Attributes

- TypeRef type_ref
- List args [Expr]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.130.1 Detailed Description

A heap allocation or constructor expression.

Attributes

- `type_ref`: The type to instantiate.
- `args`: List of constructor arguments.

7.130.2 Member Data Documentation

7.130.2.1 args

```
List l0_ast.NewExpr.args [Expr]  [static]
```

7.130.2.2 type_ref

```
TypeRef l0_ast.NewExpr.type_ref  [static]
```

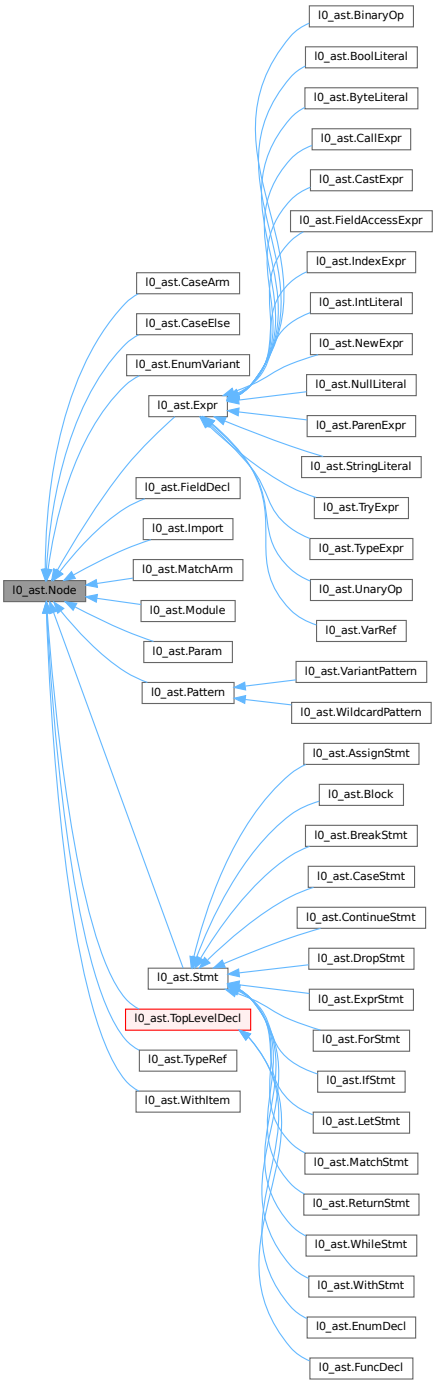
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

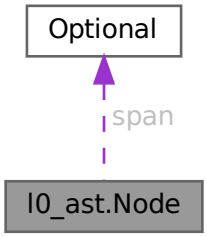
7.131 l0_ast.Node Class Reference

Base class for all AST nodes.

Inheritance diagram for l0_ast.Node:



Collaboration diagram for l0_ast.Node:



Static Public Attributes

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.131.1 Detailed Description

Base class for all AST nodes.

Attributes

- `span`: Optional source span information.

7.131.2 Member Data Documentation

7.131.2.1 span

```
Optional l0_ast.Node.span = field(default=None, repr=False, compare=False, kw_only=True)  [static]
```

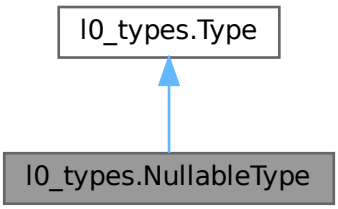
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

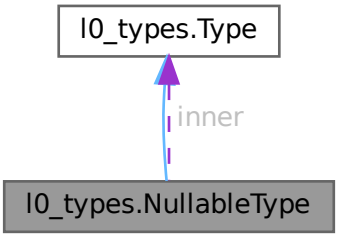
7.132 l0_types.NullableType Class Reference

A nullable (optional) wrapper for another type (T?).

Inheritance diagram for l0_types.NullableType:



Collaboration diagram for l0_types.NullableType:



Static Public Attributes

- Type inner

7.132.1 Detailed Description

A nullable (optional) wrapper for another type (T?).

Attributes

- `inner`: The Type that may be null.

7.132.2 Member Data Documentation

7.132.2.1 inner

```
Type l0_types.NullableType.inner  [static]
```

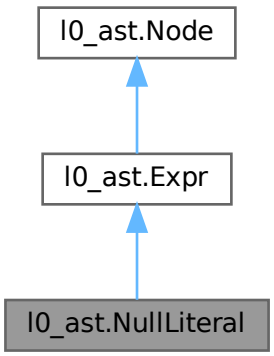
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_types.py

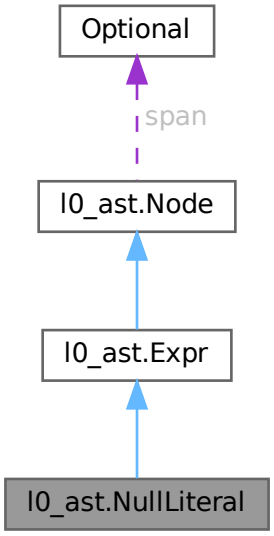
7.133 l0_ast.NullLiteral Class Reference

A null literal expression.

Inheritance diagram for l0_ast.NullLiteral:



Collaboration diagram for l0_ast.NullLiteral:



Additional Inherited Members

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.133.1 Detailed Description

A null literal expression.

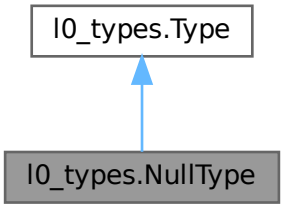
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

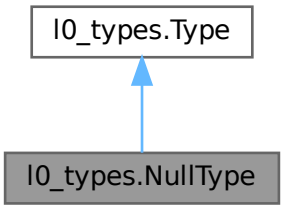
7.134 l0_types.NullType Class Reference

The type of the 'null' literal.

Inheritance diagram for l0_types.NullType:



Collaboration diagram for l0_types.NullType:



7.134.1 Detailed Description

The type of the 'null' literal.

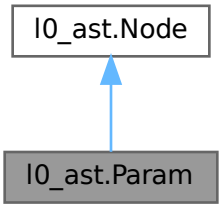
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_types.py

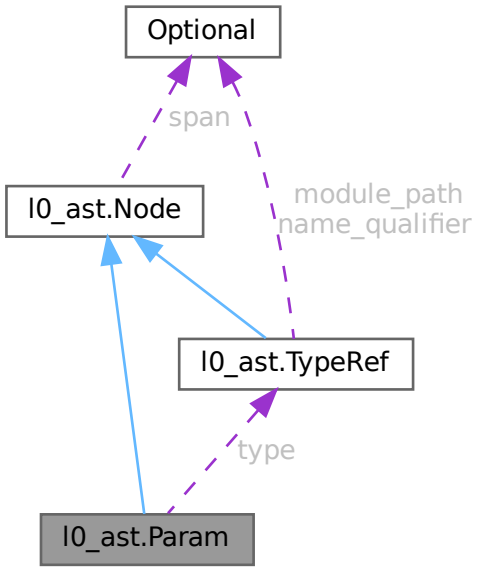
7.135 l0_ast.Param Class Reference

A function parameter declaration.

Inheritance diagram for l0_ast.Param:



Collaboration diagram for l0_ast.Param:



Static Public Attributes

- str name
- TypeRef type

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.135.1 Detailed Description

A function parameter declaration.

Attributes

- `name`: The name of the parameter.
- `type`: The type of the parameter.

7.135.2 Member Data Documentation

7.135.2.1 name

```
str l0_ast.Param.name  [static]
```

7.135.2.2 type

```
TypeRef l0_ast.Param.type  [static]
```

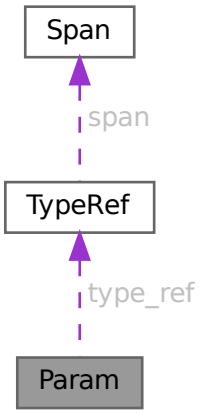
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.136 Param Struct Reference

Represents the Param structure.

Collaboration diagram for Param:



Public Attributes

- name: string
- type_ref: TypeRef*

7.136.1 Detailed Description

Represents the Param structure.

7.136.2 Member Data Documentation

7.136.2.1 name

name: `name: string`

7.136.2.2 type_ref

type_ref: `type_ref: TypeRef*`

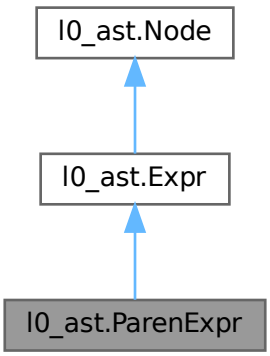
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

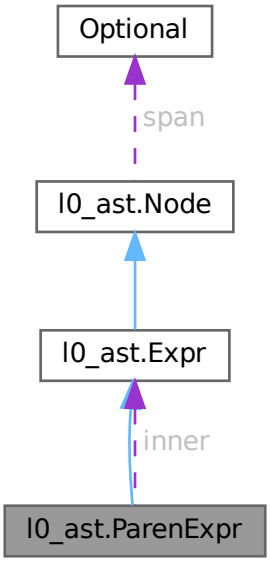
7.137 l0_ast.ParenExpr Class Reference

A parenthesized expression.

Inheritance diagram for l0_ast.ParenExpr:



Collaboration diagram for l0_ast.ParenExpr:



Static Public Attributes

- Expr inner

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.137.1 Detailed Description

A parenthesized expression.

Attributes

- `inner`: The expression inside the parentheses.

7.137.2 Member Data Documentation

7.137.2.1 inner

```
Expr l0_ast.ParenExpr.inner  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.138 l0_parser.Parser Class Reference

Recursive descent parser for L0.

Public Member Functions

- None __init__ (self, List[Token] tokens, Optional[str] filename=None, Optional[List[Diagnostic]] diagnostics=None)

  Initialize the parser.
- "Parser" from_source (cls, str source)

  Create a parser from a source string.
- Module parse_module (self, Optional[str] filename=None)

  Parse a complete L0 module.

Public Attributes

- tokens
- index
- filename
- diagnostics
- eof_aborted

Protected Member Functions

- None _error (self, str message, Optional[Token] token=None)

  Add an error diagnostic.
- NoReturn _error_bail (self, str message, Optional[Token] token=None)

  Report an error and raise a synchronization exception.
- NoReturn _error_unexpected (self, str error_code, Token tok, str context)

  Report an unexpected token error and raise a synchronization exception.
- Token _peek (self)

  Return the current token without advancing.
- None _emit_lexer_error (self, int index, Token tok)

  Emit deferred lexer diagnostics for a physical token once.
- Token _logical_token_at_cursor (self)

  Return the logical current token, skipping unrecoverable wrappers.
- None _skip_lexer_errors (self)

  Emit and step over unrecoverable LEXER_ERROR tokens at the cursor.
- None _emit_remaining_lexer_errors (self)

  Emit deferred lexer diagnostics that parsing never reached.
- Token _last (self)

  Return the previously advanced token.
- bool _at_end (self)

  Check if at the end of the token stream.
- Token _advance (self)

  Advance the current index and return the token.
- bool _check (self, TokenKind kind)

  Check if the current token is of the specified kind.
- bool _match (self, *TokenKind kinds)

  Advance and return True if the current token matches any of the kinds.

- Token _expect (self, TokenKind kind, str msg)

  Advance if the current token matches kind, otherwise bail.
- None _expect_semicolon (self, Optional[str] msg=None)

  Expect and consume a semicolon, reporting an error if missing.
- Token _expect_variable_name (self, str msg)

  Expect and consume a valid variable name (non-reserved identifier).
- Span _span_start (self)

  Create a zero-length span starting at the current token.
- Span _extend_span (self, Span start)

  Create a new span extending from start to the end of the last token.
- list[str] _get_dotted_module_name (self, Token first)

  Parse a dotted module name (e.g., a.b.c).
- Optional[tuple[list[str], Optional[list[str]], Token]] _try_parse_qualified_name (self)

  Try to parse a qualified name (e.g., mod::name).
- None _sync_top_level (self)

  Skip tokens until we find the start of a new top-level declaration or EOF.
- TopLevelDecl _parse_top_level_decl (self)

  Parse a single top-level declaration.
- FuncDecl _parse_extern_func (self)

  Parse an 'extern' function declaration.
- FuncDecl _parse_function (self, bool is_extern)

  Parse a function declaration or definition.
- StructDecl _parse_struct (self)

  Parse a struct definition.
- EnumDecl _parse_enum (self)

  Parse an enum definition.
- TypeAliasDecl _parse_type_alias (self)

  Parse a type alias declaration.
- LetDecl _parse_top_level_let (self)

  Parse a top-level 'let' binding.
- TypeRef _parse_type (self)

  Parse a type reference including pointer and nullability suffixes.
- Block _parse_block (self)

  Parse a block of statements enclosed in braces.
- None _sync_stmt (self)

  Skip tokens until we reach a statement boundary or end of block.
- Stmt _parse_stmt (self)

  Parse a single statement.
- Stmt _parse_simple_stmt (self)

  Parse a simple statement (let, break, return, assign, or expr).
- LetStmt _parse_let_stmt (self)

  Parse a local 'let' statement.
- IfStmt _parse_if_stmt (self)

  Parse an 'if' statement.
- WhileStmt _parse_while_stmt (self)

  Parse a 'while' loop statement.
- Stmt _parse_for_stmt (self)

  Parse a 'for' loop statement.
- Stmt _parse_for_update_stmt (self)

  Parse a statement permitted in a for-loop update clause.
- ReturnStmt _parse_return_stmt (self)

Parse a 'return' statement.

- DropStmt _parse_drop_stmt (self)

  Parse a 'drop' statement.
- MatchStmt _parse_match_stmt (self)

  Parse a 'match' statement.
- WithItem _parse_with_item (self)

  Parse a single item in a 'with' statement.
- WithStmt _parse_with_stmt (self)

  Parse a 'with' statement for resource management.
- CaseStmt _parse_case_stmt (self)

  Parse a 'case' statement (multi-way constant branch).
- bool _at_case_arm_start (self)

  Return True if the current token is a likely case-arm recovery boundary.
- None _sync_case_invalid_arm (self)

  Skip a malformed or forbidden case arm without leaving the case.
- Expr _parse_case_literal (self)

  Parse a literal value for a 'case' arm.
- Pattern _parse_pattern (self)

  Parse a pattern for 'match' arms.
- Stmt _parse_break_stmt (self)

  Parse a 'break' statement.
- Stmt _parse_continue_stmt (self)

  Parse a 'continue' statement.
- Expr _parse_expr (self)

  Parse an expression (entry point for expression parsing).
- None _check_reserved_binary_op (self)

  Raise a diagnostic if the next token is a reserved binary operator.
- Expr _parse_or_expr (self)

  Parse a logical OR expression.
- Expr _parse_and_expr (self)

  Parse a logical AND expression.
- Expr _parse_equality_expr (self)

  Parse an equality expression.
- Expr _parse_rel_expr (self)

  Parse a relational expression.
- Expr _parse_add_expr (self)

  Parse an addition/subtraction expression.
- Expr _parse_mul_expr (self)

  Parse a multiplication/division/modulo expression.
- Expr _parse_unary_expr (self)

  Parse a unary expression.
- Expr _parse_cast_expr (self)

  Parse a type cast expression.
- Expr _parse_postfix_expr (self)

  Parse a postfix expression (calls, indexing, field access, try).
- Expr _parse_call_argument (self)

  Parse a function call argument, attempting to parse as TypeExpr first.
- bool _is_builtin_type_name (self)

  Check if current token is a builtin type keyword.
- bool _lookahead_is_type_suffix (self)

  Check if there's a '' or '?' immediately after the current token.*

- bool _is_unambiguous_type_start (self)

  Check if the current position unambiguously starts a type reference.
- Expr _parse_primary_expr (self)

  Parse a primary expression (literals, identifiers, parenthesized expressions).

Protected Attributes

- _last_token

Static Protected Attributes

- dict _RESERVED_BINARY_OPS

7.138.1 Detailed Description

Recursive descent parser for L0.

Converts a flat list of tokens into a hierarchical AST. Implements error recovery via synchronization points.

Attributes

- `tokens`: The list of tokens to parse.
- `index`: Current position in the token list.
- `filename`: Name of the file being parsed (for diagnostics).
- `diagnostics`: Collected list of parse errors and warnings.

7.138.2 Constructor & Destructor Documentation

7.138.2.1 __init__()

```
None l0_parser.Parser.__init__ (
              self,
             List[Token] tokens,
             Optional[str]  filename = None,
             Optional[List[Diagnostic]]  diagnostics = None )
```

Initialize the parser.

Parameters

tokens	The tokens to parse.
filename	Optional source filename.
diagnostics	Optional list to collect diagnostics into.

7.138.3 Member Function Documentation

7.138.3.1 _advance()

```
Token l0_parser.Parser._advance (
              self )  [protected]
```

Advance the current index and return the token.

7.138.3.2 _at_case_arm_start()

```
bool l0_parser.Parser._at_case_arm_start (
              self )  [protected]
```

Return True if the current token is a likely case-arm recovery boundary.

7.138.3.3 _at_end()

```
bool l0_parser.Parser._at_end (
              self )  [protected]
```

Check if at the end of the token stream.

7.138.3.4 _check()

```
bool l0_parser.Parser._check (
              self,
             TokenKind kind )  [protected]
```

Check if the current token is of the specified kind.

7.138.3.5 _check_reserved_binary_op()

```
None l0_parser.Parser._check_reserved_binary_op (
              self )  [protected]
```

Raise a diagnostic if the next token is a reserved binary operator.

7.138.3.6 _emit_lexer_error()

```
None l0_parser.Parser._emit_lexer_error (
              self,
             int index,
             Token tok )  [protected]
```

Emit deferred lexer diagnostics for a physical token once.

7.138.3.7 _emit_remaining_lexer_errors()

```
None l0_parser.Parser._emit_remaining_lexer_errors (
              self )  [protected]
```

Emit deferred lexer diagnostics that parsing never reached.

7.138.3.8 _error()

```
None l0_parser.Parser._error (
              self,
             str message,
             Optional[Token]  token = None )  [protected]
```

Add an error diagnostic.

Parameters

message	The error message.
token	Optional token where the error occurred. Defaults to peek().

7.138.3.9 _error_bail()

```
NoReturn l0_parser.Parser._error_bail (
              self,
             str message,
             Optional[Token]  token = None )  [protected]
```

Report an error and raise a synchronization exception.

Parameters

message	The error message.
token	Optional token where the error occurred.

Exceptions

_ParseSyncException	Always raised to trigger recovery.

7.138.3.10 _error_unexpected()

```
NoReturn l0_parser.Parser._error_unexpected (
              self,
             str error_code,
             Token tok,
             str context )  [protected]
```

Report an unexpected token error and raise a synchronization exception.

Parameters

error_code	The diagnostic code (e.g., "[PAR-nnnn]").
tok	The unexpected token.
context	Description of what was being parsed.

7.138.3.11 _expect()

```
Token l0_parser.Parser._expect (
            self,
           TokenKind kind,
           str msg )  [protected]
```

Advance if the current token matches kind, otherwise bail.

Parameters

kind	The expected TokenKind.
msg	The error message if expectation fails.

Returns

The matched token.

7.138.3.12 _expect_semicolon()

```
None l0_parser.Parser._expect_semicolon (
            self,
           Optional[str]  msg = None )  [protected]
```

Expect and consume a semicolon, reporting an error if missing.

7.138.3.13 _expect_variable_name()

```
Token l0_parser.Parser._expect_variable_name (
            self,
           str msg )  [protected]
```

Expect and consume a valid variable name (non-reserved identifier).

7.138.3.14 _extend_span()

```
Span l0_parser.Parser._extend_span (
            self,
           Span start )  [protected]
```

Create a new span extending from start to the end of the last token.

7.138.3.15 _get_dotted_module_name()

```
list[str] l0_parser.Parser._get_dotted_module_name (
              self,
             Token first )  [protected]
```

Parse a dotted module name (e.g., a.b.c).

7.138.3.16 _is_builtin_type_name()

```
bool l0_parser.Parser._is_builtin_type_name (
              self )  [protected]
```

Check if current token is a builtin type keyword.

7.138.3.17 _is_unambiguous_type_start()

```
bool l0_parser.Parser._is_unambiguous_type_start (
              self )  [protected]
```

Check if the current position unambiguously starts a type reference.

Note

Unambiguous type cases:

- Builtin type name (`int`, `byte`, `bool`, `string`, `void`)
- Ident followed by '$*$' or '?' suffixes, ending at argument boundary: ',' or ')' **Examples:**
- sizeof(int$*$) $\rightarrow$ int is builtin $\rightarrow$ type
- sizeof(Point$*$) $\rightarrow$ Point$*$ followed by ')' $\rightarrow$ type
- sizeof(a $*$ b) $\rightarrow$ '$*$' followed by 'b', not boundary $\rightarrow$ expression
- sizeof(Point) $\rightarrow$ Point with no suffix $\rightarrow$ ambiguous, defaults to expr

7.138.3.18 _last()

```
Token l0_parser.Parser._last (
              self )  [protected]
```

Return the previously advanced token.

7.138.3.19 _logical_token_at_cursor()

```
Token l0_parser.Parser._logical_token_at_cursor (
              self )  [protected]
```

Return the logical current token, skipping unrecoverable wrappers.

7.138.3.20 _lookahead_is_type_suffix()

```
bool l0_parser.Parser._lookahead_is_type_suffix (
              self )  [protected]
```

Check if there's a '∗' or '?' immediately after the current token.

7.138.3.21 _match()

```
bool l0_parser.Parser._match (
              self,
             *TokenKind kinds )  [protected]
```

Advance and return True if the current token matches any of the kinds.

7.138.3.22 _parse_add_expr()

```
Expr l0_parser.Parser._parse_add_expr (
              self )  [protected]
```

Parse an addition/subtraction expression.

7.138.3.23 _parse_and_expr()

```
Expr l0_parser.Parser._parse_and_expr (
              self )  [protected]
```

Parse a logical AND expression.

7.138.3.24 _parse_block()

```
Block l0_parser.Parser._parse_block (
              self )  [protected]
```

Parse a block of statements enclosed in braces.

7.138.3.25 _parse_break_stmt()

```
Stmt l0_parser.Parser._parse_break_stmt (
              self )  [protected]
```

Parse a 'break' statement.

7.138.3.26 _parse_call_argument()

```
Expr l0_parser.Parser._parse_call_argument (
              self )  [protected]
```

Parse a function call argument, attempting to parse as TypeExpr first.

Note

In argument position, we can have either expressions or type expressions. We parse as TypeExpr only when syntactically unambiguous:

- Builtin type name: int, byte, bool, string, void
- Any name followed by ∗ or ?: Point∗, int?, Foo∗?

7.138.3.27 _parse_case_literal()

```
Expr l0_parser.Parser._parse_case_literal (
              self )  [protected]
```

Parse a literal value for a 'case' arm.

7.138.3.28 _parse_case_stmt()

```
CaseStmt l0_parser.Parser._parse_case_stmt (
              self )  [protected]
```

Parse a 'case' statement (multi-way constant branch).

7.138.3.29 _parse_cast_expr()

```
Expr l0_parser.Parser._parse_cast_expr (
              self )  [protected]
```

Parse a type cast expression.

7.138.3.30 _parse_continue_stmt()

```
Stmt l0_parser.Parser._parse_continue_stmt (
              self )  [protected]
```

Parse a 'continue' statement.

7.138.3.31 _parse_drop_stmt()

```
DropStmt l0_parser.Parser._parse_drop_stmt (
              self )  [protected]
```

Parse a 'drop' statement.

7.138.3.32 _parse_enum()

```
EnumDecl l0_parser.Parser._parse_enum (
              self )  [protected]
```

Parse an enum definition.

7.138.3.33 _parse_equality_expr()

```
Expr l0_parser.Parser._parse_equality_expr (
              self )  [protected]
```

Parse an equality expression.

7.138.3.34 _parse_expr()

```
Expr l0_parser.Parser._parse_expr (
              self )  [protected]
```

Parse an expression (entry point for expression parsing).

7.138.3.35 _parse_extern_func()

```
FuncDecl l0_parser.Parser._parse_extern_func (
              self )  [protected]
```

Parse an 'extern' function declaration.

7.138.3.36 _parse_for_stmt()

```
Stmt l0_parser.Parser._parse_for_stmt (
              self )  [protected]
```

Parse a 'for' loop statement.

7.138.3.37 _parse_for_update_stmt()

```
Stmt l0_parser.Parser._parse_for_update_stmt (
              self )  [protected]
```

Parse a statement permitted in a for-loop update clause.

7.138.3.38 _parse_function()

```
FuncDecl l0_parser.Parser._parse_function (
              self,
            bool is_extern )  [protected]
```

Parse a function declaration or definition.

7.138.3.39 _parse_if_stmt()

```
IfStmt l0_parser.Parser._parse_if_stmt (
              self )  [protected]
```

Parse an 'if' statement.

7.138.3.40 _parse_let_stmt()

```
LetStmt l0_parser.Parser._parse_let_stmt (
              self )  [protected]
```

Parse a local 'let' statement.

7.138.3.41 _parse_match_stmt()

```
MatchStmt l0_parser.Parser._parse_match_stmt (
              self )  [protected]
```

Parse a 'match' statement.

7.138.3.42 _parse_mul_expr()

```
Expr l0_parser.Parser._parse_mul_expr (
              self )  [protected]
```

Parse a multiplication/division/modulo expression.

7.138.3.43 _parse_or_expr()

```
Expr l0_parser.Parser._parse_or_expr (
              self )  [protected]
```

Parse a logical OR expression.

7.138.3.44 _parse_pattern()

```
Pattern l0_parser.Parser._parse_pattern (
              self )  [protected]
```

Parse a pattern for 'match' arms.

7.138.3.45 _parse_postfix_expr()

```
Expr l0_parser.Parser._parse_postfix_expr (
              self )  [protected]
```

Parse a postfix expression (calls, indexing, field access, try).

7.138.3.46 _parse_primary_expr()

```
Expr l0_parser.Parser._parse_primary_expr (
            self )  [protected]
```

Parse a primary expression (literals, identifiers, parenthesized expressions).

7.138.3.47 _parse_rel_expr()

```
Expr l0_parser.Parser._parse_rel_expr (
            self )  [protected]
```

Parse a relational expression.

7.138.3.48 _parse_return_stmt()

```
ReturnStmt l0_parser.Parser._parse_return_stmt (
            self )  [protected]
```

Parse a 'return' statement.

7.138.3.49 _parse_simple_stmt()

```
Stmt l0_parser.Parser._parse_simple_stmt (
            self )  [protected]
```

Parse a simple statement (let, break, return, assign, or expr).

7.138.3.50 _parse_stmt()

```
Stmt l0_parser.Parser._parse_stmt (
            self )  [protected]
```

Parse a single statement.

7.138.3.51 _parse_struct()

```
StructDecl l0_parser.Parser._parse_struct (
            self )  [protected]
```

Parse a struct definition.

7.138.3.52 _parse_top_level_decl()

```
TopLevelDecl l0_parser.Parser._parse_top_level_decl (
            self )  [protected]
```

Parse a single top-level declaration.

7.138.3.53 _parse_top_level_let()

```
LetDecl l0_parser.Parser._parse_top_level_let (
              self )  [protected]
```

Parse a top-level 'let' binding.

7.138.3.54 _parse_type()

```
TypeRef l0_parser.Parser._parse_type (
              self )  [protected]
```

Parse a type reference including pointer and nullability suffixes.

7.138.3.55 _parse_type_alias()

```
TypeAliasDecl l0_parser.Parser._parse_type_alias (
              self )  [protected]
```

Parse a type alias declaration.

7.138.3.56 _parse_unary_expr()

```
Expr l0_parser.Parser._parse_unary_expr (
              self )  [protected]
```

Parse a unary expression.

7.138.3.57 _parse_while_stmt()

```
WhileStmt l0_parser.Parser._parse_while_stmt (
              self )  [protected]
```

Parse a 'while' loop statement.

7.138.3.58 _parse_with_item()

```
WithItem l0_parser.Parser._parse_with_item (
              self )  [protected]
```

Parse a single item in a 'with' statement.

7.138.3.59 _parse_with_stmt()

```
WithStmt l0_parser.Parser._parse_with_stmt (
              self )  [protected]
```

Parse a 'with' statement for resource management.

7.138.3.60 _peek()

```
Token l0_parser.Parser._peek (
              self )  [protected]
```

Return the current token without advancing.

Lexer-error wrappers are interpreted as logical tokens. A wrapper with recovery is exposed as that recovered token; a wrapper without recovery is skipped. Deferred diagnostics are emitted once per physical token.

7.138.3.61 _skip_lexer_errors()

```
None l0_parser.Parser._skip_lexer_errors (
              self )  [protected]
```

Emit and step over unrecoverable LEXER_ERROR tokens at the cursor.

7.138.3.62 _span_start()

```
Span l0_parser.Parser._span_start (
              self )  [protected]
```

Create a zero-length span starting at the current token.

7.138.3.63 _sync_case_invalid_arm()

```
None l0_parser.Parser._sync_case_invalid_arm (
              self )  [protected]
```

Skip a malformed or forbidden case arm without leaving the case.

7.138.3.64 _sync_stmt()

```
None l0_parser.Parser._sync_stmt (
              self )  [protected]
```

Skip tokens until we reach a statement boundary or end of block.

7.138.3.65 _sync_top_level()

```
None l0_parser.Parser._sync_top_level (
              self )  [protected]
```

Skip tokens until we find the start of a new top-level declaration or EOF.

7.138.3.66 _try_parse_qualified_name()

```
Optional[tuple[list[str], Optional[list[str]], Token]] l0_parser.Parser._try_parse_qualified↩
_name (
              self )  [protected]
```

Try to parse a qualified name (e.g., mod::name).

Returns

A tuple (module_path, qualifier, name_token) if successful, else None.

7.138.3.67 from_source()

```
"Parser" l0_parser.Parser.from_source (
              cls,
             str source )
```

Create a parser from a source string.

Parameters

source	The source code text.

Returns

A new Parser instance populated with tokens.

7.138.3.68 parse_module()

```
Module l0_parser.Parser.parse_module (
              self,
             Optional[str]  filename = None )
```

Parse a complete L0 module.

Parameters

filename	Optional source filename for diagnostics.

Returns

A Module AST node.

7.138.4 Member Data Documentation

7.138.4.1 _last_token

```
l0_parser.Parser._last_token  [protected]
```

7.138.4.2 _RESERVED_BINARY_OPS

dict l0_parser.Parser._RESERVED_BINARY_OPS [static], [protected]

Initial value:

```
= {
        TokenKind.AMP: "'&' (bitwise AND)",
        TokenKind.PIPE: "'|' (bitwise OR)",
        TokenKind.CARET: "'^' (bitwise XOR)",
        TokenKind.LSHIFT: "'«' (left shift)",
        TokenKind.RSHIFT: "'»' (right shift)",
    }
```

7.138.4.3 diagnostics

l0_parser.Parser.diagnostics

7.138.4.4 eof_aborted

l0_parser.Parser.eof_aborted

7.138.4.5 filename

l0_parser.Parser.filename

7.138.4.6 index

l0_parser.Parser.index

7.138.4.7 tokens

l0_parser.Parser.tokens

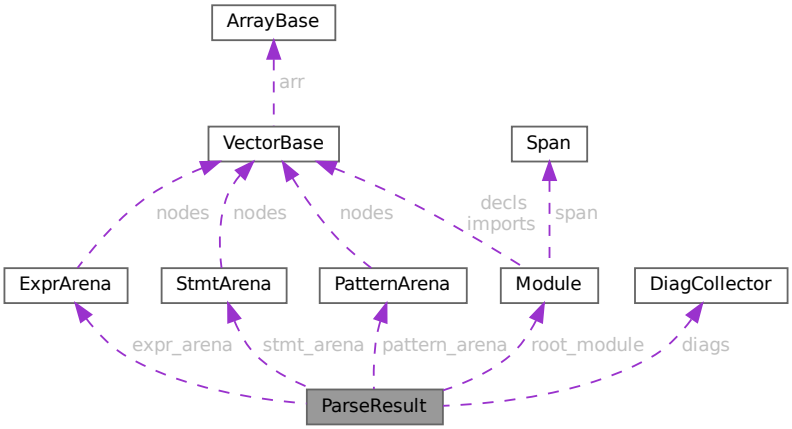
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_parser.py

7.139 ParseResult Struct Reference

Represents the ParseResult structure.

Collaboration diagram for ParseResult:



Public Attributes

- root_module: Module∗?
- expr_arena: ExprArena∗
- stmt_arena: StmtArena∗
- pattern_arena: PatternArena∗
- diags: DiagCollector∗

7.139.1 Detailed Description

Represents the ParseResult structure.

7.139.2 Member Data Documentation

7.139.2.1 diags

diags: `diags: DiagCollector*`

7.139.2.2 expr_arena

expr_arena: `expr_arena: ExprArena*`

7.139.2.3 pattern_arena

pattern_arena: `pattern_arena: PatternArena*`

7.139.2.4 root_module

root_module: `root_module: Module*?`

7.139.2.5 stmt_arena

stmt_arena: `stmt_arena: StmtArena*`

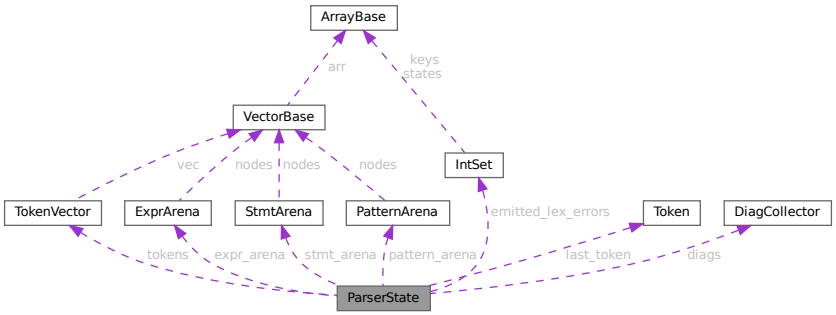
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/parser.l0

7.140 ParserState Struct Reference

Represents the ParserState structure.

Collaboration diagram for ParserState:



Public Attributes

- tokens: TokenVector
- index: int
- last_token: Token
- filename: string
- diags: DiagCollector*
- expr_arena: ExprArena*
- stmt_arena: StmtArena*
- pattern_arena: PatternArena*
- eof_aborted: bool
- emitted_lex_errors: IntSet*

7.140.1 Detailed Description

Represents the ParserState structure.

7.140.2 Member Data Documentation

7.140.2.1 diags

diags: `diags: DiagCollector*`

7.140.2.2 emitted_lex_errors

emitted_lex_errors: `emitted_lex_errors: IntSet*`

7.140.2.3 eof_aborted

eof_aborted: `eof_aborted: bool`

7.140.2.4 expr_arena

expr_arena: `expr_arena: ExprArena*`

7.140.2.5 filename

filename: `filename: string`

7.140.2.6 index

index: `index: int`

7.140.2.7 last_token

last_token: `last_token: Token`

7.140.2.8 pattern_arena

pattern_arena: `pattern_arena: PatternArena*`

7.140.2.9 stmt_arena

stmt_arena: `stmt_arena: StmtArena*`

7.140.2.10 tokens

tokens: `tokens: TokenVector`

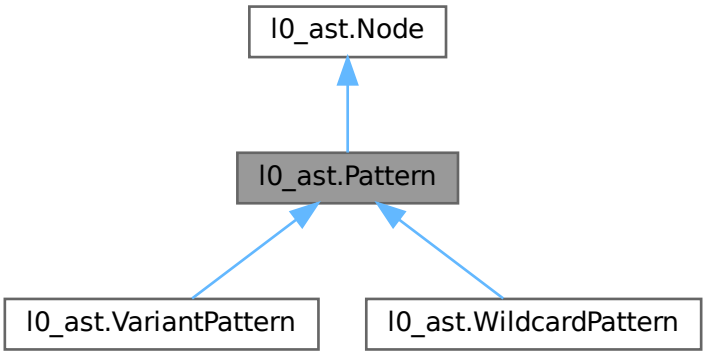
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/parser/shared.l0

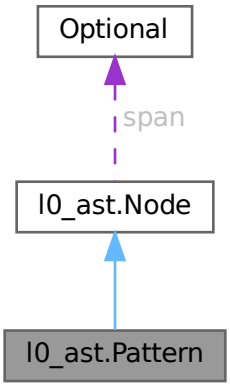
7.141 l0_ast.Pattern Class Reference

Base class for match patterns.

Inheritance diagram for l0_ast.Pattern:



Collaboration diagram for l0_ast.Pattern:



Additional Inherited Members

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.141.1 Detailed Description

Base class for match patterns.

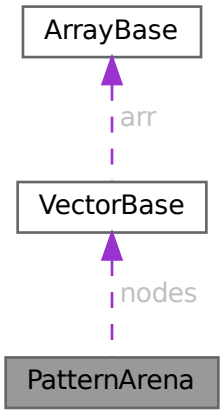
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.142 PatternArena Struct Reference

Represents the PatternArena structure.

Collaboration diagram for PatternArena:



Public Attributes

- nodes: VectorBase*

7.142.1 Detailed Description

Represents the PatternArena structure.

7.142.2 Member Data Documentation

7.142.2.1 nodes

nodes: `nodes: VectorBase*`

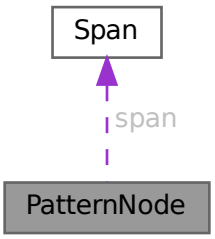
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

7.143 PatternNode Struct Reference

Represents the PatternNode structure.

Collaboration diagram for PatternNode:



Public Attributes

- kind: PatternKind
- span: Span
- name: string
- vars: StringVector*?
- module_path: StringVector*?
- name_qualifier: StringVector*?

7.143.1 Detailed Description

Represents the PatternNode structure.

7.143.2 Member Data Documentation

7.143.2.1 kind

kind: `kind: PatternKind`

7.143.2.2 module_path

module_path: `module_path: StringVector*?`

7.143.2.3 name

name: `name: string`

7.143.2.4 name_qualifier

name_qualifier: `name_qualifier: StringVector*?`

7.143.2.5 span

span: `span: Span`

7.143.2.6 vars

vars: `vars: StringVector*?`

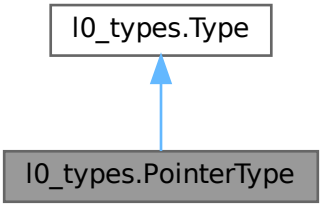
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

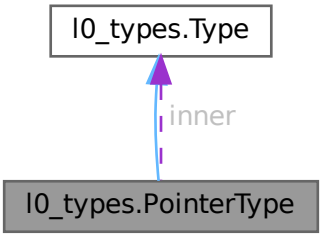
7.144 l0_types.PointerType Class Reference

A pointer to another type (T*).

Inheritance diagram for l0_types.PointerType:



Collaboration diagram for l0_types.PointerType:



Static Public Attributes

- Type inner

7.144.1 Detailed Description

A pointer to another type (T∗).

Attributes

- `inner`: The Type being pointed to.

7.144.2 Member Data Documentation

7.144.2.1 inner

```
Type l0_types.PointerType.inner  [static]
```

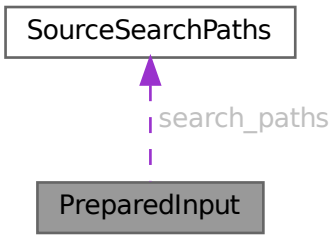
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_types.py

7.145 PreparedInput Struct Reference

Represents the PreparedInput structure.

Collaboration diagram for PreparedInput:



Public Attributes

- entry_module: string
- search_paths: SourceSearchPaths∗

7.145.1 Detailed Description

Represents the PreparedInput structure.

7.145.2 Member Data Documentation

7.145.2.1 entry_module

entry_module: `entry_module: string`

7.145.2.2 search_paths

search_paths: `search_paths: SourceSearchPaths∗`

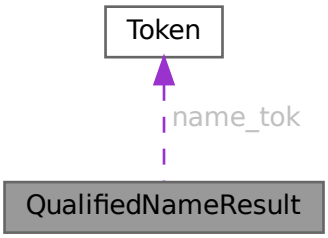
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/l0c_lib.l0

7.146 QualifiedNameResult Struct Reference

Represents the QualifiedNameResult structure.

Collaboration diagram for QualifiedNameResult:



Public Attributes

- module_path: StringVector*
- name_qualifier: StringVector*?
- name_tok: Token

7.146.1 Detailed Description

Represents the QualifiedNameResult structure.

7.146.2 Member Data Documentation

7.146.2.1 module_path

module_path: `module_path: StringVector*`

7.146.2.2 name_qualifier

name_qualifier: `name_qualifier: StringVector*?`

7.146.2.3 name_tok

name_tok: `name_tok: Token`

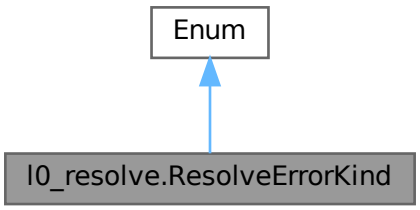
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/parser/shared.l0

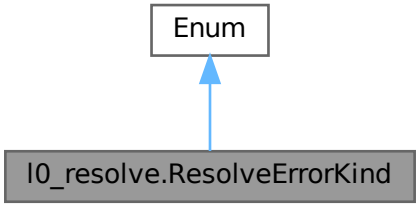
7.147 l0_resolve.ResolveErrorKind Class Reference

Kinds of errors that can occur during symbol resolution.

Inheritance diagram for l0_resolve.ResolveErrorKind:



Collaboration diagram for l0_resolve.ResolveErrorKind:



Static Public Attributes

- UNKNOWN_MODULE = auto()
- MODULE_NOT_IMPORTED = auto()
- UNKNOWN_SYMBOL = auto()
- AMBIGUOUS_SYMBOL = auto()

7.147.1 Detailed Description

Kinds of errors that can occur during symbol resolution.

7.147.2 Member Data Documentation

7.147.2.1 AMBIGUOUS_SYMBOL

```
l0_resolve.ResolveErrorKind.AMBIGUOUS_SYMBOL = auto()  [static]
```

7.147.2.2 MODULE_NOT_IMPORTED

```
l0_resolve.ResolveErrorKind.MODULE_NOT_IMPORTED = auto()  [static]
```

7.147.2.3 UNKNOWN_MODULE

```
l0_resolve.ResolveErrorKind.UNKNOWN_MODULE = auto()  [static]
```

7.147.2.4 UNKNOWN_SYMBOL

```
l0_resolve.ResolveErrorKind.UNKNOWN_SYMBOL = auto()  [static]
```

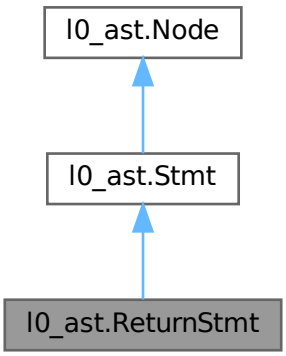
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_resolve.py

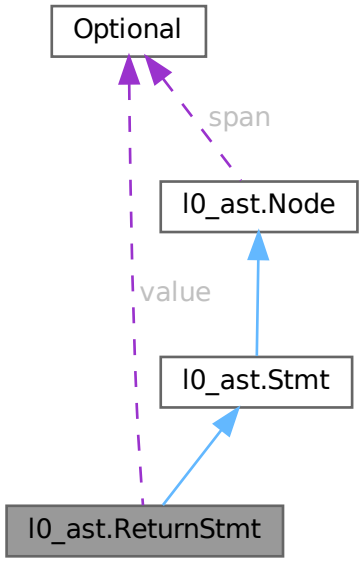
7.148 l0_ast.ReturnStmt Class Reference

A return statement.

Inheritance diagram for l0_ast.ReturnStmt:



Collaboration diagram for l0_ast.ReturnStmt:



Static Public Attributes

- Optional value ["Expr"]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.148.1 Detailed Description

A return statement.

Attributes

- `value`: Optional expression to return.

7.148.2 Member Data Documentation

7.148.2.1 value

```
Optional l0_ast.ReturnStmt.value ["Expr"]  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.149 RtFileInfo Struct Reference

Represents basic file metadata used by bootstrap tooling.

Public Attributes

- exists: bool
- is_file: bool
- is_dir: bool
- size: int?
- mtime_sec: int?
- mtime_nsec: int?

7.149.1 Detailed Description

Represents basic file metadata used by bootstrap tooling.

7.149.2 Member Data Documentation

7.149.2.1 exists

exists: `exists: bool`

7.149.2.2 is_dir

is_dir: `is_dir: bool`

7.149.2.3 is_file

is_file: `is_file: bool`

7.149.2.4 mtime_nsec

mtime_nsec: `mtime_nsec: int?`

7.149.2.5 mtime_sec

mtime_sec: `mtime_sec: int?`

7.149.2.6 size

size: `size: int?`

The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/sys/rt.l0

7.150 RtTimeParts Struct Reference

Represents components of wall or monotonic time.

Public Attributes

- sec: int
- nsec: int

7.150.1 Detailed Description

Represents components of wall or monotonic time.

7.150.2 Member Data Documentation

7.150.2.1 nsec

nsec: `nsec: int`

7.150.2.2 sec

sec: `sec: int`

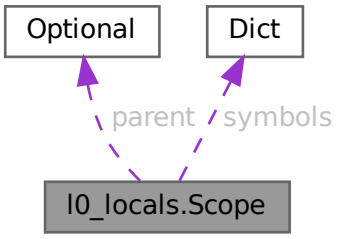
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/sys/rt.l0

7.151 l0_locals.Scope Class Reference

A lexical scope for local bindings.

Collaboration diagram for l0_locals.Scope:



Public Member Functions

- Optional[LocalSymbol] lookup (self, str name)

  Look up a name in this scope and all enclosing scopes.

Static Public Attributes

- Optional parent ["Scope"]
- Dict symbols = field(default_factory=dict)

7.151.1 Detailed Description

A lexical scope for local bindings.

Scopes form a tree structure via parent pointers, mirroring the nested structure of the source code.

Attributes

- `parent`: The enclosing scope, or None if this is a root scope.
- `symbols`: Mapping of names to LocalSymbol objects in this scope.

7.151.2 Member Function Documentation

7.151.2.1 lookup()

```
Optional[LocalSymbol] l0_locals.Scope.lookup (
             self,
            str name )
```

Look up a name in this scope and all enclosing scopes.

Parameters

name	The name to look up.

Returns

The LocalSymbol if found, otherwise None.

7.151.3 Member Data Documentation

7.151.3.1 parent

```
Optional l0_locals.Scope.parent ["Scope"]  [static]
```

7.151.3.2 symbols

```
Dict l0_locals.Scope.symbols = field(default_factory=dict)  [static]
```

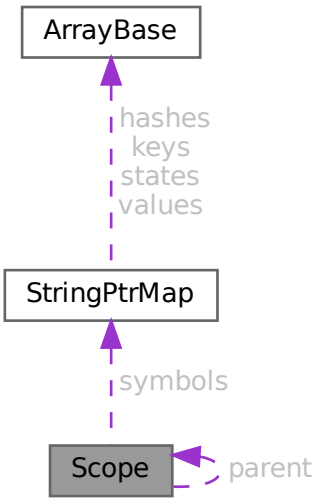
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_locals.py

7.152 Scope Struct Reference

Represents the Scope structure.

Collaboration diagram for Scope:



Public Attributes

- parent: Scope∗?
- symbols: StringPtrMap∗

7.152.1 Detailed Description

Represents the Scope structure.

7.152.2 Member Data Documentation

7.152.2.1 parent

parent: `parent: Scope∗?`

7.152.2.2 symbols

symbols: `symbols: StringPtrMap∗`

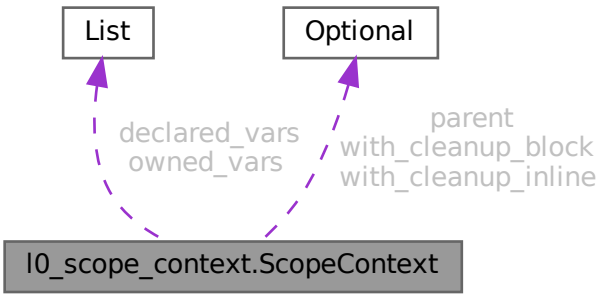
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/locals.l0

7.153 l0_scope_context.ScopeContext Class Reference

Tracks variables and cleanup requirements for a single lexical scope.

Collaboration diagram for l0_scope_context.ScopeContext:



Public Member Functions

- None add_owned (self, str var_name, Type var_type)

  Mark a variable as owned (requiring cleanup) and declared.
- None add_declared (self, str var_name, Type var_type)

  Mark a variable as declared for type lookup only.
- None remove_owned (self, str var_name)

  Remove a variable from the owned set.

Public Attributes

- owned_vars

Static Public Attributes

- List owned_vars = field(default_factory=list)
- List declared_vars = field(default_factory=list)
- Optional parent = None
- Optional with_cleanup_inline = None
- Optional with_cleanup_block = None
- bool with_cleanup_in_progress = False

7.153.1 Detailed Description

Tracks variables and cleanup requirements for a single lexical scope.

Attributes

- `owned_vars`: Variables that must be cleaned up when the scope is exited.
- `declared_vars`: All variables visible in this scope, used for type lookups.
- `parent`: The enclosing scope context, if any.
- `with_cleanup_inline`: List of cleanup statements for inline 'with' items.
- `with_cleanup_block`: Optional block for 'with ... cleanup { ... }'.
- `with_cleanup_in_progress`: Reentrancy guard for with-cleanup emission.

7.153.2 Member Function Documentation

7.153.2.1 add_declared()

```
None l0_scope_context.ScopeContext.add_declared (
              self,
             str var_name,
             Type var_type )
```

Mark a variable as declared for type lookup only.

Used for function parameters and pattern variables, which are managed by the caller or scrutinee and do not require cleanup within the current scope.

Parameters

var_name	The (mangled) name of the variable.
var_type	The L0 Type of the variable.

7.153.2.2 add_owned()

```
None l0_scope_context.ScopeContext.add_owned (
              self,
             str var_name,
             Type var_type )
```

Mark a variable as owned (requiring cleanup) and declared.

Parameters

var_name	The (mangled) name of the variable.
var_type	The L0 Type of the variable.

7.153.2.3 remove_owned()

```
None l0_scope_context.ScopeContext.remove_owned (
              self,
             str var_name )
```

Remove a variable from the owned set.

Parameters

var_name	The (mangled) name of the variable to stop tracking.

7.153.3 Member Data Documentation

7.153.3.1 declared_vars

```
List l0_scope_context.ScopeContext.declared_vars = field(default_factory=list)  [static]
```

7.153.3.2 owned_vars [1/2]

```
List l0_scope_context.ScopeContext.owned_vars = field(default_factory=list)  [static]
```

7.153.3.3 owned_vars [2/2]

```
l0_scope_context.ScopeContext.owned_vars
```

7.153.3.4 parent

```
Optional l0_scope_context.ScopeContext.parent = None  [static]
```

7.153.3.5 with_cleanup_block

```
Optional l0_scope_context.ScopeContext.with_cleanup_block = None  [static]
```

7.153.3.6 with_cleanup_in_progress

```
bool l0_scope_context.ScopeContext.with_cleanup_in_progress = False  [static]
```

7.153.3.7 with_cleanup_inline

```
Optional l0_scope_context.ScopeContext.with_cleanup_inline = None  [static]
```

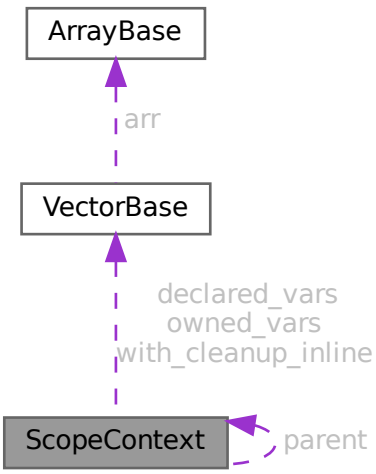
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_scope_context.py

7.154 ScopeContext Struct Reference

Represents backend cleanup state for a single lexical scope.

Collaboration diagram for ScopeContext:



Public Attributes

- scope_id: int
- parent: ScopeContext*?
- owned_vars: VectorBase*
- declared_vars: VectorBase*
- with_cleanup_inline: VectorBase*?
- with_cleanup_block: StmtId
- with_cleanup_in_progress: bool

7.154.1 Detailed Description

Represents backend cleanup state for a single lexical scope.

7.154.2 Member Data Documentation

7.154.2.1 declared_vars

declared_vars: `declared_vars: VectorBase*`

7.154.2.2 owned_vars

owned_vars: `owned_vars: VectorBase*`

7.154.2.3 parent

parent: `parent: ScopeContext*?`

7.154.2.4 scope_id

scope_id: `scope_id: int`

7.154.2.5 with_cleanup_block

with_cleanup_block: `with_cleanup_block: StmtId`

7.154.2.6 with_cleanup_in_progress

with_cleanup_in_progress: `with_cleanup_in_progress: bool`

7.154.2.7 with_cleanup_inline

with_cleanup_inline: `with_cleanup_inline: VectorBase*?`

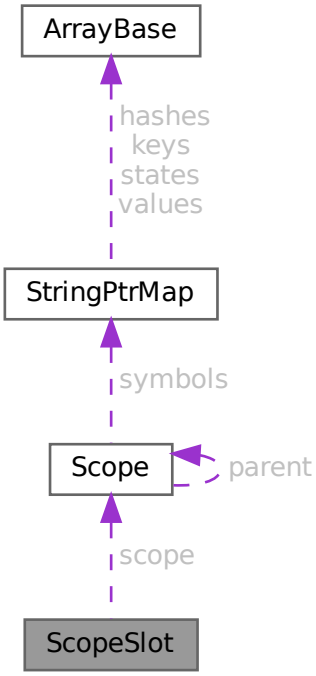
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/scope_context.l0

7.155 ScopeSlot Struct Reference

Represents the ScopeSlot structure.

Collaboration diagram for ScopeSlot:



Public Attributes

- scope: Scope*?

7.155.1 Detailed Description

Represents the ScopeSlot structure.

7.155.2 Member Data Documentation

7.155.2.1 scope

scope: `scope: Scope*?`

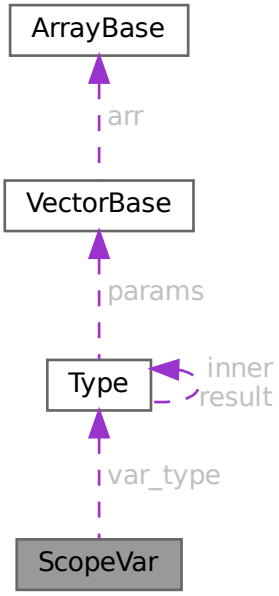
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/locals.l0

7.156 ScopeVar Struct Reference

Represents one declared variable tracked by the backend.

Collaboration diagram for ScopeVar:



Public Attributes

- name: string
- var_type: Type*

7.156.1 Detailed Description

Represents one declared variable tracked by the backend.

7.156.2 Member Data Documentation

7.156.2.1 name

name: `name: string`

7.156.2.2 var_type

var_type: `var_type: Type*`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/scope_context.l0

7.157 l0_signatures.SignatureResolver Class Reference

Resolves top-level type signatures across all modules.

Public Member Functions

- __init__ (self, CompilationUnit cu, Dict[str, ModuleEnv] module_envs)

  Initialize the signature resolver.
- None resolve (self)

  Resolve all top-level signatures in the compilation unit.

Public Attributes

- cu
- module_envs

Protected Member Functions

- None _resolve_module_signatures (self, ModuleEnv env)

  Resolve all declarations within a single module.
- None _emit (self, Diagnostic diag)

  Add a diagnostic to the collection.
- Optional[Type] _resolve_type_ref (self, ModuleEnv env, TypeRef tref, Optional[Set[Tuple[str, str]]] alias_↩ stack=None)

  Resolve a TypeRef in the context of a module environment.
- Optional[Type] _resolve_type_alias_symbol (self, ModuleEnv env, Symbol sym, Optional[Set[Tuple[str, str]]] alias_stack=None)

  Resolve a TYPE_ALIAS symbol to its target Type.
- None _resolve_struct (self, ModuleEnv env, StructDecl decl)

  Resolve field types for a struct declaration.
- None _resolve_enum (self, ModuleEnv env, EnumDecl decl)

  Resolve variant payload types for an enum declaration.

- None _resolve_func (self, ModuleEnv env, FuncDecl decl)

  Resolve parameter and return types for a function declaration.
- None _resolve_type_alias (self, ModuleEnv env, TypeAliasDecl decl)

  Resolve the target type for a type alias.
- None _resolve_let (self, ModuleEnv env, LetDecl decl)

  Resolve or infer the type for a module-level 'let' binding.
- Optional[Type] _infer_literal_type (self, ModuleEnv env, Expr expr)

  Infer type from simple literal expressions and struct/enum construction.
- bool _symbol_is_zero_arg_enum_variant (self, Symbol sym)

  Return whether a symbol names a zero-argument enum variant.
- Optional[EnumType] _enum_type_for_variant_symbol (self, Symbol sym)

  Return the owning enum type for an enum variant symbol.
- Set[Tuple[str, str]] _extract_value_type_dependencies (self, Type typ)

  Extract type dependencies for value fields only (pointer-free).
- None _detect_value_type_cycles (self)

  Detect and report cycles in value-type definitions.

7.157.1 Detailed Description

Resolves top-level type signatures across all modules.

The resolver processes:

- Function signatures (parameters and return types).
- Struct field types.
- Enum variant payload types.
- Type alias targets.
- Module-level constant (let) types.

It also performs cycle detection for value-type definitions.

Attributes

- `cu`: The compilation unit being processed.
- `module_envs`: Mapping of module names to their environments.
- `diagnostics`: Collected list of resolution diagnostics.
- `func_types`: Resolved function signatures keyed by (module, name).
- `struct_infos`: Resolved struct definitions keyed by (module, name).
- `enum_infos`: Resolved enum definitions keyed by (module, name).
- `let_types`: Resolved constant types keyed by (module, name).

7.157.2 Constructor & Destructor Documentation

7.157.2.1 __init__()

```
10_signatures.SignatureResolver.__init__ (
              self,
             CompilationUnit cu,
             Dict[str, ModuleEnv] module_envs )
```

Initialize the signature resolver.

Parameters

cu	The compilation unit.
module_envs	Environments produced by name resolution.

7.157.3 Member Function Documentation

7.157.3.1 _detect_value_type_cycles()

```
None l0_signatures.SignatureResolver._detect_value_type_cycles (
              self )  [protected]
```

Detect and report cycles in value-type definitions.

Value-type cycles (where A contains B and B contains A directly) create infinite-size types and are prohibited. Pointers must be used to break such cycles.

7.157.3.2 _emit()

```
None l0_signatures.SignatureResolver._emit (
              self,
             Diagnostic diag )  [protected]
```

Add a diagnostic to the collection.

7.157.3.3 _enum_type_for_variant_symbol()

```
Optional[EnumType] l0_signatures.SignatureResolver._enum_type_for_variant_symbol (
              self,
             Symbol sym )  [protected]
```

Return the owning enum type for an enum variant symbol.

7.157.3.4 _extract_value_type_dependencies()

```
Set[Tuple[str, str]] l0_signatures.SignatureResolver._extract_value_type_dependencies (
              self,
             Type typ )  [protected]
```

Extract type dependencies for value fields only (pointer-free).

7.157.3.5 _infer_literal_type()

```
Optional[Type] l0_signatures.SignatureResolver._infer_literal_type (
              self,
             ModuleEnv env,
             Expr expr )  [protected]
```

Infer type from simple literal expressions and struct/enum construction.

7.157.3.6 _resolve_enum()

```
None l0_signatures.SignatureResolver._resolve_enum (
              self,
             ModuleEnv env,
             EnumDecl decl )  [protected]
```

Resolve variant payload types for an enum declaration.

7.157.3.7 _resolve_func()

```
None l0_signatures.SignatureResolver._resolve_func (
              self,
             ModuleEnv env,
             FuncDecl decl )  [protected]
```

Resolve parameter and return types for a function declaration.

7.157.3.8 _resolve_let()

```
None l0_signatures.SignatureResolver._resolve_let (
              self,
             ModuleEnv env,
             LetDecl decl )  [protected]
```

Resolve or infer the type for a module-level 'let' binding.

7.157.3.9 _resolve_module_signatures()

```
None l0_signatures.SignatureResolver._resolve_module_signatures (
              self,
             ModuleEnv env )  [protected]
```

Resolve all declarations within a single module.

7.157.3.10 _resolve_struct()

```
None l0_signatures.SignatureResolver._resolve_struct (
              self,
             ModuleEnv env,
             StructDecl decl )  [protected]
```

Resolve field types for a struct declaration.

7.157.3.11 _resolve_type_alias()

```
None l0_signatures.SignatureResolver._resolve_type_alias (
              self,
             ModuleEnv env,
             TypeAliasDecl decl )  [protected]
```

Resolve the target type for a type alias.

7.157.3.12 _resolve_type_alias_symbol()

```
Optional[Type] l0_signatures.SignatureResolver._resolve_type_alias_symbol (
              self,
             ModuleEnv env,
             Symbol sym,
             Optional[Set[Tuple[str, str]]]  alias_stack = None )  [protected]
```

Resolve a TYPE_ALIAS symbol to its target Type.

Caches the result in `sym.type`. Detects recursive alias cycles.

Parameters

env	Current module environment.
sym	The type alias symbol to resolve.
alias_stack	Set of alias identifiers currently being resolved.

Returns

The target Type if successful, otherwise None.

7.157.3.13 _resolve_type_ref()

```
Optional[Type] l0_signatures.SignatureResolver._resolve_type_ref (
              self,
             ModuleEnv env,
             TypeRef tref,
             Optional[Set[Tuple[str, str]]]  alias_stack = None )  [protected]
```

Resolve a TypeRef in the context of a module environment.

Parameters

env	The current module environment.
tref	The TypeRef AST node to resolve.
alias_stack	Stack of aliases being resolved (for cycle detection).

Returns

The resolved Type if successful, otherwise None.

7.157.3.14 _symbol_is_zero_arg_enum_variant()

```
bool l0_signatures.SignatureResolver._symbol_is_zero_arg_enum_variant (
              self,
             Symbol sym )  [protected]
```

Return whether a symbol names a zero-argument enum variant.

7.157.3.15 resolve()

```
None l0_signatures.SignatureResolver.resolve (
            self )
```

Resolve all top-level signatures in the compilation unit.

This iterates through all modules and their declarations, populating the resolution tables and reporting any errors found.

7.157.4 Member Data Documentation

7.157.4.1 cu

```
l0_signatures.SignatureResolver.cu
```

7.157.4.2 module_envs

```
l0_signatures.SignatureResolver.module_envs
```

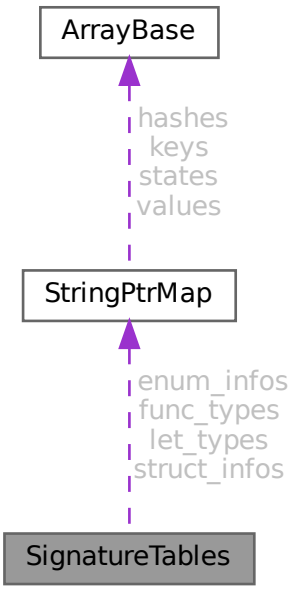
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_signatures.py

7.158 SignatureTables Struct Reference

Represents the SignatureTables structure.

Collaboration diagram for SignatureTables:



Public Attributes

- func_types: StringPtrMap*
- struct_infos: StringPtrMap*
- enum_infos: StringPtrMap*
- let_types: StringPtrMap*

7.158.1 Detailed Description

Represents the SignatureTables structure.

7.158.2 Member Data Documentation

7.158.2.1 enum_infos

enum_infos: `enum_infos: StringPtrMap*`

7.158.2.2 func_types

func_types: `func_types: StringPtrMap*`

7.158.2.3 let_types

let_types: `let_types: StringPtrMap*`

7.158.2.4 struct_infos

struct_infos: `struct_infos: StringPtrMap*`

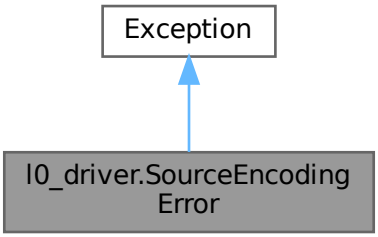
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/types.l0

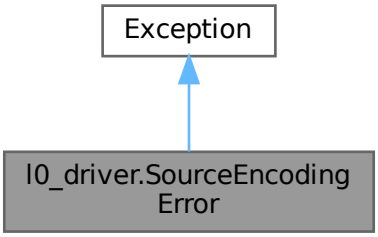
7.159 l0_driver.SourceEncodingError Class Reference

Raised when a source file cannot be decoded as UTF-8.

Inheritance diagram for l0_driver.SourceEncodingError:



Collaboration diagram for l0_driver.SourceEncodingError:



Public Member Functions

- __init__ (self, str|Path path, str message)

  Initialize a UTF-8 decoding failure.

Public Attributes

- path
- message

7.159.1 Detailed Description

Raised when a source file cannot be decoded as UTF-8.

7.159.2 Constructor & Destructor Documentation

7.159.2.1 __init__()

```
l0_driver.SourceEncodingError.__init__ (
              self,
             str | Path path,
             str message )
```

Initialize a UTF-8 decoding failure.

Parameters

path	Path to the source file that failed to decode.
message	Human-readable explanation of the decoding failure.

See also

`load_source_utf8`: Raises this exception for invalid source encodings.

7.159.3 Member Data Documentation

7.159.3.1 message

```
l0_driver.SourceEncodingError.message
```

7.159.3.2 path

```
l0_driver.SourceEncodingError.path
```

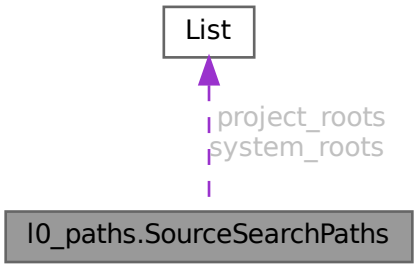
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_driver.py

7.160 l0_paths.SourceSearchPaths Class Reference

Search configuration for L0 modules.

Collaboration diagram for l0_paths.SourceSearchPaths:



Public Member Functions

- None add_system_root (self, str|Path root)

  Add a path to the system search roots.
- None add_project_root (self, str|Path root)

  Add a path to the project search roots.
- Path module_relpath (self, str module_name)

  Convert a dotted module name to its relative file path.
- Path resolve (self, str module_name)

  Find the absolute path for an L0 module name.

Static Public Attributes

- List system_roots = field(default_factory=list)
- List project_roots = field(default_factory=list)

7.160.1 Detailed Description

Search configuration for L0 modules.

Resolution order: system_roots are searched first, then project_roots.

Attributes

- `system_roots`: List of paths for standard library and toolchain modules.
- `project_roots`: List of paths for user and project-specific modules.

7.160.2 Member Function Documentation

7.160.2.1 add_project_root()

```
None l0_paths.SourceSearchPaths.add_project_root (
              self,
             str | Path root )
```

Add a path to the project search roots.

Parameters

root	The directory path to add.

7.160.2.2 add_system_root()

```
None l0_paths.SourceSearchPaths.add_system_root (
              self,
             str | Path root )
```

Add a path to the system search roots.

Parameters

root	The directory path to add.

7.160.2.3 module_relpath()

```
Path l0_paths.SourceSearchPaths.module_relpath (
             self,
            str module_name )
```

Convert a dotted module name to its relative file path.

Example: 'std.io' → 'std/io.l0'

Parameters

module_name	The dotted L0 module name.

Returns

The relative Path to the module file.

7.160.2.4 resolve()

```
Path l0_paths.SourceSearchPaths.resolve (
             self,
            str module_name )
```

Find the absolute path for an L0 module name.

Parameters

module_name	The dotted L0 module name.

Returns

The absolute Path to the first matching file found.

Exceptions

FileNotFoundError	If the module cannot be found in any root.

7.160.3 Member Data Documentation

7.160.3.1 project_roots

```
List l0_paths.SourceSearchPaths.project_roots = field(default_factory=list)  [static]
```

7.160.3.2 system_roots

```
List l0_paths.SourceSearchPaths.system_roots = field(default_factory=list)  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_paths.py

7.161 SourceSearchPaths Struct Reference

Represents the SourceSearchPaths structure.

Public Attributes

- system_roots: StringVector*
- project_roots: StringVector*

7.161.1 Detailed Description

Represents the SourceSearchPaths structure.

7.161.2 Member Data Documentation

7.161.2.1 project_roots

project_roots: `project_roots:  StringVector*`

7.161.2.2 system_roots

system_roots: `system_roots:  StringVector*`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/source_paths.l0

7.162 l0_ast.Span Class Reference

Represents a range of text in a source file.

Static Public Attributes

- int start_line
- int start_column
- int end_line
- int end_column

7.162.1 Detailed Description

Represents a range of text in a source file.

Attributes

- `start_line`: 1-based starting line number.
- `start_column`: 1-based starting column number.
- `end_line`: 1-based ending line number.
- `end_column`: 1-based ending column number.

7.162.2 Member Data Documentation

7.162.2.1 end_column

```
int l0_ast.Span.end_column  [static]
```

7.162.2.2 end_line

```
int l0_ast.Span.end_line  [static]
```

7.162.2.3 start_column

```
int l0_ast.Span.start_column  [static]
```

7.162.2.4 start_line

```
int l0_ast.Span.start_line  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.163 Span Struct Reference

Represents the Span structure.

Public Attributes

- start_line: int
- start_column: int
- end_line: int
- end_column: int

7.163.1 Detailed Description

Represents the Span structure.

7.163.2 Member Data Documentation

7.163.2.1 end_column

end_column: `end_column: int`

7.163.2.2 end_line

end_line: `end_line: int`

7.163.2.3 start_column

start_column: `start_column: int`

7.163.2.4 start_line

start_line: `start_line: int`

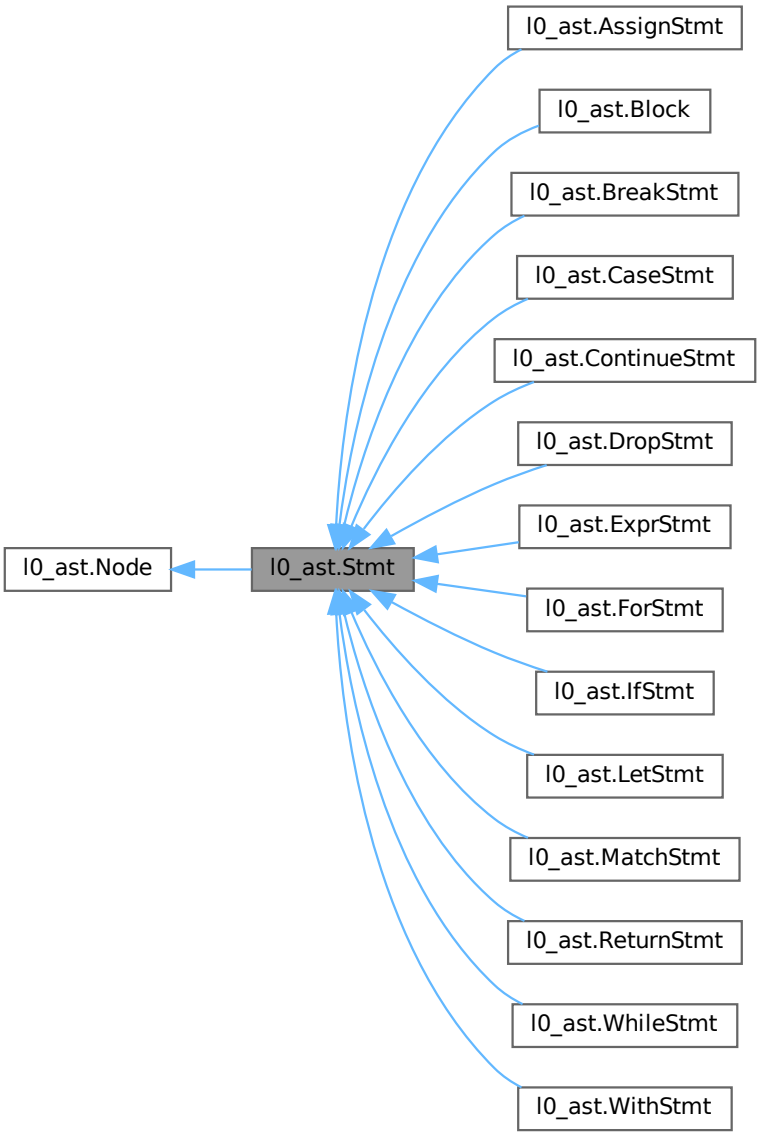
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

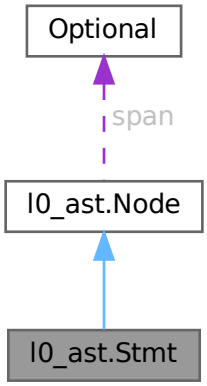
7.164 l0_ast.Stmt Class Reference

Base class for all statements.

Inheritance diagram for l0_ast.Stmt:



Collaboration diagram for l0_ast.Stmt:



Additional Inherited Members

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.164.1 Detailed Description

Base class for all statements.

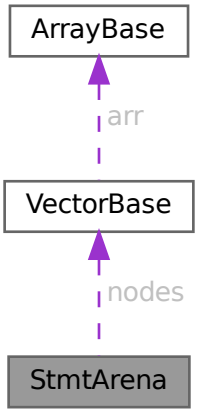
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.165 StmtArena Struct Reference

Represents the StmtArena structure.

Collaboration diagram for StmtArena:



Public Attributes

- nodes: VectorBase*

7.165.1 Detailed Description

Represents the StmtArena structure.

7.165.2 Member Data Documentation

7.165.2.1 nodes

nodes: `nodes: VectorBase*`

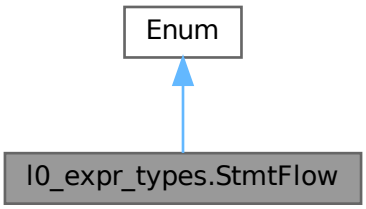
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

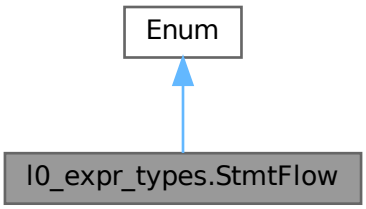
7.166 l0_expr_types.StmtFlow Class Reference

Reachability after checking a statement or block.

Inheritance diagram for l0_expr_types.StmtFlow:



Collaboration diagram for l0_expr_types.StmtFlow:



Static Public Attributes

- str FALLTHROUGH = "fallthrough"
- str RETURNS = "returns"
- str STOPS = "stops"

7.166.1 Detailed Description

Reachability after checking a statement or block.

7.166.2 Member Data Documentation

7.166.2.1 FALLTHROUGH

```
str l0_expr_types.StmtFlow.FALLTHROUGH = "fallthrough"  [static]
```

7.166.2.2 RETURNS

```
str l0_expr_types.StmtFlow.RETURNS = "returns"  [static]
```

7.166.2.3 STOPS

```
str l0_expr_types.StmtFlow.STOPS = "stops"  [static]
```

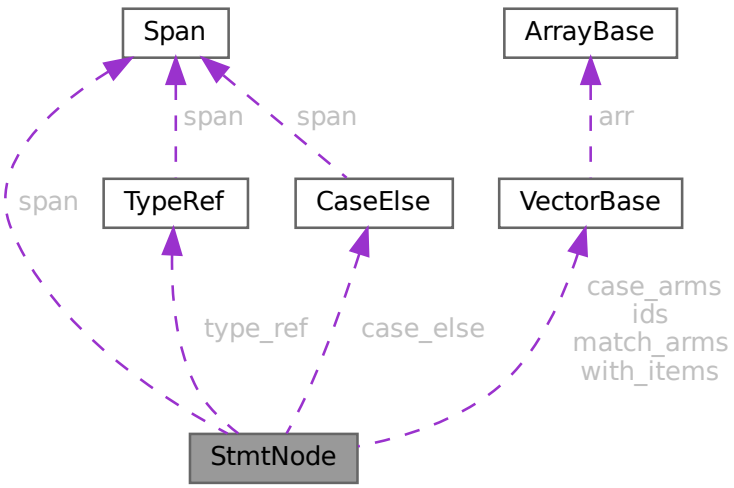
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_expr_types.py

7.167 StmtNode Struct Reference

Represents the StmtNode structure.

Collaboration diagram for StmtNode:



Public Attributes

- kind: StmtKind
- span: Span
- name: string
- type_ref: TypeRef∗?
- expr_a: ExprId
- expr_b: ExprId
- stmt_a: StmtId
- stmt_b: StmtId
- stmt_c: StmtId
- ids: VectorBase∗?
- match_arms: VectorBase∗?
- case_arms: VectorBase∗?
- case_else: CaseElse∗?
- with_items: VectorBase∗?

7.167.1 Detailed Description

Represents the StmtNode structure.

7.167.2 Member Data Documentation

7.167.2.1 case_arms

case_arms: `case_arms: VectorBase*?`

7.167.2.2 case_else

case_else: `case_else: CaseElse*?`

7.167.2.3 expr_a

expr_a: `expr_a: ExprId`

7.167.2.4 expr_b

expr_b: `expr_b: ExprId`

7.167.2.5 ids

ids: `ids: VectorBase*?`

7.167.2.6 kind

kind: `kind: StmtKind`

7.167.2.7 match_arms

match_arms: `match_arms: VectorBase*?`

7.167.2.8 name

name: `name: string`

7.167.2.9 span

span: `span: Span`

7.167.2.10 stmt_a

stmt_a: `stmt_a: StmtId`

7.167.2.11 stmt_b

stmt_b: `stmt_b: StmtId`

7.167.2.12 stmt_c

stmt_c: `stmt_c: StmtId`

7.167.2.13 type_ref

type_ref: `type_ref: TypeRef*?`

7.167.2.14 with_items

with_items: `with_items: VectorBase*?`

The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

7.168 StringBuffer Struct Reference

StringBuffer allows for efficient construction of strings by accumulating parts in a vector.

Public Attributes

- parts: StringVector*
- size: int

7.168.1 Detailed Description

StringBuffer allows for efficient construction of strings by accumulating parts in a vector.

7.168.2 Member Data Documentation

7.168.2.1 parts

parts: `parts: StringVector*`

7.168.2.2 size

size: `size: int`

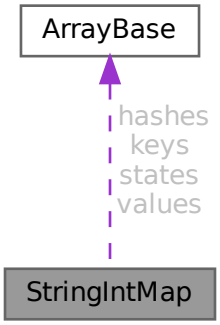
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/text.l0

7.169 StringIntMap Struct Reference

StringIntMap stores associations between strings and integers.

Collaboration diagram for StringIntMap:



Public Attributes

- capacity: int
- count: int
- tomb_count: int
- states: ArrayBase∗
- hashes: ArrayBase∗
- keys: ArrayBase∗
- values: ArrayBase∗

7.169.1 Detailed Description

StringIntMap stores associations between strings and integers.

Same layout and probing as StringPtrMap, but values are int instead of void∗.

7.169.2 Member Data Documentation

7.169.2.1 capacity

capacity: `capacity: int`

7.169.2.2 count

count: `count: int`

7.169.2.3 hashes

hashes: `hashes: ArrayBase*`

7.169.2.4 keys

keys: `keys: ArrayBase*`

7.169.2.5 states

states: `states: ArrayBase*`

7.169.2.6 tomb_count

tomb_count: `tomb_count: int`

7.169.2.7 values

values: `values: ArrayBase*`

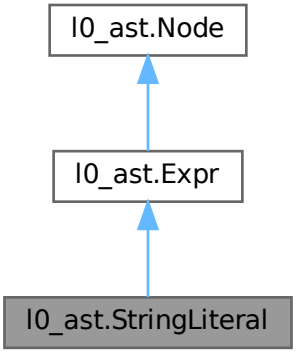
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/hashmap.l0

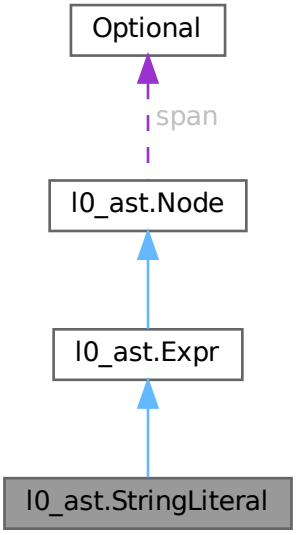
7.170 l0_ast.StringLiteral Class Reference

A string literal expression.

Inheritance diagram for l0_ast.StringLiteral:



Collaboration diagram for l0_ast.StringLiteral:



Static Public Attributes

- str value

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.170.1 Detailed Description

A string literal expression.

Attributes

- `value`: The string value.

7.170.2 Member Data Documentation

7.170.2.1 value

```
str l0_ast.StringLiteral.value  [static]
```

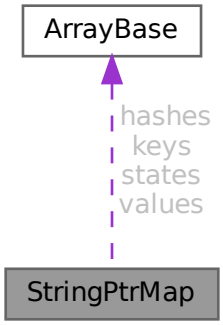
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.171 StringPtrMap Struct Reference

StringPtrMap is an open-addressing hash table with linear probing.

Collaboration diagram for StringPtrMap:



Public Attributes

- capacity: int
- count: int
- tomb_count: int
- states: ArrayBase*
- hashes: ArrayBase*
- keys: ArrayBase*
- values: ArrayBase*

7.171.1 Detailed Description

StringPtrMap is an open-addressing hash table with linear probing.

Keys are ARC-managed strings (manual retain/release). Values are caller-managed void pointers.

Struct-of-arrays layout: states — byte per slot (EMPTY / OCCUPIED / TOMBSTONE) hashes — int per slot (cached hash code) keys — string per slot (ARC-managed) values — void* per slot

7.171.2 Member Data Documentation

7.171.2.1 capacity

capacity: `capacity: int`

7.171.2.2 count

count: `count: int`

7.171.2.3 hashes

hashes: `hashes: ArrayBase*`

7.171.2.4 keys

keys: `keys: ArrayBase*`

7.171.2.5 states

states: `states: ArrayBase*`

7.171.2.6 tomb_count

tomb_count: `tomb_count: int`

7.171.2.7 values

values: `values: ArrayBase*`

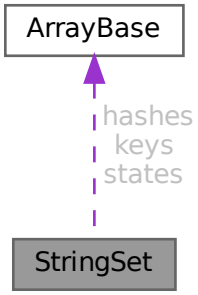
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/hashmap.l0

7.172 StringSet Struct Reference

StringSet is an open-addressing hash set with linear probing.

Collaboration diagram for StringSet:



Public Attributes

- capacity: int
- count: int
- tomb_count: int
- states: ArrayBase*
- hashes: ArrayBase*
- keys: ArrayBase*

7.172.1 Detailed Description

StringSet is an open-addressing hash set with linear probing.

Struct-of-arrays: states, hashes, keys.

7.172.2 Member Data Documentation

7.172.2.1 capacity

capacity: `capacity: int`

7.172.2.2 count

count: `count: int`

7.172.2.3 hashes

hashes: `hashes: ArrayBase*`

7.172.2.4 keys

keys: `keys: ArrayBase*`

7.172.2.5 states

states: `states: ArrayBase*`

7.172.2.6 tomb_count

tomb_count: `tomb_count: int`

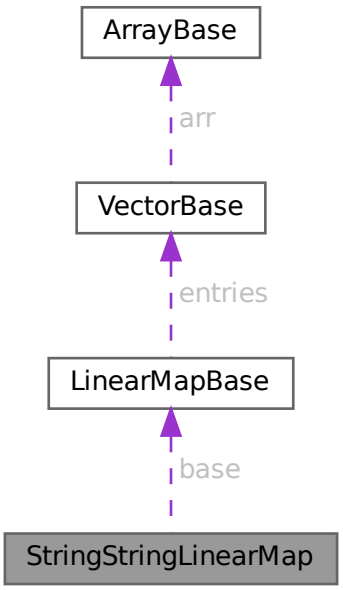
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/hashset.l0

7.173 StringStringLinearMap Struct Reference

A linear map specialized for string keys and string values.

Collaboration diagram for StringStringLinearMap:



Public Attributes

- base: LinearMapBase∗

7.173.1 Detailed Description

A linear map specialized for string keys and string values.

7.173.2 Member Data Documentation

7.173.2.1 base

base: `base: LinearMapBase∗`

The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/linear_map.l0

7.174 StringStringLinearMapEntry Struct Reference

Entry for a string-to-string linear map.

Public Attributes

- key: string
- value: string

7.174.1 Detailed Description

Entry for a string-to-string linear map.

7.174.2 Member Data Documentation

7.174.2.1 key

key: `key: string`

7.174.2.2 value

value: `value: string`

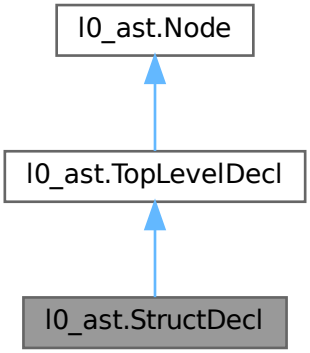
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/linear_map.l0

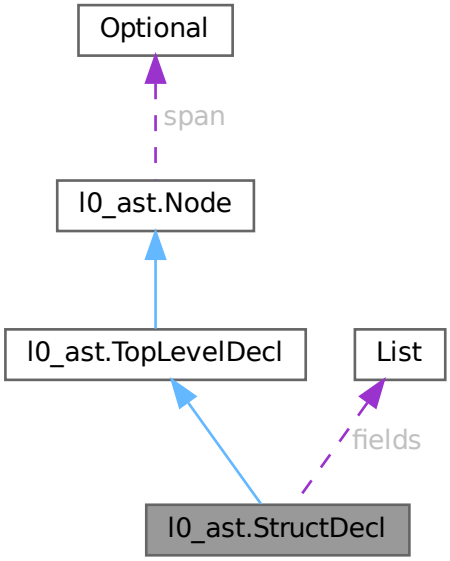
7.175 l0_ast.StructDecl Class Reference

A struct declaration.

Inheritance diagram for l0_ast.StructDecl:



Collaboration diagram for l0_ast.StructDecl:



Static Public Attributes

- str name
- List fields [FieldDecl]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.175.1 Detailed Description

A struct declaration.

Attributes

- `name`: The name of the struct.
- `fields`: List of field declarations.

7.175.2 Member Data Documentation

7.175.2.1 fields

```
List l0_ast.StructDecl.fields [FieldDecl]  [static]
```

7.175.2.2 name

```
str l0_ast.StructDecl.name  [static]
```

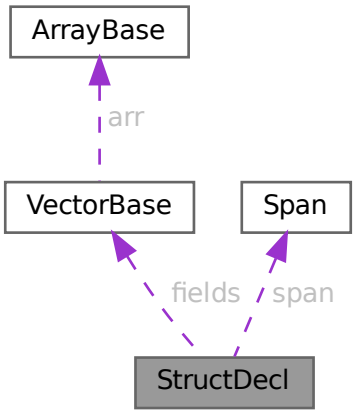
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.176 StructDecl Struct Reference

Represents the StructDecl structure.

Collaboration diagram for StructDecl:



Public Attributes

- name: string
- fields: VectorBase*
- span: Span

7.176.1 Detailed Description

Represents the StructDecl structure.

7.176.2 Member Data Documentation

7.176.2.1 fields

fields: `fields: VectorBase*`

7.176.2.2 name

name: `name: string`

7.176.2.3 span

span: `span: Span`

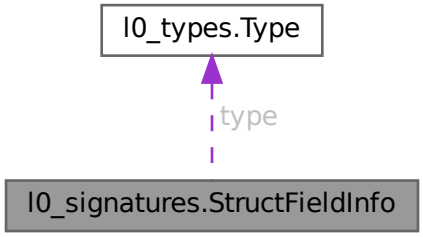
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

7.177 l0_signatures.StructFieldInfo Class Reference

Information about a single struct field.

Collaboration diagram for l0_signatures.StructFieldInfo:



Static Public Attributes

- str name
- Type type

7.177.1 Detailed Description

Information about a single struct field.

Attributes

- `name`: The name of the field.
- `type`: The resolved Type of the field.

7.177.2 Member Data Documentation

7.177.2.1 name

```
str l0_signatures.StructFieldInfo.name  [static]
```

7.177.2.2 type

```
Type l0_signatures.StructFieldInfo.type  [static]
```

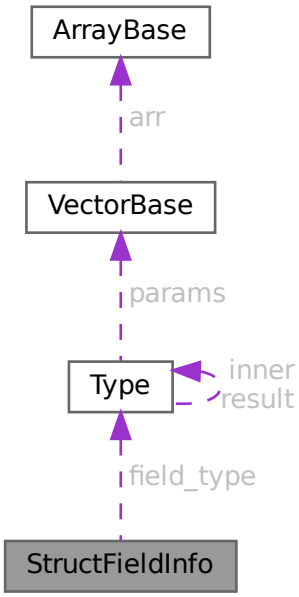
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_signatures.py

7.178 StructFieldInfo Struct Reference

Represents the StructFieldInfo structure.

Collaboration diagram for StructFieldInfo:



Public Attributes

- name: string
- field_type: Type*

7.178.1 Detailed Description

Represents the StructFieldInfo structure.

7.178.2 Member Data Documentation

7.178.2.1 field_type

field_type: `field_type: Type*`

7.178.2.2 name

name: `name: string`

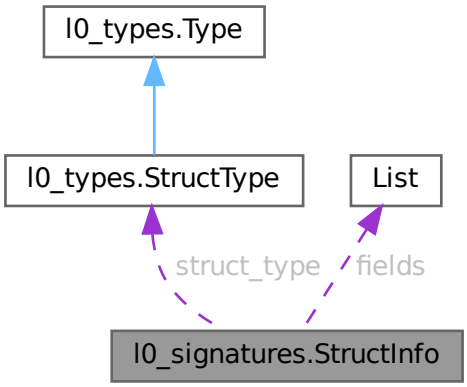
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/types.l0

7.179 l0_signatures.StructInfo Class Reference

Resolved type information for a struct.

Collaboration diagram for l0_signatures.StructInfo:



Static Public Attributes

- StructType struct_type
- List fields [StructFieldInfo]

7.179.1 Detailed Description

Resolved type information for a struct.

Attributes

- `struct_type`: The base StructType.
- `fields`: List of StructFieldInfo objects for all fields.

7.179.2 Member Data Documentation

7.179.2.1 fields

```
List l0_signatures.StructInfo.fields [StructFieldInfo]  [static]
```

7.179.2.2 struct_type

```
StructType l0_signatures.StructInfo.struct_type  [static]
```

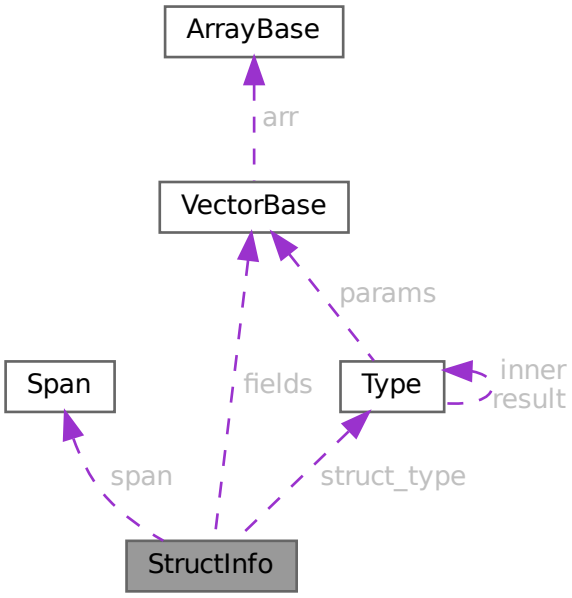
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_signatures.py

7.180 StructInfo Struct Reference

Represents the StructInfo structure.

Collaboration diagram for StructInfo:



Public Attributes

- key: string
- module_name: string
- name: string
- filename: string
- span: Span
- struct_type: Type*
- fields: VectorBase*

7.180.1 Detailed Description

Represents the StructInfo structure.

7.180.2 Member Data Documentation

7.180.2.1 fields

fields: `fields: VectorBase*`

7.180.2.2 filename

filename: `filename: string`

7.180.2.3 key

key: `key: string`

7.180.2.4 module_name

module_name: `module_name: string`

7.180.2.5 name

name: `name: string`

7.180.2.6 span

span: `span: Span`

7.180.2.7 struct_type

struct_type: `struct_type: Type*`

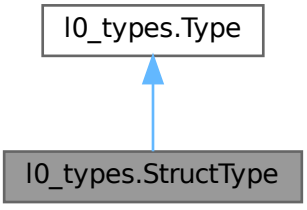
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/types.l0

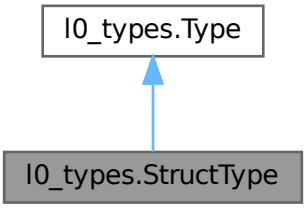
7.181 l0_types.StructType Class Reference

A user-defined struct type.

Inheritance diagram for l0_types.StructType:



Collaboration diagram for l0_types.StructType:



Static Public Attributes

- str module
- str name

7.181.1 Detailed Description

A user-defined struct type.

Attributes

- `module`: The name of the module where the struct is defined.
- `name`: The name of the struct.

7.181.2 Member Data Documentation

7.181.2.1 module

```
str l0_types.StructType.module  [static]
```

7.181.2.2 name

```
str l0_types.StructType.name  [static]
```

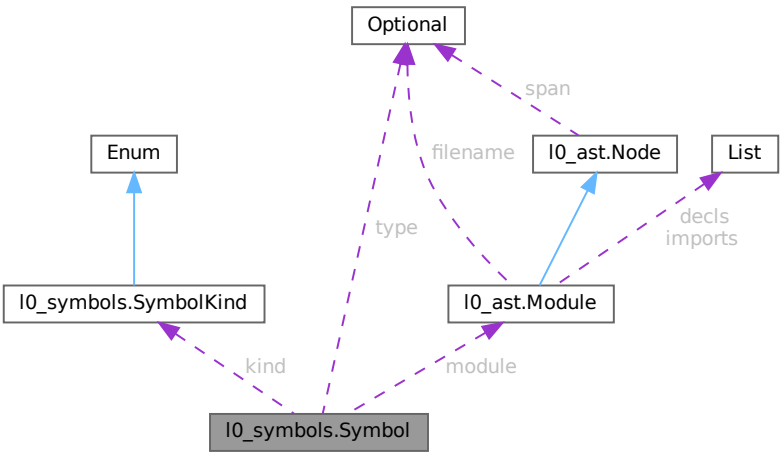
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_types.py

7.182 l0_symbols.Symbol Class Reference

A resolved module-level symbol.

Collaboration diagram for l0_symbols.Symbol:



Static Public Attributes

- str name
- SymbolKind kind
- Module module
- object node
- Optional type = None

7.182.1 Detailed Description

A resolved module-level symbol.

Attributes

- `name`: The name of the symbol.
- `kind`: The category of the symbol.
- `module`: The module that defines this symbol.
- `node`: The AST node where the symbol was defined.
- `type`: Optional resolved semantic Type.

7.182.2 Member Data Documentation

7.182.2.1 kind

```
SymbolKind l0_symbols.Symbol.kind  [static]
```

7.182.2.2 module

```
Module l0_symbols.Symbol.module  [static]
```

7.182.2.3 name

```
str l0_symbols.Symbol.name  [static]
```

7.182.2.4 node

```
object l0_symbols.Symbol.node  [static]
```

7.182.2.5 type

```
Optional l0_symbols.Symbol.type = None  [static]
```

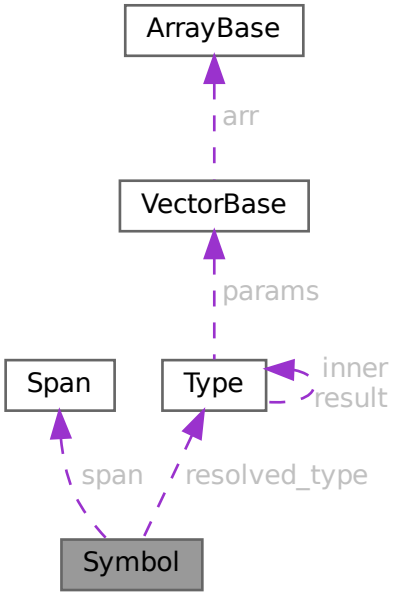
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_symbols.py

7.183 Symbol Struct Reference

Represents the Symbol structure.

Collaboration diagram for Symbol:



Public Attributes

- name: string
- kind: SymbolKind
- module_name: string
- owner_name: string
- decl_ptr: void*
- span: Span
- resolved_type: Type*?

7.183.1 Detailed Description

Represents the Symbol structure.

7.183.2 Member Data Documentation

7.183.2.1 decl_ptr

decl_ptr: `decl_ptr: void*`

7.183.2.2 kind

kind: `kind: SymbolKind`

7.183.2.3 module_name

module_name: `module_name: string`

7.183.2.4 name

name: `name: string`

7.183.2.5 owner_name

owner_name: `owner_name: string`

7.183.2.6 resolved_type

resolved_type: `resolved_type: Type*?`

7.183.2.7 span

span: `span: Span`

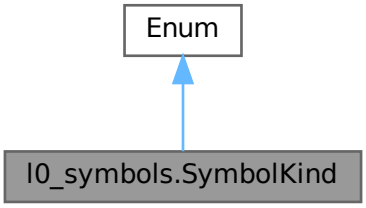
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/symbols.l0

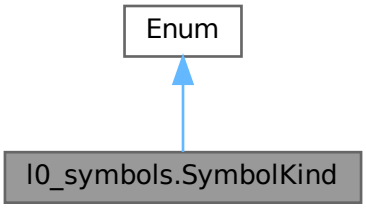
7.184 l0_symbols.SymbolKind Class Reference

Enumeration of all top-level symbol categories.

Inheritance diagram for l0_symbols.SymbolKind:



Collaboration diagram for l0_symbols.SymbolKind:



Static Public Attributes

- MODULE = auto()
- FUNC = auto()
- STRUCT = auto()
- ENUM = auto()
- ENUM_VARIANT = auto()
- TYPE_ALIAS = auto()
- LET = auto()

7.184.1 Detailed Description

Enumeration of all top-level symbol categories.

Attributes

- `MODULE`: A module; reserved for future use (aliases, etc.).
- `FUNC`: A function.
- `STRUCT`: A struct type.
- `ENUM`: An enum type.
- `ENUM_VARIANT`: An enum variant constructor.
- `TYPE_ALIAS`: A type alias.
- `LET`: A module-level constant or variable.

7.184.2 Member Data Documentation

7.184.2.1 ENUM

```
l0_symbols.SymbolKind.ENUM = auto()  [static]
```

7.184.2.2 ENUM_VARIANT

```
l0_symbols.SymbolKind.ENUM_VARIANT = auto()  [static]
```

7.184.2.3 FUNC

```
l0_symbols.SymbolKind.FUNC = auto()  [static]
```

7.184.2.4 LET

```
l0_symbols.SymbolKind.LET = auto()  [static]
```

7.184.2.5 MODULE

```
l0_symbols.SymbolKind.MODULE = auto()  [static]
```

7.184.2.6 STRUCT

```
l0_symbols.SymbolKind.STRUCT = auto()  [static]
```

7.184.2.7 TYPE_ALIAS

```
l0_symbols.SymbolKind.TYPE_ALIAS = auto()  [static]
```

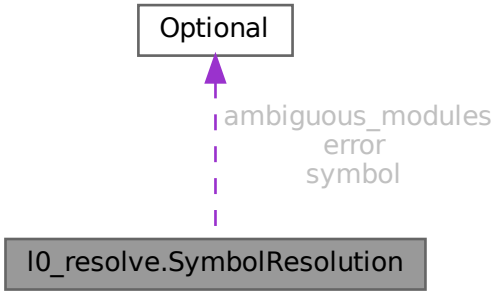
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_symbols.py

7.185 l0_resolve.SymbolResolution Class Reference

The result of a symbol resolution attempt.

Collaboration diagram for l0_resolve.SymbolResolution:



Static Public Attributes

- Optional symbol [Symbol]
- Optional error [ResolveErrorKind]
- str module_name
- str name
- Optional ambiguous_modules = None

7.185.1 Detailed Description

The result of a symbol resolution attempt.

Attributes

- `symbol`: The resolved Symbol if successful, otherwise None.
- `error`: The kind of error that occurred, or None if successful.
- `module_name`: The name of the module where resolution was attempted.
- `name`: The name of the symbol being resolved.
- `ambiguous_modules`: Tuple of module names if resolution failed due to ambiguity.

7.185.2 Member Data Documentation

7.185.2.1 ambiguous_modules

```
Optional l0_resolve.SymbolResolution.ambiguous_modules = None  [static]
```

7.185.2.2 error

```
Optional l0_resolve.SymbolResolution.error [ResolveErrorKind]  [static]
```

7.185.2.3 module_name

```
str l0_resolve.SymbolResolution.module_name  [static]
```

7.185.2.4 name

```
str l0_resolve.SymbolResolution.name  [static]
```

7.185.2.5 symbol

```
Optional l0_resolve.SymbolResolution.symbol [Symbol]  [static]
```

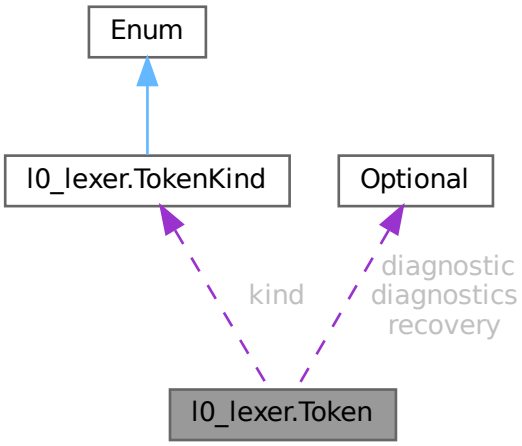
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_resolve.py

7.186 l0_lexer.Token Class Reference

A single token produced by the lexer.

Collaboration diagram for l0_lexer.Token:



Public Member Functions

- str __repr__ (self)

  Returns a string representation of the token.

Static Public Attributes

- TokenKind kind
- str text
- int line
- int column
- Optional diagnostic = None
- Optional diagnostics = None
- Optional recovery = None

7.186.1 Detailed Description

A single token produced by the lexer.

Attributes

- `kind`: The kind of token.
- `text`: The literal text of the token.
- `line`: 1-based line number in source.
- `column`: 1-based column number in source.
- `diagnostic`: Deferred diagnostic carried by LEXER_ERROR tokens; kept for compatibility with existing single-diagnostic callers.
- `diagnostics`: Deferred diagnostics carried by LEXER_ERROR tokens; the parser emits them when it consumes or skips the wrapper.
- `recovery`: Optional parser-visible logical token for recoverable lexer diagnostics.

7.186.2 Member Function Documentation

7.186.2.1 __repr__()

```
str l0_lexer.Token.__repr__ (
              self )
```

Returns a string representation of the token.

7.186.3 Member Data Documentation

7.186.3.1 column

```
int l0_lexer.Token.column  [static]
```

7.186.3.2 diagnostic

```
Optional l0_lexer.Token.diagnostic = None  [static]
```

7.186.3.3 diagnostics

```
Optional l0_lexer.Token.diagnostics = None  [static]
```

7.186.3.4 kind

```
TokenKind l0_lexer.Token.kind  [static]
```

7.186.3.5 line

```
int l0_lexer.Token.line  [static]
```

7.186.3.6 recovery

```
Optional l0_lexer.Token.recovery = None  [static]
```

7.186.3.7 text

```
str l0_lexer.Token.text  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_lexer.py

7.187 Token Struct Reference

Represents the Token structure.

Public Attributes

- token_type: TokenType
- index: int
- line: int
- column: int

7.187.1 Detailed Description

Represents the Token structure.

7.187.2 Member Data Documentation

7.187.2.1 column

column: `column: int`

7.187.2.2 index

index: `index: int`

7.187.2.3 line

line: `line: int`

7.187.2.4 token_type

token_type: `token_type: TokenType`

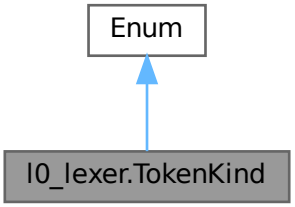
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/tokens.l0

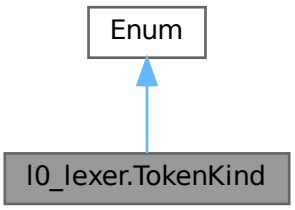
7.188 l0_lexer.TokenKind Class Reference

Enumeration of all supported token types in L0.

Inheritance diagram for l0_lexer.TokenKind:



Collaboration diagram for l0_lexer.TokenKind:



Static Public Attributes

- EOF = auto()
- LEXER_ERROR = auto()
- IDENT = auto()
- UNDERSCORE = auto()
- INT = auto()
- BYTE = auto()
- STRING = auto()
- MODULE = auto()
- IMPORT = auto()
- FUNC = auto()
- STRUCT = auto()
- ENUM = auto()
- TYPE = auto()
- EXTERN = auto()
- LET = auto()
- RETURN = auto()
- MATCH = auto()
- CASE = auto()
- IF = auto()
- ELSE = auto()
- WHILE = auto()
- FOR = auto()
- BREAK = auto()
- CONTINUE = auto()
- TRUE = auto()
- FALSE = auto()
- NULL = auto()
- AS = auto()
- NEW = auto()
- DROP = auto()
- WITH = auto()
- CLEANUP = auto()
- LBRACE = auto()
- RBRACE = auto()
- LPAREN = auto()
- RPAREN = auto()

- LBRACKET = auto()
- RBRACKET = auto()
- COMMA = auto()
- SEMI = auto()
- COLON = auto()
- DOUBLE_COLON = auto()
- ARROW_FUNC = auto()
- ARROW_MATCH = auto()
- EQ = auto()
- PLUS = auto()
- MINUS = auto()
- STAR = auto()
- SLASH = auto()
- MODULO = auto()
- LT = auto()
- GT = auto()
- LE = auto()
- GE = auto()
- EQEQ = auto()
- NE = auto()
- ANDAND = auto()
- OROR = auto()
- BANG = auto()
- QUESTION = auto()
- DOT = auto()
- AMP = auto()
- PIPE = auto()
- CARET = auto()
- TILDE = auto()
- LSHIFT = auto()
- RSHIFT = auto()
- FUTURE_EXTENSION = auto()

7.188.1 Detailed Description

Enumeration of all supported token types in L0.

Attributes

- `EOF`: End of file marker.
- `IDENT`: Identifier (e.g., variable or function name).
- `UNDERSCORE`: Wildcard identifier ('_').
- `INT`: Integer literal.
- `BYTE`: Byte/character literal.
- `STRING`: String literal.
- `MODULE`: 'module' keyword.
- `IMPORT`: 'import' keyword.
- `FUNC`: 'func' keyword.
- `STRUCT`: 'struct' keyword.
- `ENUM`: 'enum' keyword.
- `TYPE`: 'type' keyword.

- EXTERN: 'extern' keyword.
- LET: 'let' keyword.
- RETURN: 'return' keyword.
- MATCH: 'match' keyword.
- CASE: 'case' keyword.
- IF: 'if' keyword.
- ELSE: 'else' keyword.
- WHILE: 'while' keyword.
- FOR: 'for' keyword.
- BREAK: 'break' keyword.
- CONTINUE: 'continue' keyword.
- TRUE: 'true' keyword.
- FALSE: 'false' keyword.
- NULL: 'null' keyword.
- AS: 'as' keyword.
- NEW: 'new' keyword.
- DROP: 'drop' keyword.
- WITH: 'with' keyword.
- CLEANUP: 'cleanup' keyword.
- LBRACE: '{'
- RBRACE: '}'
- LPAREN: '('
- RPAREN: ')'
- LBRACKET: '['
- RBRACKET: ']'
- COMMA: ','
- SEMI: ';'
- COLON: ':'
- DOUBLE_COLON: '::'
- ARROW_FUNC: '→'
- ARROW_MATCH: '=>'
- EQ: '='
- PLUS: '+'
- MINUS: '-'
- STAR: '*'
- SLASH: '/'
- MODULO: ''
- LT: '<'
- GT: '>'
- LE: '<='
- GE: '>='
- EQEQ: '=='
- NE: '!='
- ANDAND: '&&'
- OROR: '||'

- BANG: '!'
- QUESTION: '?'
- DOT: '.'
- AMP: '&' (reserved)
- PIPE: '|' (reserved)
- CARET: '^' (reserved)
- TILDE: '~' (reserved)
- LSHIFT: '<<' (reserved)
- RSHIFT: '>>' (reserved)
- FUTURE_EXTENSION: Placeholder for future tokens.
- LEXER_ERROR: Lexer diagnostic wrapper with an optional recovery token.

7.188.2 Member Data Documentation

7.188.2.1 AMP

```
l0_lexer.TokenKind.AMP = auto()  [static]
```

7.188.2.2 ANDAND

```
l0_lexer.TokenKind.ANDAND = auto()  [static]
```

7.188.2.3 ARROW_FUNC

```
l0_lexer.TokenKind.ARROW_FUNC = auto()  [static]
```

7.188.2.4 ARROW_MATCH

```
l0_lexer.TokenKind.ARROW_MATCH = auto()  [static]
```

7.188.2.5 AS

```
l0_lexer.TokenKind.AS = auto()  [static]
```

7.188.2.6 BANG

```
l0_lexer.TokenKind.BANG = auto()  [static]
```

7.188.2.7 BREAK

```
l0_lexer.TokenKind.BREAK = auto()  [static]
```

7.188.2.8 BYTE

```
l0_lexer.TokenKind.BYTE = auto()  [static]
```

7.188.2.9 CARET

```
l0_lexer.TokenKind.CARET = auto()  [static]
```

7.188.2.10 CASE

```
l0_lexer.TokenKind.CASE = auto()  [static]
```

7.188.2.11 CLEANUP

```
l0_lexer.TokenKind.CLEANUP = auto()  [static]
```

7.188.2.12 COLON

```
l0_lexer.TokenKind.COLON = auto()  [static]
```

7.188.2.13 COMMA

```
l0_lexer.TokenKind.COMMA = auto()  [static]
```

7.188.2.14 CONTINUE

```
l0_lexer.TokenKind.CONTINUE = auto()  [static]
```

7.188.2.15 DOT

```
l0_lexer.TokenKind.DOT = auto()  [static]
```

7.188.2.16 DOUBLE_COLON

```
l0_lexer.TokenKind.DOUBLE_COLON = auto()  [static]
```

7.188.2.17 DROP

```
l0_lexer.TokenKind.DROP = auto()  [static]
```

7.188.2.18 ELSE

```
l0_lexer.TokenKind.ELSE = auto()  [static]
```

7.188.2.19 ENUM

```
l0_lexer.TokenKind.ENUM = auto()  [static]
```

7.188.2.20 EOF

```
l0_lexer.TokenKind.EOF = auto()  [static]
```

7.188.2.21 EQ

```
l0_lexer.TokenKind.EQ = auto()  [static]
```

7.188.2.22 EQEQ

```
l0_lexer.TokenKind.EQEQ = auto()  [static]
```

7.188.2.23 EXTERN

```
l0_lexer.TokenKind.EXTERN = auto()  [static]
```

7.188.2.24 FALSE

```
l0_lexer.TokenKind.FALSE = auto()  [static]
```

7.188.2.25 FOR

```
l0_lexer.TokenKind.FOR = auto()  [static]
```

7.188.2.26 FUNC

```
l0_lexer.TokenKind.FUNC = auto()  [static]
```

7.188.2.27 FUTURE_EXTENSION

```
l0_lexer.TokenKind.FUTURE_EXTENSION = auto()  [static]
```

7.188.2.28 GE

```
l0_lexer.TokenKind.GE = auto()  [static]
```

7.188.2.29 GT

```
l0_lexer.TokenKind.GT = auto()  [static]
```

7.188.2.30 IDENT

```
l0_lexer.TokenKind.IDENT = auto()  [static]
```

7.188.2.31 IF

```
l0_lexer.TokenKind.IF = auto()  [static]
```

7.188.2.32 IMPORT

```
l0_lexer.TokenKind.IMPORT = auto()  [static]
```

7.188.2.33 INT

```
l0_lexer.TokenKind.INT = auto()  [static]
```

7.188.2.34 LBRACE

```
l0_lexer.TokenKind.LBRACE = auto()  [static]
```

7.188.2.35 LBRACKET

```
l0_lexer.TokenKind.LBRACKET = auto()  [static]
```

7.188.2.36 LE

```
l0_lexer.TokenKind.LE = auto()  [static]
```

7.188.2.37 LET

```
l0_lexer.TokenKind.LET = auto()  [static]
```

7.188.2.38 LEXER_ERROR

```
l0_lexer.TokenKind.LEXER_ERROR = auto()  [static]
```

7.188.2.39 LPAREN

```
l0_lexer.TokenKind.LPAREN = auto()  [static]
```

7.188.2.40 LSHIFT

```
l0_lexer.TokenKind.LSHIFT = auto()  [static]
```

7.188.2.41 LT

```
l0_lexer.TokenKind.LT = auto()  [static]
```

7.188.2.42 MATCH

```
l0_lexer.TokenKind.MATCH = auto()  [static]
```

7.188.2.43 MINUS

```
l0_lexer.TokenKind.MINUS = auto()  [static]
```

7.188.2.44 MODULE

```
l0_lexer.TokenKind.MODULE = auto()  [static]
```

7.188.2.45 MODULO

```
l0_lexer.TokenKind.MODULO = auto()  [static]
```

7.188.2.46 NE

```
l0_lexer.TokenKind.NE = auto()  [static]
```

7.188.2.47 NEW

```
l0_lexer.TokenKind.NEW = auto()  [static]
```

7.188.2.48 NULL

```
l0_lexer.TokenKind.NULL = auto()  [static]
```

7.188.2.49 OROR

```
l0_lexer.TokenKind.OROR = auto()  [static]
```

7.188.2.50 PIPE

```
l0_lexer.TokenKind.PIPE = auto()  [static]
```

7.188.2.51 PLUS

```
l0_lexer.TokenKind.PLUS = auto()  [static]
```

7.188.2.52 QUESTION

```
l0_lexer.TokenKind.QUESTION = auto()  [static]
```

7.188.2.53 RBRACE

```
l0_lexer.TokenKind.RBRACE = auto()  [static]
```

7.188.2.54 RBRACKET

```
l0_lexer.TokenKind.RBRACKET = auto()  [static]
```

7.188.2.55 RETURN

```
l0_lexer.TokenKind.RETURN = auto()  [static]
```

7.188.2.56 RPAREN

```
l0_lexer.TokenKind.RPAREN = auto()  [static]
```

7.188.2.57 RSHIFT

```
l0_lexer.TokenKind.RSHIFT = auto()  [static]
```

7.188.2.58 SEMI

```
l0_lexer.TokenKind.SEMI = auto()  [static]
```

7.188.2.59 SLASH

```
l0_lexer.TokenKind.SLASH = auto()  [static]
```

7.188.2.60 STAR

```
l0_lexer.TokenKind.STAR = auto()  [static]
```

7.188.2.61 STRING

```
l0_lexer.TokenKind.STRING = auto()  [static]
```

7.188.2.62 STRUCT

```
l0_lexer.TokenKind.STRUCT = auto()  [static]
```

7.188.2.63 TILDE

```
l0_lexer.TokenKind.TILDE = auto()  [static]
```

7.188.2.64 TRUE

```
l0_lexer.TokenKind.TRUE = auto()  [static]
```

7.188.2.65 TYPE

```
l0_lexer.TokenKind.TYPE = auto()  [static]
```

7.188.2.66 UNDERSCORE

```
l0_lexer.TokenKind.UNDERSCORE = auto()  [static]
```

7.188.2.67 WHILE

```
l0_lexer.TokenKind.WHILE = auto()  [static]
```

7.188.2.68 WITH

```
l0_lexer.TokenKind.WITH = auto()  [static]
```

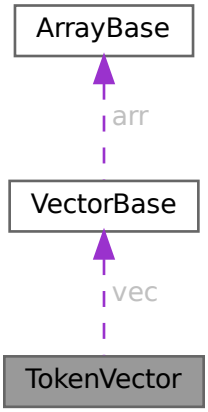
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_lexer.py

7.189 TokenVector Struct Reference

Represents the TokenVector structure.

Collaboration diagram for TokenVector:



Public Attributes

- vec: VectorBase*

7.189.1 Detailed Description

Represents the TokenVector structure.

7.189.2 Member Data Documentation

7.189.2.1 vec

vec: `vec: VectorBase*`

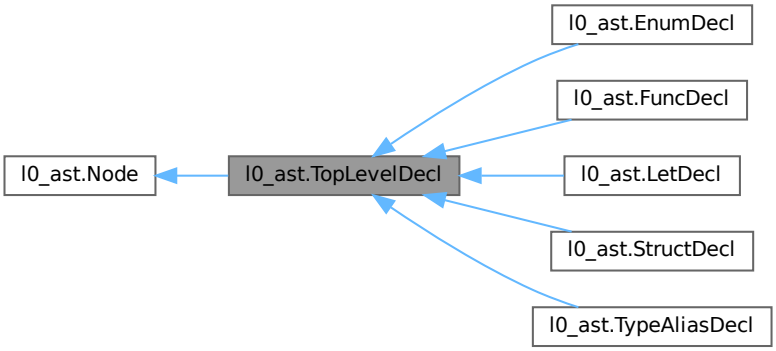
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/tokens.l0

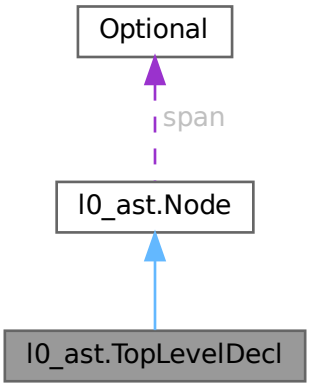
7.190 l0_ast.TopLevelDecl Class Reference

Base class for top-level declarations (functions, structs, etc.).

Inheritance diagram for l0_ast.TopLevelDecl:



Collaboration diagram for l0_ast.TopLevelDecl:



Additional Inherited Members

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.190.1 Detailed Description

Base class for top-level declarations (functions, structs, etc.).

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.191 TopLevelDecl Struct Reference

Represents the TopLevelDecl structure.

Public Attributes

- kind: TopLevelDeclKind
- data: void*

7.191.1 Detailed Description

Represents the TopLevelDecl structure.

7.191.2 Member Data Documentation

7.191.2.1 data

data: `data: void*`

7.191.2.2 kind

kind: `kind: TopLevelDeclKind`

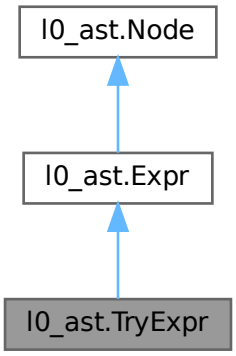
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

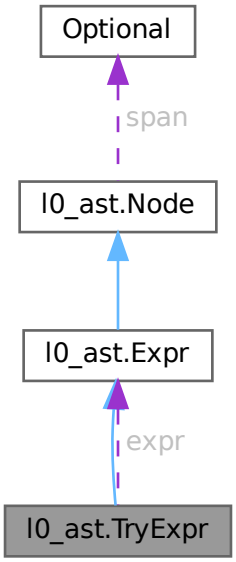
7.192 l0_ast.TryExpr Class Reference

A try expression for error handling.

Inheritance diagram for l0_ast.TryExpr:



Collaboration diagram for l0_ast.TryExpr:



Static Public Attributes

- Expr expr

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.192.1 Detailed Description

A try expression for error handling.

Attributes

- `expr`: The expression to attempt.

7.192.2 Member Data Documentation

7.192.2.1 expr

```
Expr l0_ast.TryExpr.expr  [static]
```

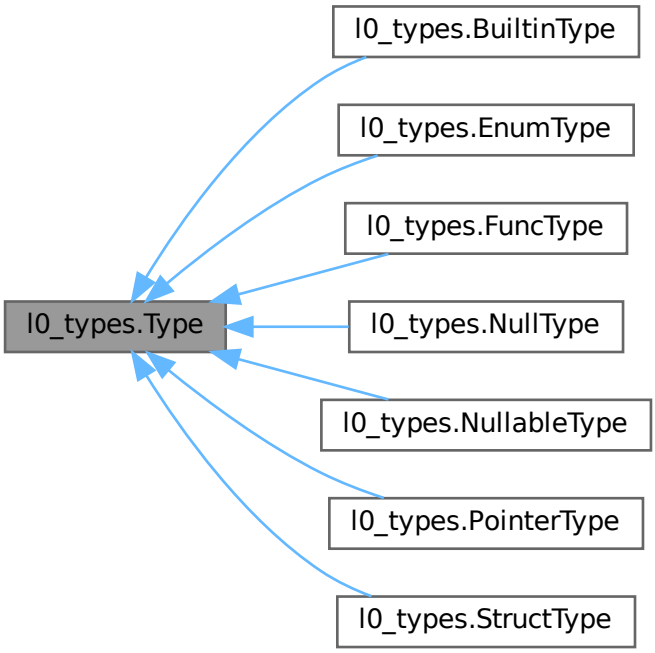
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.193 l0_types.Type Class Reference

Base class for all semantic types.

Inheritance diagram for l0_types.Type:



7.193.1 Detailed Description

Base class for all semantic types.

Used as a common marker; concrete types are represented by specialized dataclasses.

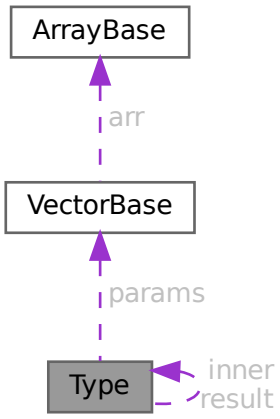
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_types.py

7.194 Type Struct Reference

Represents the Type structure.

Collaboration diagram for Type:



Public Attributes

- kind: TypeKind
- module_name: string
- name: string
- inner: Type∗?
- params: VectorBase∗?
- result: Type∗?

7.194.1 Detailed Description

Represents the Type structure.

7.194.2 Member Data Documentation

7.194.2.1 inner

inner: `inner: Type*?`

7.194.2.2 kind

kind: `kind: TypeKind`

7.194.2.3 module_name

module_name: `module_name: string`

7.194.2.4 name

name: `name: string`

7.194.2.5 params

params: `params: VectorBase*?`

7.194.2.6 result

result: `result: Type*?`

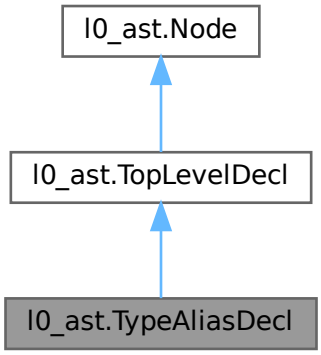
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/types.l0

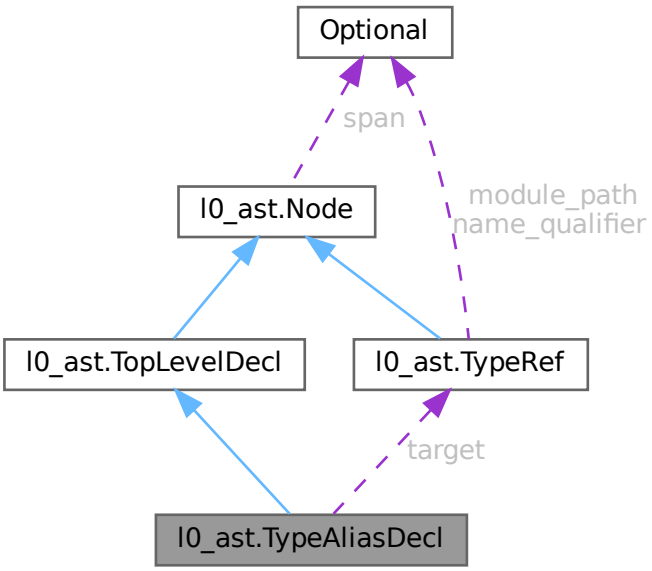
7.195 l0_ast.TypeAliasDecl Class Reference

A type alias declaration.

Inheritance diagram for l0_ast.TypeAliasDecl:



Collaboration diagram for l0_ast.TypeAliasDecl:



Static Public Attributes

- str name
- TypeRef target

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.195.1 Detailed Description

A type alias declaration.

Attributes

- `name`: The name of the alias.
- `target`: The target type being aliased.

7.195.2 Member Data Documentation

7.195.2.1 name

```
str l0_ast.TypeAliasDecl.name  [static]
```

7.195.2.2 target

```
TypeRef l0_ast.TypeAliasDecl.target  [static]
```

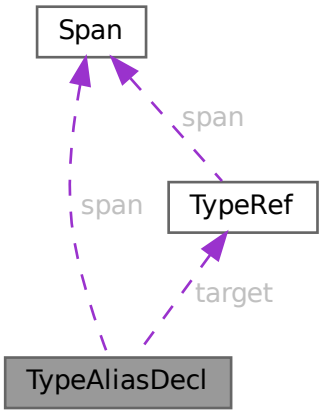
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.196 TypeAliasDecl Struct Reference

Represents the TypeAliasDecl structure.

Collaboration diagram for TypeAliasDecl:



Public Attributes

- name: string
- target: TypeRef*
- span: Span

7.196.1 Detailed Description

Represents the TypeAliasDecl structure.

7.196.2 Member Data Documentation

7.196.2.1 name

name: `name: string`

7.196.2.2 span

span: `span: Span`

7.196.2.3 target

target: `target: TypeRef*`

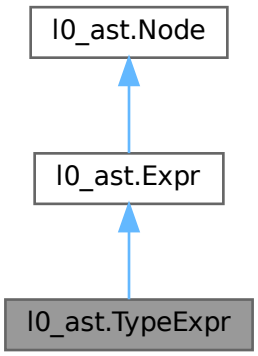
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

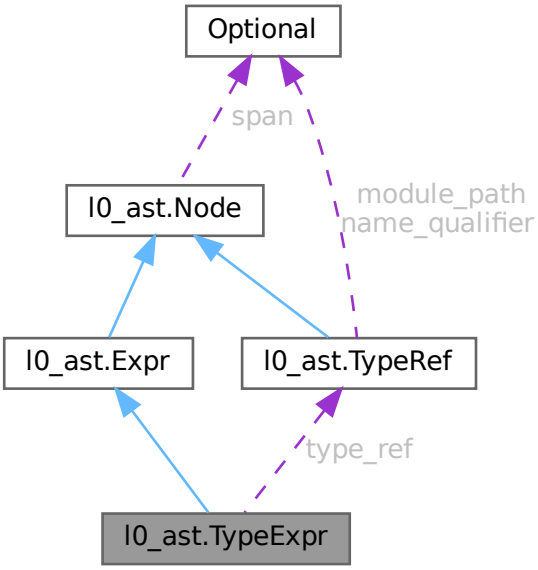
7.197 l0_ast.TypeExpr Class Reference

A type used in an expression position.

Inheritance diagram for l0_ast.TypeExpr:



Collaboration diagram for l0_ast.TypeExpr:



Static Public Attributes

- TypeRef type_ref

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.197.1 Detailed Description

A type used in an expression position.

Attributes

- `type_ref`: The type reference.

7.197.2 Member Data Documentation

7.197.2.1 type_ref

```
TypeRef l0_ast.TypeExpr.type_ref  [static]
```

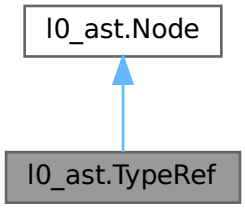
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

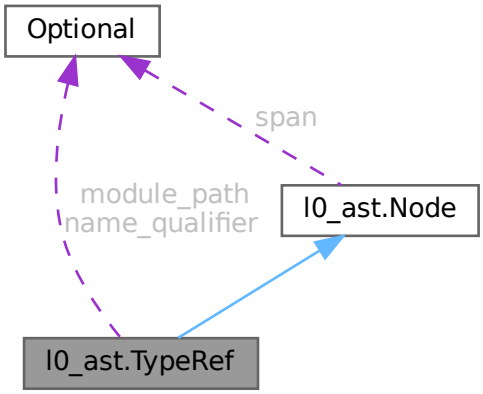
7.198 l0_ast.TypeRef Class Reference

Reference to a type in the source code.

Inheritance diagram for l0_ast.TypeRef:



Collaboration diagram for l0_ast.TypeRef:



Static Public Attributes

- str name
- int pointer_depth = 0
- bool is_nullable = False
- Optional module_path = None
- Optional name_qualifier = None

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.198.1 Detailed Description

Reference to a type in the source code.

Attributes

- `name`: The name of the type (e.g., "int", "MyStruct").
- `pointer_depth`: Number of pointer levels (number of '*' suffixes).
- `is_nullable`: Whether the type is nullable (trailing '?').
- `module_path`: Optional list of module components for qualified names.
- `name_qualifier`: Optional extra segments for namespaced types.

7.198.2 Member Data Documentation

7.198.2.1 is_nullable

```
bool l0_ast.TypeRef.is_nullable = False  [static]
```

7.198.2.2 module_path

```
Optional l0_ast.TypeRef.module_path = None  [static]
```

7.198.2.3 name

```
str l0_ast.TypeRef.name  [static]
```

7.198.2.4 name_qualifier

```
Optional l0_ast.TypeRef.name_qualifier = None  [static]
```

7.198.2.5 pointer_depth

```
int l0_ast.TypeRef.pointer_depth = 0  [static]
```

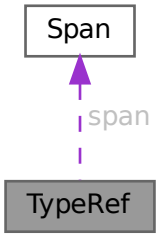
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.199 TypeRef Struct Reference

Represents the TypeRef structure.

Collaboration diagram for TypeRef:



Public Attributes

- name: string
- pointer_depth: int
- is_nullable: bool
- module_path: StringVector*?
- name_qualifier: StringVector*?
- span: Span

7.199.1 Detailed Description

Represents the TypeRef structure.

7.199.2 Member Data Documentation

7.199.2.1 is_nullable

is_nullable: `is_nullable: bool`

7.199.2.2 module_path

module_path: `module_path: StringVector*?`

7.199.2.3 name

name: `name: string`

7.199.2.4 name_qualifier

name_qualifier: `name_qualifier: StringVector*?`

7.199.2.5 pointer_depth

pointer_depth: `pointer_depth: int`

7.199.2.6 span

span: `span: Span`

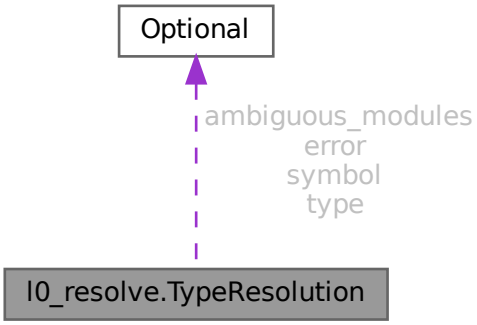
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

7.200 l0_resolve.TypeResolution Class Reference

The result of a type reference resolution attempt.

Collaboration diagram for l0_resolve.TypeResolution:



Static Public Attributes

- Optional type [Type]
- Optional error [TypeResolveErrorKind]
- str module_name
- str name
- Optional symbol = None
- Optional ambiguous_modules = None

7.200.1 Detailed Description

The result of a type reference resolution attempt.

Attributes

- `type`: The resolved Type if successful, otherwise None.
- `error`: The kind of error that occurred, or None if successful.
- `module_name`: The name of the module where resolution was attempted.
- `name`: The name of the type being resolved.
- `symbol`: The Symbol object corresponding to the type, if applicable.
- `ambiguous_modules`: Tuple of module names if resolution failed due to ambiguity.

7.200.2 Member Data Documentation

7.200.2.1 ambiguous_modules

```
Optional l0_resolve.TypeResolution.ambiguous_modules = None  [static]
```

7.200.2.2 error

```
Optional l0_resolve.TypeResolution.error [TypeResolveErrorKind]  [static]
```

7.200.2.3 module_name

```
str l0_resolve.TypeResolution.module_name  [static]
```

7.200.2.4 name

```
str l0_resolve.TypeResolution.name  [static]
```

7.200.2.5 symbol

```
Optional l0_resolve.TypeResolution.symbol = None  [static]
```

7.200.2.6 type

```
Optional l0_resolve.TypeResolution.type [Type]  [static]
```

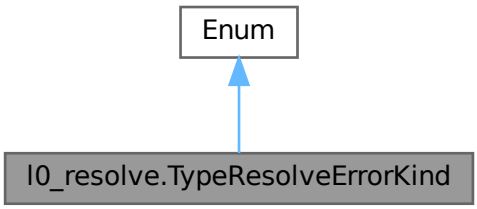
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_resolve.py

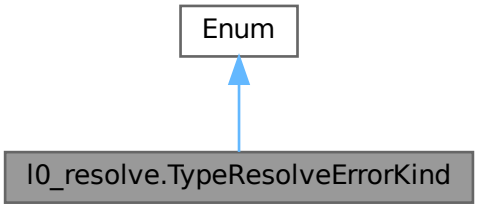
7.201 l0_resolve.TypeResolveErrorKind Class Reference

Kinds of errors that can occur during type resolution.

Inheritance diagram for l0_resolve.TypeResolveErrorKind:



Collaboration diagram for l0_resolve.TypeResolveErrorKind:



Static Public Attributes

- UNKNOWN_TYPE = auto()
- NOT_A_TYPE = auto()
- UNRESOLVED_ALIAS = auto()
- VARIANT_AS_TYPE = auto()
- UNKNOWN_MODULE = auto()
- MODULE_NOT_IMPORTED = auto()
- INVALID_NULLABLE_VOID = auto()
- AMBIGUOUS_TYPE = auto()

7.201.1 Detailed Description

Kinds of errors that can occur during type resolution.

7.201.2 Member Data Documentation

7.201.2.1 AMBIGUOUS_TYPE

```
l0_resolve.TypeResolveErrorKind.AMBIGUOUS_TYPE = auto()  [static]
```

7.201.2.2 INVALID_NULLABLE_VOID

```
l0_resolve.TypeResolveErrorKind.INVALID_NULLABLE_VOID = auto()  [static]
```

7.201.2.3 MODULE_NOT_IMPORTED

```
l0_resolve.TypeResolveErrorKind.MODULE_NOT_IMPORTED = auto()  [static]
```

7.201.2.4 NOT_A_TYPE

```
l0_resolve.TypeResolveErrorKind.NOT_A_TYPE = auto()  [static]
```

7.201.2.5 UNKNOWN_MODULE

```
l0_resolve.TypeResolveErrorKind.UNKNOWN_MODULE = auto()  [static]
```

7.201.2.6 UNKNOWN_TYPE

```
l0_resolve.TypeResolveErrorKind.UNKNOWN_TYPE = auto()  [static]
```

7.201.2.7 UNRESOLVED_ALIAS

```
l0_resolve.TypeResolveErrorKind.UNRESOLVED_ALIAS = auto()  [static]
```

7.201.2.8 VARIANT_AS_TYPE

```
l0_resolve.TypeResolveErrorKind.VARIANT_AS_TYPE = auto()  [static]
```

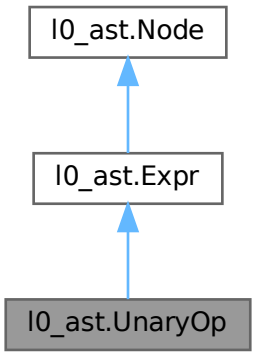
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_resolve.py

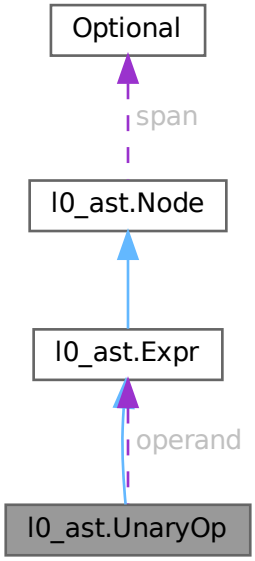
7.202 l0_ast.UnaryOp Class Reference

A unary operator expression.

Inheritance diagram for l0_ast.UnaryOp:



Collaboration diagram for l0_ast.UnaryOp:



Static Public Attributes

- str op
- Expr operand

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.202.1 Detailed Description

A unary operator expression.

Attributes

- `op`: The operator string (e.g., "-", "!").
- `operand`: The expression the operator is applied to.

7.202.2 Member Data Documentation

7.202.2.1 op

```
str l0_ast.UnaryOp.op  [static]
```

7.202.2.2 operand

```
Expr l0_ast.UnaryOp.operand  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.203 Unit Struct Reference

Definition of the Unit type, representing a value with no information.

7.203.1 Detailed Description

Definition of the Unit type, representing a value with no information.

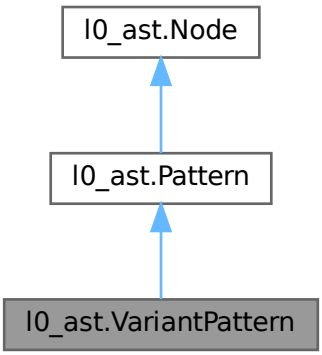
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/unit.l0

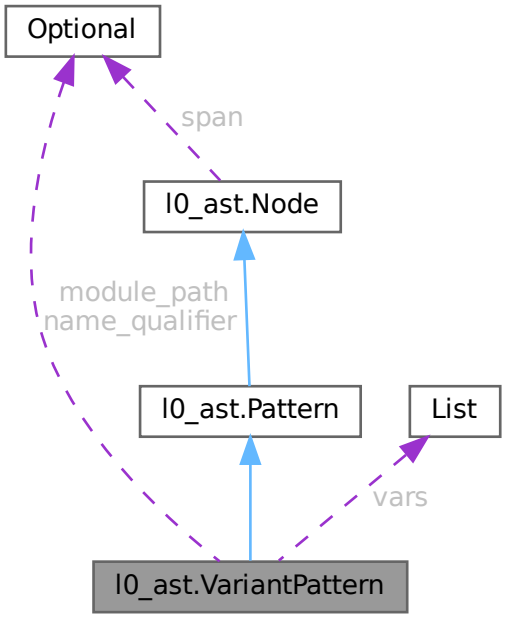
7.204 l0_ast.VariantPattern Class Reference

A pattern that matches an enum variant.

Inheritance diagram for l0_ast.VariantPattern:



Collaboration diagram for l0_ast.VariantPattern:



Static Public Attributes

- str name
- List vars [str]
- Optional module_path = None
- Optional name_qualifier = None

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.204.1 Detailed Description

A pattern that matches an enum variant.

Attributes

- `name`: The name of the variant.
- `vars`: List of variable names to bind to variant fields.
- `module_path`: Optional list of module components.
- `name_qualifier`: Optional extra name segments.

7.204.2 Member Data Documentation

7.204.2.1 module_path

```
Optional l0_ast.VariantPattern.module_path = None  [static]
```

7.204.2.2 name

```
str l0_ast.VariantPattern.name  [static]
```

7.204.2.3 name_qualifier

```
Optional l0_ast.VariantPattern.name_qualifier = None  [static]
```

7.204.2.4 vars

```
List l0_ast.VariantPattern.vars [str]  [static]
```

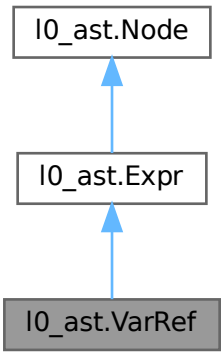
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

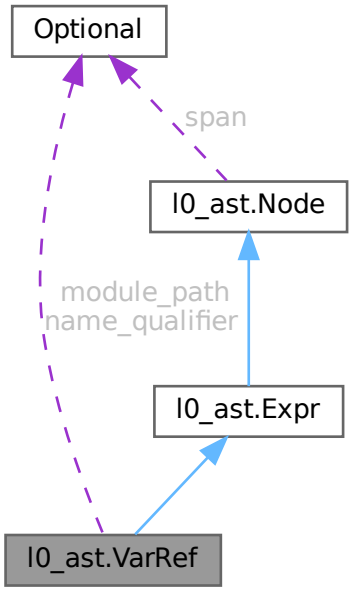
7.205 l0_ast.VarRef Class Reference

A variable reference expression.

Inheritance diagram for l0_ast.VarRef:



Collaboration diagram for l0_ast.VarRef:



Static Public Attributes

- str name
- Optional module_path = None
- Optional name_qualifier = None

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.205.1 Detailed Description

A variable reference expression.

Attributes

- `name`: The name of the variable.
- `module_path`: Optional list of module components.
- `name_qualifier`: Optional extra name segments.

7.205.2 Member Data Documentation

7.205.2.1 module_path

```
Optional l0_ast.VarRef.module_path = None  [static]
```

7.205.2.2 name

```
str l0_ast.VarRef.name  [static]
```

7.205.2.3 name_qualifier

```
Optional l0_ast.VarRef.name_qualifier = None  [static]
```

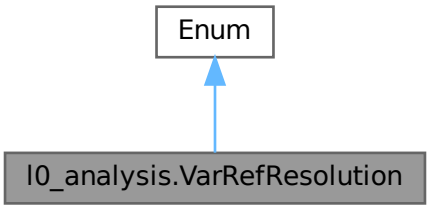
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

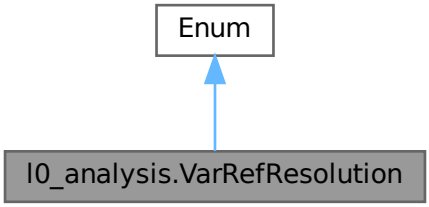
7.206 l0_analysis.VarRefResolution Class Reference

Enumeration of variable reference resolution targets.

Inheritance diagram for l0_analysis.VarRefResolution:



Collaboration diagram for l0_analysis.VarRefResolution:



Static Public Attributes

- str LOCAL = "local"
- str MODULE = "module"

7.206.1 Detailed Description

Enumeration of variable reference resolution targets.

7.206.2 Member Data Documentation

7.206.2.1 LOCAL

```
str l0_analysis.VarRefResolution.LOCAL = "local"  [static]
```

7.206.2.2 MODULE

```
str l0_analysis.VarRefResolution.MODULE = "module"  [static]
```

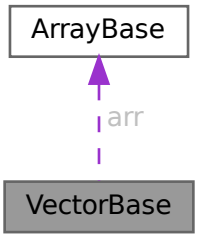
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_analysis.py

7.207 VectorBase Struct Reference

VectorBase implements a dynamic array (vector) of elements of a given size.

Collaboration diagram for VectorBase:



Public Attributes

- arr: ArrayBase*
- length: int

7.207.1 Detailed Description

VectorBase implements a dynamic array (vector) of elements of a given size.

It uses ArrayBase internally to manage storage. It can grow dynamically as elements are added, and a capacity can be reserved in advance.

Note: This is a low-level implementation and does not provide type safety. It is the caller's responsibility to manage types and casting.

7.207.2 Member Data Documentation

7.207.2.1 arr

arr: `arr:  ArrayBase*`

7.207.2.2 length

length: `length: int`

The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/vector.l0

7.208 WallTime Struct Reference

Represents wall clock time in seconds and nanoseconds since the Unix epoch.

Public Attributes

- sec: int
- nsec: int

7.208.1 Detailed Description

Represents wall clock time in seconds and nanoseconds since the Unix epoch.

7.208.2 Member Data Documentation

7.208.2.1 nsec

nsec: `nsec: int`

7.208.2.2 sec

sec: `sec: int`

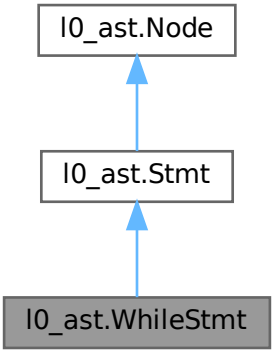
The documentation for this struct was generated from the following file:

- compiler/shared/l0/stdlib/std/time.l0

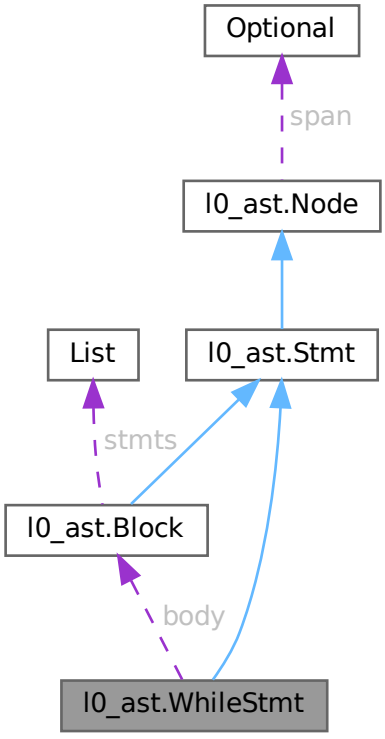
7.209 l0_ast.WhileStmt Class Reference

A while loop statement.

Inheritance diagram for l0_ast.WhileStmt:



Collaboration diagram for l0_ast.WhileStmt:



Static Public Attributes

- str cond : "Expr"
- Block body

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.209.1 Detailed Description

A while loop statement.

Attributes

- `cond`: The condition expression.
- `body`: The block to execute while the condition is true.

7.209.2 Member Data Documentation

7.209.2.1 body

```
Block l0_ast.WhileStmt.body  [static]
```

7.209.2.2 cond

```
str l0_ast.WhileStmt.cond :  "Expr"  [static]
```

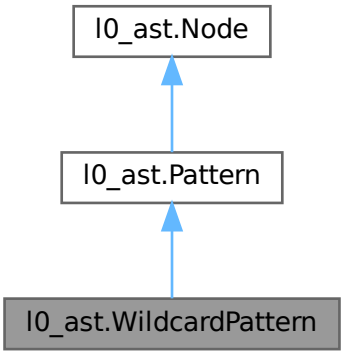
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

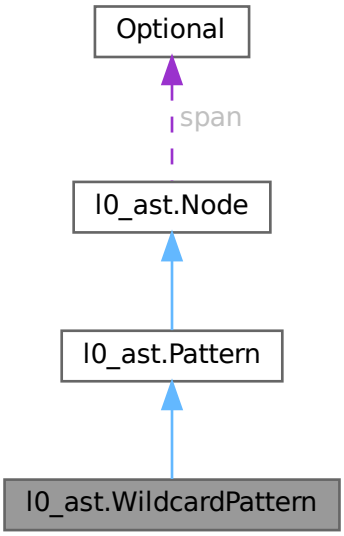
7.210 l0_ast.WildcardPattern Class Reference

A wildcard pattern ('_') that matches anything.

Inheritance diagram for l0_ast.WildcardPattern:



Collaboration diagram for l0_ast.WildcardPattern:



Additional Inherited Members

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.210.1 Detailed Description

A wildcard pattern ('_') that matches anything.

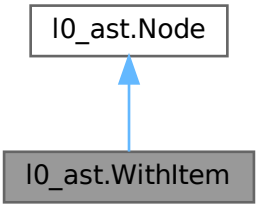
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

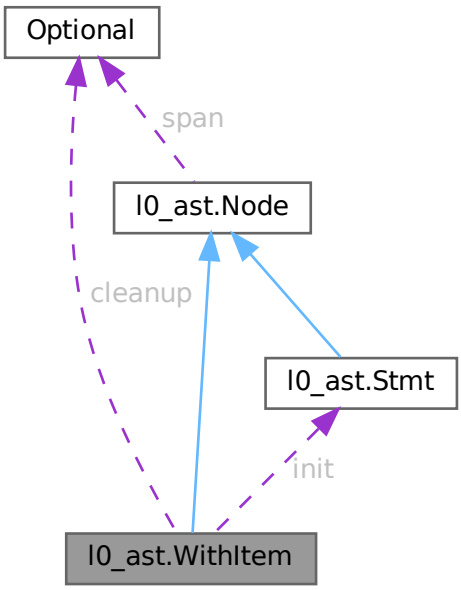
7.211 l0_ast.WithItem Class Reference

An item in a with statement.

Inheritance diagram for l0_ast.WithItem:



Collaboration diagram for l0_ast.WithItem:



Static Public Attributes

- Stmt init
- Optional cleanup [Stmt]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.211.1 Detailed Description

An item in a with statement.

Attributes

- `init`: Initialization statement.
- `cleanup`: Optional cleanup statement (mutually exclusive with a cleanup block in WithStmt).

7.211.2 Member Data Documentation

7.211.2.1 cleanup

```
Optional l0_ast.WithItem.cleanup [Stmt]  [static]
```

7.211.2.2 init

```
Stmt l0_ast.WithItem.init  [static]
```

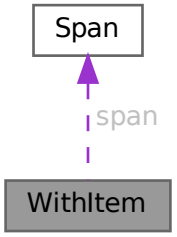
The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

7.212 WithItem Struct Reference

Represents the WithItem structure.

Collaboration diagram for WithItem:



Public Attributes

- init: StmtId
- cleanup_stmt: StmtId
- has_cleanup: bool
- span: Span

7.212.1 Detailed Description

Represents the WithItem structure.

7.212.2 Member Data Documentation

7.212.2.1 cleanup_stmt

cleanup_stmt: `cleanup_stmt: StmtId`

7.212.2.2 has_cleanup

has_cleanup: `has_cleanup: bool`

7.212.2.3 init

init: `init: StmtId`

7.212.2.4 span

span: `span: Span`

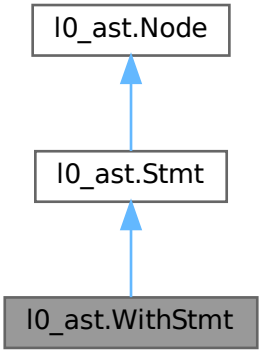
The documentation for this struct was generated from the following file:

- compiler/stage2_l0/src/ast.l0

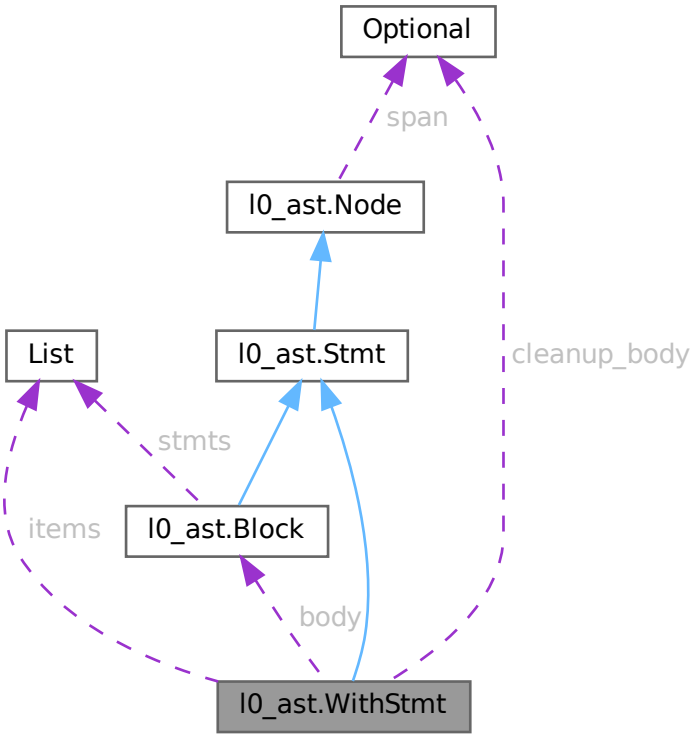
7.213 l0_ast.WithStmt Class Reference

A with statement for RAII-style resource management.

Inheritance diagram for l0_ast.WithStmt:



Collaboration diagram for l0_ast.WithStmt:



Static Public Attributes

- List items ["WithItem"]
- Block body
- Optional cleanup_body [Block]

Static Public Attributes inherited from l0_ast.Node

- Optional span = field(default=None, repr=False, compare=False, kw_only=True)

7.213.1 Detailed Description

A with statement for RAII-style resource management.

Attributes

- `items`: List of items to manage.
- `body`: The main block of the statement.
- `cleanup_body`: Optional block to execute for cleanup (mutually exclusive with per-item cleanup statements).

7.213.2 Member Data Documentation

7.213.2.1 body

```
Block l0_ast.WithStmt.body  [static]
```

7.213.2.2 cleanup_body

```
Optional l0_ast.WithStmt.cleanup_body [Block]  [static]
```

7.213.2.3 items

```
List l0_ast.WithStmt.items ["WithItem"]  [static]
```

The documentation for this class was generated from the following file:

- compiler/stage1_py/l0_ast.py

Chapter 8

File Documentation

8.1 compiler/shared/l0/stdlib/std/array.l0 File Reference

Module: `std.array`

```
import std.string;
import std.assert;
import sys.memory;
import sys.rt;
```

Classes

- struct ArrayBase

  ArrayBase implements an array of elements of a given size.
- struct ByteArray

  Byte-specialized fixed-size array wrapper.

Functions

- func arr_create(element_size: int, length: int) → ArrayBase∗

  Creates a new ArrayBase with the specified element size and length.
- func arr_check(self: ArrayBase∗, index: int)

  Checks if the given index is within the bounds of the array.
- func arr_resize(self: ArrayBase∗, new_length: int)

  Resizes the array to a new length.
- func arr_get(self: ArrayBase∗, index: int) → void∗

  Returns a pointer to the element at the specified index.
- func arr_zap(self: ArrayBase∗, index: int)

  Zeros out the element at the specified index.
- func arr_free(self: ArrayBase∗)

  Free an `ArrayBase` and its element storage.
- func ba_create(length: int) → ByteArray∗

  Create a fixed-size byte array.
- func ba_capacity(self: ByteArray∗) → int

Return the capacity of a byte array.

- func ba_get(self: ByteArray∗, index: int) → byte

  Return one byte from the array.

- func ba_set(self: ByteArray∗, index: int, value: byte)

  Store one byte into the array.

- func ba_zap(self: ByteArray∗, index: int)

  Zero one byte slot in the array.

- func ba_free(self: ByteArray∗)

  Free a byte array and its backing storage.

8.1.1 Function Documentation

8.1.1.1 arr_check()

```
func arr_check(self:  ArrayBase*, index:  int)
```

Checks if the given index is within the bounds of the array.

Parameters

self	The pointer to the ArrayBase.
index	The index to check.

8.1.1.2 arr_create()

```
func arr_create(element_size:  int, length:  int) → ArrayBase*
```

Creates a new ArrayBase with the specified element size and length.

Parameters

element_size	The size of each element in bytes.
length	The number of elements in the array.

Returns

A pointer to the newly created ArrayBase.

8.1.1.3 arr_free()

```
func arr_free(self:  ArrayBase*)
```

Free an `ArrayBase` and its element storage.

Parameters

self	Array to free.

8.1.1.4 arr_get()

```
func arr_get(self:  ArrayBase*, index:  int) → void*
```

Returns a pointer to the element at the specified index.

Parameters

self	The pointer to the ArrayBase.
index	The index of the element.

Returns

A pointer to the element.

8.1.1.5 arr_resize()

```
func arr_resize(self:  ArrayBase*, new_length:  int)
```

Resizes the array to a new length.

Parameters

self	The pointer to the ArrayBase.
new_length	The new length of the array.

8.1.1.6 arr_zap()

```
func arr_zap(self:  ArrayBase*, index:  int)
```

Zeros out the element at the specified index.

Parameters

self	The pointer to the ArrayBase.
index	The index of the element to zap.

8.1.1.7 ba_capacity()

```
func ba_capacity(self:  ByteArray*) → int
```

Return the capacity of a byte array.

Parameters

self	Byte array to inspect.

Returns

Number of byte slots.

8.1.1.8 ba_create()

```
func ba_create(length:  int) → ByteArray*
```

Create a fixed-size byte array.

Parameters

length	Number of bytes to allocate.

Returns

New byte array wrapper.

8.1.1.9 ba_free()

```
func ba_free(self:  ByteArray*)
```

Free a byte array and its backing storage.

Parameters

self	Byte array to free.

8.1.1.10 ba_get()

```
func ba_get(self:  ByteArray*, index:  int) → byte
```

Return one byte from the array.

Parameters

self	Byte array to inspect.
index	Byte index to read.

Returns

Byte value at `index`.

8.1.1.11 ba_set()

```
func ba_set(self:  ByteArray*, index:  int, value:  byte)
```

Store one byte into the array.

Parameters

self	Byte array to update.
index	Byte index to write.
value	Byte value to store.

8.1.1.12 ba_zap()

```
func ba_zap(self:  ByteArray*, index:  int)
```

Zero one byte slot in the array.

Parameters

self	Byte array to update.
index	Byte index to clear.

8.2 compiler/shared/l0/stdlib/std/assert.l0 File Reference

Module: `std.assert`

```
import sys.rt;
```

Functions

- func assert(cond: bool, msg: string)
  Asserts that the given condition is true.

8.2.1 Function Documentation

8.2.1.1 assert()

```
func assert(cond:  bool, msg:  string)
```

Asserts that the given condition is true.

If the condition is false, aborts the program with the provided message.

Parameters

cond	The condition to evaluate.
msg	The message to display if the assertion fails.

8.3 compiler/shared/l0/stdlib/std/fs.l0 File Reference

Module: `std.fs`

```
import std.unit;
import sys.rt;
```

Classes

- struct FileInfo

  Shared file metadata wrapper around `sys.rt::RtFileInfo`.

Functions

- func _fs_from_rt(info: RtFileInfo) → FileInfo

  Convert one runtime metadata record into the public stdlib wrapper type.
- func stat(path: string) → FileInfo

  Return metadata for a filesystem path.
- func exists(path: string) → bool

  Return whether any filesystem object exists at a path.
- func is_file(path: string) → bool

  Return whether a path exists and is a regular file.
- func is_dir(path: string) → bool

  Return whether a path exists and is a directory.
- func file_size(path: string) → int?

  Return the size of a file when available.
- func mtime_sec(path: string) → int?

  Return the modification time in whole seconds when available.
- func delete_file(path: string) → Unit?

  Delete the file at the given path.
- func read_file(path: string) → string?

  Read the entire content of a file.
- func write_file(path: string, data: string) → Unit?

  Write the entire content of a file.

8.3.1 Function Documentation

8.3.1.1 _fs_from_rt()

```
func _fs_from_rt(info: RtFileInfo) → FileInfo
```

Convert one runtime metadata record into the public stdlib wrapper type.

Parameters

info	Runtime metadata record.

Returns

Public stdlib metadata record.

8.3.1.2 delete_file()

```
func delete_file(path:  string) → Unit?
```

Delete the file at the given path.

Parameters

path	File path to remove.

Returns

present() on success, or null on error.

8.3.1.3 exists()

```
func exists(path:  string) → bool
```

Return whether any filesystem object exists at a path.

Parameters

path	Filesystem path to inspect.

Returns

`true` when the path exists.

8.3.1.4 file_size()

```
func file_size(path:  string) → int?
```

Return the size of a file when available.

Parameters

path	Filesystem path to inspect.

Returns

File size in bytes, or `null` when unavailable.

8.3.1.5 is_dir()

```
func is_dir(path:  string) → bool
```

Return whether a path exists and is a directory.

Parameters

path	Filesystem path to inspect.

Returns

`true` when `path` is a directory.

8.3.1.6 is_file()

```
func is_file(path:  string) → bool
```

Return whether a path exists and is a regular file.

Parameters

path	Filesystem path to inspect.

Returns

`true` when `path` is a file.

8.3.1.7 mtime_sec()

```
func mtime_sec(path:  string) → int?
```

Return the modification time in whole seconds when available.

Parameters

path	Filesystem path to inspect.

Returns

Unix modification time in seconds, or `null` when unavailable.

8.3.1.8 read_file()

```
func read_file(path:  string) → string?
```

Read the entire content of a file.

Parameters

path	File path to read.

Returns

File contents, or null on error.

8.3.1.9 stat()

```
func stat(path:  string) → FileInfo
```

Return metadata for a filesystem path.

Parameters

path	Filesystem path to inspect.

Returns

Metadata record with nullable size and timestamp fields.

8.3.1.10 write_file()

```
func write_file(path:  string, data:  string) → Unit?
```

Write the entire content of a file.

Parameters

path	File path to write.
data	String content to write.

Returns

present() on success, or null on error.

8.4 compiler/shared/l0/stdlib/std/hashmap.l0 File Reference

Module: `std.hashmap`

```
import std.vector;
import std.array;
import sys.hash;
import std.string;
import std.assert;
import sys.memory;
import sys.rt;
```

Classes

- struct StringPtrMap

  StringPtrMap is an open-addressing hash table with linear probing.
- struct StringIntMap

  StringIntMap stores associations between strings and integers.

Functions

- func _hm_slot(h: int, cap: int) → int

  Compute a non-negative slot index from a hash and a power-of-2 capacity.
- func spm_create() → StringPtrMap∗

  Creates a new StringPtrMap with default capacity.
- func spm_create_with_capacity(min_cap: int) → StringPtrMap∗

  Creates a new StringPtrMap with at least the specified capacity.
- func _spm_alloc(cap: int) → StringPtrMap∗

  Internal allocator for StringPtrMap.
- func _spm_get_state(self: StringPtrMap∗, i: int) → byte

  Returns the state of the specified slot.
- func _spm_set_state(self: StringPtrMap∗, i: int, s: byte)

  Sets the state of the specified slot.
- func _spm_get_hash(self: StringPtrMap∗, i: int) → int

  Returns the hash value of the specified slot.
- func _spm_set_hash(self: StringPtrMap∗, i: int, h: int)

  Sets the hash value of the specified slot.
- func _spm_get_key(self: StringPtrMap∗, i: int) → string

  Returns the key string of the specified slot.
- func _spm_set_key_retain(self: StringPtrMap∗, i: int, k: string)

  Sets the key string of the specified slot and retains it.
- func _spm_release_key(self: StringPtrMap∗, i: int)

  Releases the key string of the specified slot and zaps it.
- func _spm_get_value(self: StringPtrMap∗, i: int) → void∗

  Returns the value associated with the specified slot.
- func _spm_set_value(self: StringPtrMap∗, i: int, v: void∗)

  Sets the value associated with the specified slot.
- func _spm_find(self: StringPtrMap∗, key: string, h: int) → int

  Find the slot index of an existing key.
- func _spm_find_insert(self: StringPtrMap∗, h: int) → int

  Find the first non-occupied slot for insertion.
- func _spm_needs_grow(self: StringPtrMap∗) → bool

  Checks if the map needs to grow.
- func _spm_rehash(self: StringPtrMap∗)

  Rehashes the map to a larger capacity.
- func spm_put(self: StringPtrMap∗, key: string, value: void∗)

  Insert or update a key-value pair.
- func spm_get(self: StringPtrMap∗, key: string) → void∗?

  Look up a key.
- func spm_has(self: StringPtrMap∗, key: string) → bool

  Check whether a key exists in the map.
- func spm_remove(self: StringPtrMap∗, key: string) → bool

Remove a key.

- func spm_size(self: StringPtrMap*) → int

  Returns the number of entries in the map.
- func spm_capacity(self: StringPtrMap*) → int

  Returns the current capacity of the map.
- func spm_clear(self: StringPtrMap*)

  Remove all entries.
- func spm_keys(self: StringPtrMap*) → StringVector*

  Collect all keys into a new StringVector.
- func spm_free(self: StringPtrMap*)

  Free the map and all backing storage.
- func spm_slot_occupied(self: StringPtrMap*, i: int) → bool

  Checks if a slot is occupied.
- func spm_slot_key(self: StringPtrMap*, i: int) → string

  Returns the key at the specified slot.
- func spm_slot_value(self: StringPtrMap*, i: int) → void*

  Returns the value at the specified slot.
- func sim_create() → StringIntMap*

  Creates a new StringIntMap with default capacity.
- func sim_create_with_capacity(min_cap: int) → StringIntMap*

  Creates a new StringIntMap with at least the specified capacity.
- func _sim_alloc(cap: int) → StringIntMap*

  Internal allocator for StringIntMap.
- func _sim_get_state(self: StringIntMap*, i: int) → byte

  Returns the state of the specified slot.
- func _sim_set_state(self: StringIntMap*, i: int, s: byte)

  Sets the state of the specified slot.
- func _sim_get_hash(self: StringIntMap*, i: int) → int

  Returns the hash value of the specified slot.
- func _sim_set_hash(self: StringIntMap*, i: int, h: int)

  Sets the hash value of the specified slot.
- func _sim_get_key(self: StringIntMap*, i: int) → string

  Returns the key string of the specified slot.
- func _sim_set_key_retain(self: StringIntMap*, i: int, k: string)

  Sets the key string of the specified slot and retains it.
- func _sim_release_key(self: StringIntMap*, i: int)

  Releases the key string of the specified slot and zaps it.
- func _sim_get_value(self: StringIntMap*, i: int) → int

  Returns the value associated with the specified slot.
- func _sim_set_value(self: StringIntMap*, i: int, v: int)

  Sets the value associated with the specified slot.
- func _sim_find(self: StringIntMap*, key: string, h: int) → int

  Find the slot index of an existing key.
- func _sim_find_insert(self: StringIntMap*, h: int) → int

  Find the first non-occupied slot for insertion.
- func _sim_needs_grow(self: StringIntMap*) → bool

  Checks if the map needs to grow.
- func _sim_rehash(self: StringIntMap*)

  Rehashes the map to a larger capacity.
- func sim_put(self: StringIntMap*, key: string, value: int)

  Insert or update a key-value pair.

- func sim_get(self: StringIntMap∗, key: string) → int?

  Look up a key.
- func sim_has(self: StringIntMap∗, key: string) → bool

  Check whether a key exists in the map.
- func sim_remove(self: StringIntMap∗, key: string) → bool

  Remove a key.
- func sim_size(self: StringIntMap∗) → int

  Returns the number of entries in the map.
- func sim_capacity(self: StringIntMap∗) → int

  Returns the current capacity of the map.
- func sim_clear(self: StringIntMap∗)

  Remove all entries.
- func sim_keys(self: StringIntMap∗) → StringVector∗

  Collect all keys into a new StringVector.
- func sim_free(self: StringIntMap∗)

  Free the map and all backing storage.
- func sim_slot_occupied(self: StringIntMap∗, i: int) → bool

  Checks if a slot is occupied.
- func sim_slot_key(self: StringIntMap∗, i: int) → string

  Returns the key at the specified slot.
- func sim_slot_value(self: StringIntMap∗, i: int) → int

  Returns the value at the specified slot.

Variables

- let HM_EMPTY: byte = 0
- let HM_OCCUPIED: byte = 1
- let HM_TOMBSTONE: byte = 2
- let HM_DEFAULT_CAP: int = 16

8.4.1 Function Documentation

8.4.1.1 _hm_slot()

```
func _hm_slot(h:  int, cap:  int) → int
```

Compute a non-negative slot index from a hash and a power-of-2 capacity.

Parameters

h	The hash value.
cap	The capacity of the hash table.

Returns

The calculated slot index.

8.4.1.2 _sim_alloc()

```
func _sim_alloc(cap:  int) → StringIntMap*
```

Internal allocator for StringIntMap.

Parameters

cap	The capacity to allocate.

Returns

A pointer to the newly allocated StringIntMap.

8.4.1.3 _sim_find()

```
func _sim_find(self:  StringIntMap*, key:  string, h:  int) → int
```

Find the slot index of an existing key.

Parameters

self	The pointer to the StringIntMap.
key	The key string to find.
h	The hash value of the key.

Returns

The slot index if found, or -1 if not in the map.

8.4.1.4 _sim_find_insert()

```
func _sim_find_insert(self:  StringIntMap*, h:  int) → int
```

Find the first non-occupied slot for insertion.

Parameters

self	The pointer to the StringIntMap.
h	The hash value of the key.

Returns

The available slot index.

8.4.1.5 _sim_get_hash()

```
func _sim_get_hash(self:  StringIntMap*, i:  int) → int
```

Returns the hash value of the specified slot.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.

Returns

The hash value of the slot.

8.4.1.6 _sim_get_key()

```
func _sim_get_key(self:  StringIntMap*, i:  int) → string
```

Returns the key string of the specified slot.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.

Returns

The key string of the slot.

8.4.1.7 _sim_get_state()

```
func _sim_get_state(self:  StringIntMap*, i:  int) → byte
```

Returns the state of the specified slot.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.

Returns

The state byte of the slot.

8.4.1.8 _sim_get_value()

```
func _sim_get_value(self:  StringIntMap*, i:  int) → int
```

Returns the value associated with the specified slot.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.

Returns

The value of the slot.

8.4.1.9 _sim_needs_grow()

```
func _sim_needs_grow(self:  StringIntMap*) → bool
```

Checks if the map needs to grow.

Parameters

self	The pointer to the StringIntMap.

Returns

True if growth is needed, false otherwise.

8.4.1.10 _sim_rehash()

```
func _sim_rehash(self:  StringIntMap*)
```

Rehashes the map to a larger capacity.

Parameters

self	The pointer to the StringIntMap.

8.4.1.11 _sim_release_key()

```
func _sim_release_key(self:  StringIntMap*, i:  int)
```

Releases the key string of the specified slot and zaps it.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.

8.4.1.12 _sim_set_hash()

```
func _sim_set_hash(self:  StringIntMap*, i:  int, h:  int)
```

Sets the hash value of the specified slot.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.
h	The hash value to set.

8.4.1.13 _sim_set_key_retain()

```
func _sim_set_key_retain(self:  StringIntMap*, i:  int, k:  string)
```

Sets the key string of the specified slot and retains it.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.
k	The key string to set.

8.4.1.14 _sim_set_state()

```
func _sim_set_state(self:  StringIntMap*, i:  int, s:  byte)
```

Sets the state of the specified slot.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.
s	The state byte to set.

8.4.1.15 _sim_set_value()

```
func _sim_set_value(self:  StringIntMap*, i:  int, v:  int)
```

Sets the value associated with the specified slot.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.
v	The value to set.

8.4.1.16 _spm_alloc()

```
func _spm_alloc(cap:  int) → StringPtrMap*
```

Internal allocator for StringPtrMap.

Parameters

cap	The capacity to allocate.

Returns

A pointer to the newly allocated StringPtrMap.

8.4.1.17 _spm_find()

```
func _spm_find(self:  StringPtrMap*, key:  string, h:  int) → int
```

Find the slot index of an existing key.

Parameters

self	The pointer to the StringPtrMap.
key	The key string to find.
h	The hash value of the key.

Returns

The slot index if found, or -1 if not in the map.

8.4.1.18 _spm_find_insert()

```
func _spm_find_insert(self:  StringPtrMap*, h:  int) → int
```

Find the first non-occupied slot for insertion.

Parameters

self	The pointer to the StringPtrMap.
h	The hash value of the key.

Returns

The available slot index.

8.4.1.19 _spm_get_hash()

```
func _spm_get_hash(self:  StringPtrMap*, i:  int) → int
```

Returns the hash value of the specified slot.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.

Returns

The hash value of the slot.

8.4.1.20 _spm_get_key()

```
func _spm_get_key(self:  StringPtrMap*, i:  int) → string
```

Returns the key string of the specified slot.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.

Returns

The key string of the slot.

8.4.1.21 _spm_get_state()

```
func _spm_get_state(self:  StringPtrMap*, i:  int) → byte
```

Returns the state of the specified slot.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.

Returns

The state byte of the slot.

8.4.1.22 _spm_get_value()

```
func _spm_get_value(self:  StringPtrMap*, i:  int) → void*
```

Returns the value associated with the specified slot.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.

Returns

The value pointer of the slot.

8.4.1.23 _spm_needs_grow()

```
func _spm_needs_grow(self: StringPtrMap*) → bool
```

Checks if the map needs to grow.

Parameters

self	The pointer to the StringPtrMap.

Returns

True if growth is needed, false otherwise.

8.4.1.24 _spm_rehash()

```
func _spm_rehash(self: StringPtrMap*)
```

Rehashes the map to a larger capacity.

Parameters

self	The pointer to the StringPtrMap.

8.4.1.25 _spm_release_key()

```
func _spm_release_key(self: StringPtrMap*, i: int)
```

Releases the key string of the specified slot and zaps it.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.

8.4.1.26 _spm_set_hash()

```
func _spm_set_hash(self:  StringPtrMap*, i:  int, h:  int)
```

Sets the hash value of the specified slot.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.
h	The hash value to set.

8.4.1.27 _spm_set_key_retain()

```
func _spm_set_key_retain(self:  StringPtrMap*, i:  int, k:  string)
```

Sets the key string of the specified slot and retains it.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.
k	The key string to set.

8.4.1.28 _spm_set_state()

```
func _spm_set_state(self:  StringPtrMap*, i:  int, s:  byte)
```

Sets the state of the specified slot.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.
s	The state byte to set.

8.4.1.29 _spm_set_value()

```
func _spm_set_value(self:  StringPtrMap*, i:  int, v:  void*)
```

Sets the value associated with the specified slot.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.
v	The value pointer to set.

8.4.1.30 sim_capacity()

```
func sim_capacity(self:  StringIntMap*) → int
```

Returns the current capacity of the map.

Parameters

self	The pointer to the StringIntMap.

Returns

The capacity.

8.4.1.31 sim_clear()

```
func sim_clear(self:  StringIntMap*)
```

Remove all entries.

Releases all key strings.

Parameters

self	The pointer to the StringIntMap.

8.4.1.32 sim_create()

```
func sim_create() → StringIntMap*
```

Creates a new StringIntMap with default capacity.

Returns

A pointer to the newly created StringIntMap.

8.4.1.33 sim_create_with_capacity()

```
func sim_create_with_capacity(min_cap:  int) → StringIntMap*
```

Creates a new StringIntMap with at least the specified capacity.

Parameters

min_cap	The minimum capacity for the map.

Returns

A pointer to the newly created StringIntMap.

8.4.1.34 sim_free()

```
func sim_free(self:  StringIntMap*)
```

Free the map and all backing storage.

Releases all key strings.

Parameters

self	The pointer to the StringIntMap to free.

8.4.1.35 sim_get()

```
func sim_get(self:  StringIntMap*, key:  string) → int?
```

Look up a key.

Returns the associated value, or null if not found.

Parameters

self	The pointer to the StringIntMap.
key	The key string to look up.

Returns

The associated value, or null if not found.

8.4.1.36 sim_has()

```
func sim_has(self:  StringIntMap*, key:  string) → bool
```

Check whether a key exists in the map.

Parameters

self	The pointer to the StringIntMap.
key	The key string to check.

Returns

True if the key exists, false otherwise.

8.4.1.37 sim_keys()

```
func sim_keys(self:  StringIntMap*) → StringVector*
```

Collect all keys into a new StringVector.

Caller owns the result and must sv_free() it.

Parameters

self	The pointer to the StringIntMap.

Returns

A new StringVector containing all keys.

8.4.1.38 sim_put()

```
func sim_put(self:  StringIntMap*, key:  string, value:  int)
```

Insert or update a key-value pair.

Key is retained on first insertion. On update, only the value changes.

Parameters

self	The pointer to the StringIntMap.
key	The key string.
value	The value to set.

8.4.1.39 sim_remove()

```
func sim_remove(self:  StringIntMap*, key:  string) → bool
```

Remove a key.

Returns true if the key was found and removed.

Parameters

self	The pointer to the StringIntMap.
key	The key string to remove.

Returns

True if found and removed, false otherwise.

8.4.1.40 sim_size()

```
func sim_size(self:  StringIntMap*) → int
```

Returns the number of entries in the map.

Parameters

self	The pointer to the StringIntMap.

Returns

The count of elements.

8.4.1.41 sim_slot_key()

```
func sim_slot_key(self:  StringIntMap*, i:  int) → string
```

Returns the key at the specified slot.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.

Returns

The key string.

8.4.1.42 sim_slot_occupied()

```
func sim_slot_occupied(self:  StringIntMap*, i:  int) → bool
```

Checks if a slot is occupied.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.

Returns

True if occupied, false otherwise.

8.4.1.43 sim_slot_value()

```
func sim_slot_value(self:  StringIntMap*, i:  int) → int
```

Returns the value at the specified slot.

Parameters

self	The pointer to the StringIntMap.
i	The slot index.

Returns

The value.

8.4.1.44 spm_capacity()

```
func spm_capacity(self:  StringPtrMap*) → int
```

Returns the current capacity of the map.

Parameters

self	The pointer to the StringPtrMap.

Returns

The capacity.

8.4.1.45 spm_clear()

```
func spm_clear(self:  StringPtrMap*)
```

Remove all entries.

Releases all key strings.

Parameters

self	The pointer to the StringPtrMap.

8.4.1.46 spm_create()

```
func spm_create() → StringPtrMap*
```

Creates a new StringPtrMap with default capacity.

Returns

A pointer to the newly created StringPtrMap.

8.4.1.47 spm_create_with_capacity()

```
func spm_create_with_capacity(min_cap:  int) → StringPtrMap*
```

Creates a new StringPtrMap with at least the specified capacity.

Parameters

min_cap	The minimum capacity for the map.

Returns

A pointer to the newly created StringPtrMap.

8.4.1.48 spm_free()

```
func spm_free(self:  StringPtrMap*)
```

Free the map and all backing storage.

Releases all key strings. Does NOT free values (caller-managed).

Parameters

self	The pointer to the StringPtrMap to free.

8.4.1.49 spm_get()

```
func spm_get(self:  StringPtrMap*, key:  string) → void*?
```

Look up a key.

Returns the associated value, or null if not found.

Note: if null is a valid stored value, use spm_has() to disambiguate.

Parameters

self	The pointer to the StringPtrMap.
key	The key string to look up.

Returns

The associated value pointer, or null if not found.

8.4.1.50 spm_has()

```
func spm_has(self:  StringPtrMap*, key:  string) → bool
```

Check whether a key exists in the map.

Parameters

self	The pointer to the StringPtrMap.
key	The key string to check.

Returns

True if the key exists, false otherwise.

8.4.1.51 spm_keys()

```
func spm_keys(self:  StringPtrMap*) → StringVector*
```

Collect all keys into a new StringVector.

Caller owns the result and must sv_free() it.

Parameters

self	The pointer to the StringPtrMap.

Returns

A new StringVector containing all keys.

8.4.1.52 spm_put()

```
func spm_put(self:  StringPtrMap*, key:  string, value:  void*)
```

Insert or update a key-value pair.

Key is retained on first insertion. On update, only the value changes.

Parameters

self	The pointer to the StringPtrMap.
key	The key string.
value	The value pointer.

8.4.1.53 spm_remove()

```
func spm_remove(self:  StringPtrMap*, key:  string) → bool
```

Remove a key.

Returns true if the key was found and removed.

Parameters

self	The pointer to the StringPtrMap.
key	The key string to remove.

Returns

True if found and removed, false otherwise.

8.4.1.54 spm_size()

```
func spm_size(self:  StringPtrMap*) → int
```

Returns the number of entries in the map.

Parameters

self	The pointer to the StringPtrMap.

Returns

The count of elements.

8.4.1.55 spm_slot_key()

```
func spm_slot_key(self:  StringPtrMap*, i:  int) → string
```

Returns the key at the specified slot.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.

Returns

The key string.

8.4.1.56 spm_slot_occupied()

```
func spm_slot_occupied(self:  StringPtrMap*, i:  int) → bool
```

Checks if a slot is occupied.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.

Returns

True if occupied, false otherwise.

8.4.1.57 spm_slot_value()

```
func spm_slot_value(self:  StringPtrMap*, i:  int) → void*
```

Returns the value at the specified slot.

Parameters

self	The pointer to the StringPtrMap.
i	The slot index.

Returns

The value pointer.

8.4.2 Variable Documentation

8.4.2.1 HM_DEFAULT_CAP

```
let HM_DEFAULT_CAP: int = 16
```

8.4.2.2 HM_EMPTY

```
let HM_EMPTY: byte = 0
```

8.4.2.3 HM_OCCUPIED

```
let HM_OCCUPIED: byte = 1
```

8.4.2.4 HM_TOMBSTONE

```
let HM_TOMBSTONE: byte = 2
```

8.5 compiler/shared/l0/stdlib/std/hashset.l0 File Reference

Module: `std.hashset`

```
import std.vector;
import std.array;
import std.string;
import std.assert;
import sys.hash;
import sys.memory;
import sys.rt;
```

Classes

- struct StringSet

  StringSet is an open-addressing hash set with linear probing.

Functions

- func _hs_slot(h: int, cap: int) → int

  Compute a non-negative slot index from a hash and a power-of-2 capacity.
- func ss_create() → StringSet∗

  Creates a new StringSet with default capacity.
- func ss_create_with_capacity(min_cap: int) → StringSet∗

  Creates a new StringSet with at least the specified capacity.
- func _ss_alloc(cap: int) → StringSet∗

  Internal allocator for StringSet.
- func _ss_get_state(self: StringSet∗, i: int) → byte

  Returns the state of the specified slot.
- func _ss_set_state(self: StringSet∗, i: int, s: byte)

  Sets the state of the specified slot.
- func _ss_get_hash(self: StringSet∗, i: int) → int

  Returns the hash value of the specified slot.
- func _ss_set_hash(self: StringSet∗, i: int, h: int)

  Sets the hash value of the specified slot.
- func _ss_get_key(self: StringSet∗, i: int) → string

  Returns the key string of the specified slot.
- func _ss_set_key_retain(self: StringSet∗, i: int, k: string)

  Sets the key string of the specified slot and retains it.
- func _ss_release_key(self: StringSet∗, i: int)

  Releases the key string of the specified slot and zaps it.
- func _ss_find(self: StringSet∗, key: string, h: int) → int

  Find the slot index of an existing key.
- func _ss_find_insert(self: StringSet∗, h: int) → int

  Find the first non-occupied slot for insertion.
- func _ss_needs_grow(self: StringSet∗) → bool

  Checks if the set needs to grow.
- func _ss_rehash(self: StringSet∗)

  Rehashes the set to a larger capacity.

- func ss_add(self: StringSet*, key: string) → bool

  Add a string to the set.
- func ss_has(self: StringSet*, key: string) → bool

  Check whether a string is in the set.
- func ss_remove(self: StringSet*, key: string) → bool

  Remove a string from the set.
- func ss_size(self: StringSet*) → int

  Returns the number of entries in the set.
- func ss_capacity(self: StringSet*) → int

  Returns the current capacity of the set.
- func ss_clear(self: StringSet*)

  Remove all entries from the set.
- func ss_to_vector(self: StringSet*) → StringVector*

  Collect all elements into a new StringVector.
- func ss_free(self: StringSet*)

  Free the set and all backing storage.
- func ss_slot_occupied(self: StringSet*, i: int) → bool

  Checks if a slot is occupied.
- func ss_slot_key(self: StringSet*, i: int) → string

  Returns the key at the specified slot.

Variables

- let HS_EMPTY: byte = 0
- let HS_OCCUPIED: byte = 1
- let HS_TOMBSTONE: byte = 2
- let HS_DEFAULT_CAP: int = 16

8.5.1 Function Documentation

8.5.1.1 _hs_slot()

```
func _hs_slot(h:  int, cap:  int) → int
```

Compute a non-negative slot index from a hash and a power-of-2 capacity.

Parameters

h	The hash value.
cap	The capacity of the hash set.

Returns

The calculated slot index.

8.5.1.2 _ss_alloc()

```
func _ss_alloc(cap:  int) → StringSet*
```

Internal allocator for StringSet.

Parameters

cap	The capacity to allocate.

Returns

A pointer to the newly allocated StringSet.

8.5.1.3 _ss_find()

```
func _ss_find(self:  StringSet*, key:  string, h:  int) → int
```

Find the slot index of an existing key.

Parameters

self	The pointer to the StringSet.
key	The key string to find.
h	The hash value of the key.

Returns

The slot index if found, or -1 if not in the set.

8.5.1.4 _ss_find_insert()

```
func _ss_find_insert(self:  StringSet*, h:  int) → int
```

Find the first non-occupied slot for insertion.

Parameters

self	The pointer to the StringSet.
h	The hash value of the key.

Returns

The available slot index.

8.5.1.5 _ss_get_hash()

```
func _ss_get_hash(self:  StringSet*, i:  int) → int
```

Returns the hash value of the specified slot.

Parameters

self	The pointer to the StringSet.
i	The slot index.

Returns

The hash value of the slot.

8.5.1.6 _ss_get_key()

```
func _ss_get_key(self:  StringSet*, i:  int) → string
```

Returns the key string of the specified slot.

Parameters

self	The pointer to the StringSet.
i	The slot index.

Returns

The key string of the slot.

8.5.1.7 _ss_get_state()

```
func _ss_get_state(self:  StringSet*, i:  int) → byte
```

Returns the state of the specified slot.

Parameters

self	The pointer to the StringSet.
i	The slot index.

Returns

The state byte of the slot.

8.5.1.8 _ss_needs_grow()

```
func _ss_needs_grow(self:  StringSet*) → bool
```

Checks if the set needs to grow.

Parameters

self	The pointer to the StringSet.

Returns

True if growth is needed, false otherwise.

8.5.1.9 _ss_rehash()

```
func _ss_rehash(self:  StringSet*)
```

Rehashes the set to a larger capacity.

Parameters

self	The pointer to the StringSet.

8.5.1.10 _ss_release_key()

```
func _ss_release_key(self:  StringSet*, i:  int)
```

Releases the key string of the specified slot and zaps it.

Parameters

self	The pointer to the StringSet.
i	The slot index.

8.5.1.11 _ss_set_hash()

```
func _ss_set_hash(self:  StringSet*, i:  int, h:  int)
```

Sets the hash value of the specified slot.

Parameters

self	The pointer to the StringSet.
i	The slot index.
h	The hash value to set.

8.5.1.12 _ss_set_key_retain()

```
func _ss_set_key_retain(self:  StringSet*, i:  int, k:  string)
```

Sets the key string of the specified slot and retains it.

Parameters

self	The pointer to the StringSet.
i	The slot index.
k	The key string to set.

8.5.1.13 _ss_set_state()

```
func _ss_set_state(self:  StringSet*, i:  int, s:  byte)
```

Sets the state of the specified slot.

Parameters

self	The pointer to the StringSet.
i	The slot index.
s	The state byte to set.

8.5.1.14 ss_add()

```
func ss_add(self:  StringSet*, key:  string) → bool
```

Add a string to the set.

Returns true if the string was newly added, false if already present.

Parameters

self	The pointer to the StringSet.
key	The key string to add.

Returns

True if the string was newly added, false otherwise.

8.5.1.15 ss_capacity()

```
func ss_capacity(self:  StringSet*) → int
```

Returns the current capacity of the set.

Parameters

self	The pointer to the StringSet.

Returns

The capacity.

8.5.1.16 ss_clear()

```
func ss_clear(self:  StringSet*)
```

Remove all entries from the set.

Releases all key strings.

Parameters

self	The pointer to the StringSet.

8.5.1.17 ss_create()

```
func ss_create() → StringSet*
```

Creates a new StringSet with default capacity.

Returns

A pointer to the newly created StringSet.

8.5.1.18 ss_create_with_capacity()

```
func ss_create_with_capacity(min_cap:  int) → StringSet*
```

Creates a new StringSet with at least the specified capacity.

Parameters

min_cap	The minimum capacity for the set.

Returns

A pointer to the newly created StringSet.

8.5.1.19 ss_free()

```
func ss_free(self:  StringSet*)
```

Free the set and all backing storage.

Releases all key strings.

Parameters

self	The pointer to the StringSet to free.

8.5.1.20 ss_has()

```
func ss_has(self:  StringSet*, key:  string) → bool
```

Check whether a string is in the set.

Parameters

self	The pointer to the StringSet.
key	The key string to check.

Returns

True if the string is in the set, false otherwise.

8.5.1.21 ss_remove()

```
func ss_remove(self:  StringSet*, key:  string) → bool
```

Remove a string from the set.

Returns true if it was found and removed.

Parameters

self	The pointer to the StringSet.
key	The key string to remove.

Returns

True if found and removed, false otherwise.

8.5.1.22 ss_size()

```
func ss_size(self:  StringSet*) → int
```

Returns the number of entries in the set.

Parameters

self	The pointer to the StringSet.

Returns

The count of elements.

8.5.1.23 ss_slot_key()

```
func ss_slot_key(self:  StringSet*, i:  int) → string
```

Returns the key at the specified slot.

Parameters

self	The pointer to the StringSet.
i	The slot index.

Returns

The key string.

8.5.1.24 ss_slot_occupied()

```
func ss_slot_occupied(self:  StringSet*, i:  int) → bool
```

Checks if a slot is occupied.

Parameters

self	The pointer to the StringSet.
i	The slot index.

Returns

True if occupied, false otherwise.

8.5.1.25 ss_to_vector()

```
func ss_to_vector(self:  StringSet*) → StringVector*
```

Collect all elements into a new StringVector.

Caller owns the result and must sv_free() it.

Parameters

self	The pointer to the StringSet.

Returns

A new StringVector containing all elements.

8.5.2 Variable Documentation

8.5.2.1 HS_DEFAULT_CAP

```
let HS_DEFAULT_CAP: int = 16
```

8.5.2.2 HS_EMPTY

```
let HS_EMPTY: byte = 0
```

8.5.2.3 HS_OCCUPIED

```
let HS_OCCUPIED: byte = 1
```

8.5.2.4 HS_TOMBSTONE

```
let HS_TOMBSTONE: byte = 2
```

8.6 compiler/shared/l0/stdlib/std/integer.l0 File Reference

Module: `std.integer`

```
import std.assert;
```

Functions

- func int_max() → int

  Returns the maximum value for a 32-bit signed integer.
- func int_min() → int

  Returns the minimum value for a 32-bit signed integer.
- func checked_add_int(a: int, b: int) → int?

  Performs checked integer addition.
- func checked_sub_int(a: int, b: int) → int?

  Performs checked integer subtraction.
- func checked_mul_int(a: int, b: int) → int?

  Performs checked integer multiplication.
- func emod(a: int, b: int) → int

  Calculates the Euclidean modulo: always returns a non-negative result, even if `a` is negative.
- func ediv(a: int, b: int) → int

  Calculates the Euclidean quotient paired with `emod`.

- func div_floor(a: int, b: int) → int

  Calculates the mathematical floor of `a / b`.
- func div_ceil(a: int, b: int) → int

  Calculates the mathematical ceiling of `a / b`.
- func min(a: int, b: int) → int

  Returns the smaller of two integers.
- func max(a: int, b: int) → int

  Returns the larger of two integers.
- func clamp(x: int, lo: int, hi: int) → int

  Clamps `x` *into the closed interval* `[lo, hi]`.
- func sign(x: int) → int

  Returns the sign of `x` *as* `-1`, `0`, *or* `1`.
- func is_even(x: int) → bool

  Returns whether `x` *is even.*
- func is_odd(x: int) → bool

  Returns whether `x` *is odd.*
- func is_multiple(a: int, b: int) → bool

  Returns whether `a` *is a multiple of* `b`.
- func abs(x: int) → int?

  Returns the absolute value of `x`.
- func gcd(a: int, b: int) → int?

  Returns the non-negative greatest common divisor of `a` *and* `b`.
- func lcm(a: int, b: int) → int?

  Returns the non-negative least common multiple of `a` *and* `b`.
- func pow(base: int, exp: int) → int?

  Returns `base` *raised to the non-negative exponent* `exp`.
- func isqrt(x: int) → int?

  Returns the floor integer square root of `x`.
- func align_down(x: int, align: int) → int?

  Rounds `x` *down to the nearest multiple of* `align`.
- func align_up(x: int, align: int) → int?

  Rounds `x` *up to the nearest multiple of* `align`.
- func is_aligned(x: int, align: int) → bool

  Returns whether `x` *is already aligned to* `align`.

8.6.1 Function Documentation

8.6.1.1 abs()

```
func abs(x:  int) → int?
```

Returns the absolute value of `x`.

Parameters

x	The input value.

Returns

The absolute value, or null when it is not representable as `int`.

8.6.1.2 align_down()

```
func align_down(x:  int, align:  int) → int?
```

Rounds `x` down to the nearest multiple of `align`.

Parameters

x	The input value.
align	The positive alignment.

Returns

The aligned value, or null when the mathematical result is not representable as `int`.

8.6.1.3 align_up()

```
func align_up(x:  int, align:  int) → int?
```

Rounds `x` up to the nearest multiple of `align`.

Parameters

x	The input value.
align	The positive alignment.

Returns

The aligned value, or null when the mathematical result is not representable as `int`.

8.6.1.4 checked_add_int()

```
func checked_add_int(a:  int, b:  int) → int?
```

Performs checked integer addition.

Parameters

a	The first operand.
b	The second operand.

Returns

The sum, or null on overflow.

8.6.1.5 checked_mul_int()

```
func checked_mul_int(a:  int, b:  int) → int?
```

Performs checked integer multiplication.

Parameters

a	The first operand.
b	The second operand.

Returns

The product, or null on overflow.

8.6.1.6 checked_sub_int()

```
func checked_sub_int(a:  int, b:  int) → int?
```

Performs checked integer subtraction.

Parameters

a	The first operand.
b	The second operand.

Returns

The difference, or null on overflow.

8.6.1.7 clamp()

```
func clamp(x:  int, lo:  int, hi:  int) → int
```

Clamps `x` into the closed interval `[lo, hi]`.

Parameters

x	The value to clamp.
lo	The lower bound.
hi	The upper bound.

Returns

The clamped value.

8.6.1.8 div_ceil()

```
func div_ceil(a: int, b: int) → int
```

Calculates the mathematical ceiling of `a / b`.

The quotient must be representable as `int`.

Parameters

a	The dividend.
b	The divisor (must be non-zero).

Returns

The ceiling quotient.

8.6.1.9 div_floor()

```
func div_floor(a: int, b: int) → int
```

Calculates the mathematical floor of `a / b`.

The quotient must be representable as `int`.

Parameters

a	The dividend.
b	The divisor (must be non-zero).

Returns

The floor quotient.

8.6.1.10 ediv()

```
func ediv(a: int, b: int) → int
```

Calculates the Euclidean quotient paired with `emod`.

Parameters

a	The dividend.
b	The divisor (must be > 0).

Returns

The Euclidean quotient.

8.6.1.11 emod()

```
func emod(a:  int, b:  int) → int
```

Calculates the Euclidean modulo: always returns a non-negative result, even if `a` is negative.

Requires `b > 0`.

Parameters

a	The dividend.
b	The divisor (must be > 0).

Returns

The non-negative remainder.

8.6.1.12 gcd()

```
func gcd(a:  int, b:  int) → int?
```

Returns the non-negative greatest common divisor of `a` and `b`.

Parameters

a	The first operand.
b	The second operand.

Returns

The greatest common divisor, or null when it is not representable as `int`.

8.6.1.13 int_max()

```
func int_max() → int
```

Returns the maximum value for a 32-bit signed integer.

Returns

INT_MAX.

8.6.1.14 int_min()

```
func int_min() → int
```

Returns the minimum value for a 32-bit signed integer.

Returns

INT_MIN.

8.6.1.15 is_aligned()

```
func is_aligned(x:  int, align:  int) → bool
```

Returns whether `x` is already aligned to `align`.

Parameters

x	The input value.
align	The positive alignment.

Returns

True when `x` is a multiple of `align`.

8.6.1.16 is_even()

```
func is_even(x:  int) → bool
```

Returns whether `x` is even.

Parameters

x	The input value.

Returns

True when `x` is evenly divisible by 2.

8.6.1.17 is_multiple()

```
func is_multiple(a:  int, b:  int) → bool
```

Returns whether `a` is a multiple of `b`.

Parameters

a	The dividend.
b	The divisor (must be non-zero).

Returns

True when `a` is evenly divisible by `b`.

8.6.1.18 is_odd()

```
func is_odd(x:  int) → bool
```

Returns whether `x` is odd.

Parameters

x	The input value.

Returns

True when `x` is not evenly divisible by 2.

8.6.1.19 isqrt()

```
func isqrt(x:  int) → int?
```

Returns the floor integer square root of `x`.

Parameters

x	The input value.

Returns

The floor square root, or null when `x` $<$ `0`.

8.6.1.20 lcm()

```
func lcm(a:  int, b:  int) → int?
```

Returns the non-negative least common multiple of `a` and `b`.

Parameters

a	The first operand.
b	The second operand.

Returns

The least common multiple, or null when it is not representable as `int`.

8.6.1.21 max()

```
func max(a:  int, b:  int) → int
```

Returns the larger of two integers.

Parameters

a	The first operand.
b	The second operand.

Returns

The maximum value.

8.6.1.22 min()

```
func min(a:  int, b:  int) → int
```

Returns the smaller of two integers.

Parameters

a	The first operand.
b	The second operand.

Returns

The minimum value.

8.6.1.23 pow()

```
func pow(base:  int, exp:  int) → int?
```

Returns `base` raised to the non-negative exponent `exp`.

Parameters

base	The base value.
exp	The exponent.

Returns

The result, or null when `exp < 0` or the result is not representable as `int`.

8.6.1.24 sign()

```
func sign(x:  int) → int
```

Returns the sign of `x` as `−1`, `0`, or `1`.

Parameters

x	The input value.

Returns

The sign indicator.

8.7 compiler/shared/l0/stdlib/std/io.l0 File Reference

Module: `std.io`

```
import std.unit;
import std.assert;
import std.array;
import sys.memory;
import sys.rt;
```

Functions

- func read_line() → string?

  Read a line from standard input.
- func read_char() → int?

  Read a single character from standard input.
- func read_char_or_eof() → int

  Read a single character from standard input.
- func _io_check_byte_range(buf: ByteArray∗, start: int, count: int)

  Validate one byte-array subrange before raw I/O.
- func read_stdin_some(buf: ByteArray∗, start: int, count: int) → int?

  Read up to `count` raw bytes from standard input into one byte-array subrange.
- func write_stdout_some(buf: ByteArray∗, start: int, count: int) → int?

  Write up to `count` raw bytes from one byte-array subrange to standard output.
- func write_stderr_some(buf: ByteArray∗, start: int, count: int) → int?

  Write up to `count` raw bytes from one byte-array subrange to standard error.
- func write_stdout_all(buf: ByteArray∗, start: int, count: int) → Unit?

  Write exactly `count` raw bytes from one byte-array subrange to standard output.
- func write_stderr_all(buf: ByteArray∗, start: int, count: int) → Unit?

Write exactly `count` raw bytes from one byte-array subrange to standard error.

- func flush_stdout() → void

  Flush the standard output buffer.
- func flush_stderr() → void

  Flush the standard error buffer.
- func printl() → void

  Print a newline to standard output.
- func print_s(s: string) → void

  Print a string to standard output.
- func print_i(x: int) → void

  Print an integer to standard output.
- func print_b(x: bool) → void

  Print a boolean to standard output.
- func printl_s(s: string) → void

  Print a string to standard output followed by a newline.
- func printl_i(x: int) → void

  Print an integer to standard output followed by a newline.
- func printl_b(x: bool) → void

  Print a boolean to standard output followed by a newline.
- func print_ss(s1: string, s2: string) → void

  Print two strings to standard output separated by a space.
- func print_si(s: string, x: int) → void

  Print a string and an integer to standard output separated by a space.
- func print_sb(s: string, b: bool) → void

  Print a string and a boolean to standard output separated by a space.
- func print_is(x: int, s: string) → void

  Print an integer and a string to standard output separated by a space.
- func print_ii(x1: int, x2: int) → void

  Print two integers to standard output separated by a space.
- func print_ib(x: int, b: bool) → void

  Print an integer and a boolean to standard output separated by a space.
- func print_bs(b: bool, s: string) → void

  Print a boolean and a string to standard output separated by a space.
- func print_bi(b: bool, x: int) → void

  Print a boolean and an integer to standard output separated by a space.
- func print_bb(b1: bool, b2: bool) → void

  Print two booleans to standard output separated by a space.
- func printl_ss(s1: string, s2: string) → void

  Print two strings to standard output separated by a space, followed by a newline.
- func printl_si(s: string, x: int) → void

  Print a string and an integer to standard output separated by a space, followed by a newline.
- func printl_sb(s: string, b: bool) → void

  Print a string and a boolean to standard output separated by a space, followed by a newline.
- func printl_is(x: int, s: string) → void

  Print an integer and a string to standard output separated by a space, followed by a newline.
- func printl_ii(x1: int, x2: int) → void

  Print two integers to standard output separated by a space, followed by a newline.
- func printl_ib(x: int, b: bool) → void

  Print an integer and a boolean to standard output separated by a space, followed by a newline.
- func printl_bs(b: bool, s: string) → void

  Print a boolean and a string to standard output separated by a space, followed by a newline.

- func printl_bi(b: bool, x: int) → void

  Print a boolean and an integer to standard output separated by a space, followed by a newline.
- func printl_bb(b1: bool, b2: bool) → void

  Print two booleans to standard output separated by a space, followed by a newline.
- func err_printl() → void

  Print a newline to standard error.
- func err_print_s(s: string) → void

  Print a string to standard error.
- func err_print_i(x: int) → void

  Print an integer to standard error.
- func err_print_b(x: bool) → void

  Print a boolean to standard error.
- func err_printl_s(s: string) → void

  Print a string to standard error followed by a newline.
- func err_printl_i(x: int) → void

  Print an integer to standard error followed by a newline.
- func err_printl_b(x: bool) → void

  Print a boolean to standard error followed by a newline.
- func err_print_ss(s1: string, s2: string) → void

  Print two strings to standard error separated by a space.
- func err_print_si(s: string, x: int) → void

  Print a string and an integer to standard error separated by a space.
- func err_print_sb(s: string, b: bool) → void

  Print a string and a boolean to standard error separated by a space.
- func err_print_is(x: int, s: string) → void

  Print an integer and a string to standard error separated by a space.
- func err_print_ii(x1: int, x2: int) → void

  Print two integers to standard error separated by a space.
- func err_print_ib(x: int, b: bool) → void

  Print an integer and a boolean to standard error separated by a space.
- func err_print_bs(b: bool, s: string) → void

  Print a boolean and a string to standard error separated by a space.
- func err_print_bi(b: bool, x: int) → void

  Print a boolean and an integer to standard error separated by a space.
- func err_print_bb(b1: bool, b2: bool) → void

  Print two booleans to standard error separated by a space.

8.7.1 Function Documentation

8.7.1.1 _io_check_byte_range()

```
func _io_check_byte_range(buf:  ByteArray*, start:  int, count:  int)
```

Validate one byte-array subrange before raw I/O.

Parameters

buf	Byte array to inspect.
start	Start offset within the array.
count	Number of bytes in range.

8.7.1.2 err_print_b()

```
func err_print_b(x:  bool) → void
```

Print a boolean to standard error.

Parameters

x	The boolean to print.

8.7.1.3 err_print_bb()

```
func err_print_bb(b1:  bool, b2:  bool) → void
```

Print two booleans to standard error separated by a space.

Parameters

b1	The first boolean.
b2	The second boolean.

8.7.1.4 err_print_bi()

```
func err_print_bi(b:  bool, x:  int) → void
```

Print a boolean and an integer to standard error separated by a space.

Parameters

b	The boolean to print.
x	The integer to print.

8.7.1.5 err_print_bs()

```
func err_print_bs(b:  bool, s:  string) → void
```

Print a boolean and a string to standard error separated by a space.

Parameters

b	The boolean to print.
s	The string to print.

8.7.1.6 err_print_i()

```
func err_print_i(x:  int) → void
```

Print an integer to standard error.

Parameters

x	The integer to print.

8.7.1.7 err_print_ib()

```
func err_print_ib(x:  int, b:  bool) → void
```

Print an integer and a boolean to standard error separated by a space.

Parameters

x	The integer to print.
b	The boolean to print.

8.7.1.8 err_print_ii()

```
func err_print_ii(x1:  int, x2:  int) → void
```

Print two integers to standard error separated by a space.

Parameters

x1	The first integer.
x2	The second integer.

8.7.1.9 err_print_is()

```
func err_print_is(x:  int, s:  string) → void
```

Print an integer and a string to standard error separated by a space.

Parameters

x	The integer to print.
s	The string to print.

8.7.1.10 err_print_s()

```
func err_print_s(s:  string) → void
```

Print a string to standard error.

Parameters

s	The string to print.

8.7.1.11 err_print_sb()

```
func err_print_sb(s:  string, b:  bool) → void
```

Print a string and a boolean to standard error separated by a space.

Parameters

s	The string to print.
b	The boolean to print.

8.7.1.12 err_print_si()

```
func err_print_si(s:  string, x:  int) → void
```

Print a string and an integer to standard error separated by a space.

Parameters

s	The string to print.
x	The integer to print.

8.7.1.13 err_print_ss()

```
func err_print_ss(s1:  string, s2:  string) → void
```

Print two strings to standard error separated by a space.

Parameters

s1	The first string.
s2	The second string.

8.7.1.14 err_printl()

```
func err_printl() → void
```

Print a newline to standard error.

8.7.1.15 err_printl_b()

```
func err_printl_b(x: bool) → void
```

Print a boolean to standard error followed by a newline.

Parameters

x	The boolean to print.

8.7.1.16 err_printl_i()

```
func err_printl_i(x: int) → void
```

Print an integer to standard error followed by a newline.

Parameters

x	The integer to print.

8.7.1.17 err_printl_s()

```
func err_printl_s(s: string) → void
```

Print a string to standard error followed by a newline.

Parameters

s	The string to print.

8.7.1.18 flush_stderr()

```
func flush_stderr() → void
```

Flush the standard error buffer.

8.7.1.19 flush_stdout()

```
func flush_stdout() → void
```

Flush the standard output buffer.

8.7.1.20 print_b()

```
func print_b(x:  bool) → void
```

Print a boolean to standard output.

Parameters

x	The boolean to print.

8.7.1.21 print_bb()

```
func print_bb(b1:  bool, b2:  bool) → void
```

Print two booleans to standard output separated by a space.

Parameters

b1	The first boolean.
b2	The second boolean.

8.7.1.22 print_bi()

```
func print_bi(b:  bool, x:  int) → void
```

Print a boolean and an integer to standard output separated by a space.

Parameters

b	The boolean to print.
x	The integer to print.

8.7.1.23 print_bs()

```
func print_bs(b:  bool, s:  string) → void
```

Print a boolean and a string to standard output separated by a space.

Parameters

b	The boolean to print.
s	The string to print.

8.7.1.24 print_i()

```
func print_i(x:  int) → void
```

Print an integer to standard output.

Parameters

x	The integer to print.

8.7.1.25 print_ib()

```
func print_ib(x:  int, b:  bool) → void
```

Print an integer and a boolean to standard output separated by a space.

Parameters

x	The integer to print.
b	The boolean to print.

8.7.1.26 print_ii()

```
func print_ii(x1:  int, x2:  int) → void
```

Print two integers to standard output separated by a space.

Parameters

x1	The first integer.
x2	The second integer.

8.7.1.27 print_is()

```
func print_is(x:  int, s:  string) → void
```

Print an integer and a string to standard output separated by a space.

Parameters

x	The integer to print.
s	The string to print.

8.7.1.28 print_s()

```
func print_s(s:  string) → void
```

Print a string to standard output.

Parameters

s	The string to print.

8.7.1.29 print_sb()

```
func print_sb(s:  string, b:  bool) → void
```

Print a string and a boolean to standard output separated by a space.

Parameters

s	The string to print.
b	The boolean to print.

8.7.1.30 print_si()

```
func print_si(s:  string, x:  int) → void
```

Print a string and an integer to standard output separated by a space.

Parameters

s	The string to print.
x	The integer to print.

8.7.1.31 print_ss()

```
func print_ss(s1:  string, s2:  string) → void
```

Print two strings to standard output separated by a space.

Parameters

s1	The first string.
s2	The second string.

8.7.1.32 printl()

```
func printl() → void
```

Print a newline to standard output.

8.7.1.33 printl_b()

```
func printl_b(x: bool) → void
```

Print a boolean to standard output followed by a newline.

Parameters

x	The boolean to print.

8.7.1.34 printl_bb()

```
func printl_bb(b1: bool, b2: bool) → void
```

Print two booleans to standard output separated by a space, followed by a newline.

Parameters

b1	The first boolean.
b2	The second boolean.

8.7.1.35 printl_bi()

```
func printl_bi(b: bool, x: int) → void
```

Print a boolean and an integer to standard output separated by a space, followed by a newline.

Parameters

b	The boolean to print.
x	The integer to print.

8.7.1.36 printl_bs()

```
func printl_bs(b: bool, s: string) → void
```

Print a boolean and a string to standard output separated by a space, followed by a newline.

Parameters

b	The boolean to print.
s	The string to print.

8.7.1.37 printl_i()

```
func printl_i(x:  int) → void
```

Print an integer to standard output followed by a newline.

Parameters

x	The integer to print.

8.7.1.38 printl_ib()

```
func printl_ib(x:  int, b:  bool) → void
```

Print an integer and a boolean to standard output separated by a space, followed by a newline.

Parameters

x	The integer to print.
b	The boolean to print.

8.7.1.39 printl_ii()

```
func printl_ii(x1:  int, x2:  int) → void
```

Print two integers to standard output separated by a space, followed by a newline.

Parameters

x1	The first integer.
x2	The second integer.

8.7.1.40 printl_is()

```
func printl_is(x:  int, s:  string) → void
```

Print an integer and a string to standard output separated by a space, followed by a newline.

Parameters

x	The integer to print.
s	The string to print.

8.7.1.41 printl_s()

```
func printl_s(s:  string) → void
```

Print a string to standard output followed by a newline.

Parameters

s	The string to print.

8.7.1.42 printl_sb()

```
func printl_sb(s:  string, b:  bool) → void
```

Print a string and a boolean to standard output separated by a space, followed by a newline.

Parameters

s	The string to print.
b	The boolean to print.

8.7.1.43 printl_si()

```
func printl_si(s:  string, x:  int) → void
```

Print a string and an integer to standard output separated by a space, followed by a newline.

Parameters

s	The string to print.
x	The integer to print.

8.7.1.44 printl_ss()

```
func printl_ss(s1:  string, s2:  string) → void
```

Print two strings to standard output separated by a space, followed by a newline.

Parameters

s1	The first string.
s2	The second string.

8.7.1.45 read_char()

```
func read_char() → int?
```

Read a single character from standard input.

Returns

The character read as an integer, or null on EOF or error.

8.7.1.46 read_char_or_eof()

```
func read_char_or_eof() → int
```

Read a single character from standard input.

Returns

The character read, or -1 on EOF or error.

8.7.1.47 read_line()

```
func read_line() → string?
```

Read a line from standard input.

Returns

The line read, or null on EOF or error.

8.7.1.48 read_stdin_some()

```
func read_stdin_some(buf: ByteArray*, start: int, count: int) → int?
```

Read up to `count` raw bytes from standard input into one byte-array subrange.

Parameters

buf	Destination buffer.
start	Start offset within the buffer.
count	Maximum number of bytes to read.

Returns

Bytes read, `0` on EOF, or null on error.

8.7.1.49 write_stderr_all()

```
func write_stderr_all(buf:  ByteArray*, start:  int, count:  int) → Unit?
```

Write exactly `count` raw bytes from one byte-array subrange to standard error.

Parameters

buf	Source buffer.
start	Start offset within the buffer.
count	Exact number of bytes to write.

Returns

present() on success, or null on error.

8.7.1.50 write_stderr_some()

```
func write_stderr_some(buf:  ByteArray*, start:  int, count:  int) → int?
```

Write up to `count` raw bytes from one byte-array subrange to standard error.

Parameters

buf	Source buffer.
start	Start offset within the buffer.
count	Maximum number of bytes to write.

Returns

Bytes written, or null on error.

8.7.1.51 write_stdout_all()

```
func write_stdout_all(buf:  ByteArray*, start:  int, count:  int) → Unit?
```

Write exactly `count` raw bytes from one byte-array subrange to standard output.

Parameters

buf	Source buffer.
start	Start offset within the buffer.
count	Exact number of bytes to write.

Returns

present() on success, or null on error.

8.7.1.52 write_stdout_some()

```
func write_stdout_some(buf: ByteArray*, start: int, count: int) → int?
```

Write up to `count` raw bytes from one byte-array subrange to standard output.

Parameters

buf	Source buffer.
start	Start offset within the buffer.
count	Maximum number of bytes to write.

Returns

Bytes written, or null on error.

8.8 compiler/shared/l0/stdlib/std/linear_map.l0 File Reference

Module: `std.linear_map`

```
import std.vector;
import sys.hash;
import std.assert;
import std.string;
import sys.memory;
import sys.rt;
```

Classes

- struct LinearMapEntryBase

  Linear map implementation using linear search.
- struct LinearMapBase

  Base structure for a linear map.
- struct StringStringLinearMapEntry

  Entry for a string-to-string linear map.
- struct StringStringLinearMap

  A linear map specialized for string keys and string values.
- struct IntStringLinearMapEntry

  Entry for an integer-to-string linear map.
- struct IntStringLinearMap

  A linear map specialized for integer keys and string values.

Functions

- func _lm_grow(self: LinearMapBase∗)

  Increases the capacity of the map if needed and grows the length by one.
- func _lm_entry_by_index(self: LinearMapBase∗, index: int) → LinearMapEntryBase∗

  Returns a pointer to the entry at the specified index.
- func _lm_value_by_entry(self: LinearMapBase∗, entry_ptr: LinearMapEntryBase∗) → void∗

  Returns a pointer to the value within an entry.
- func _lm_key_by_entry(self: LinearMapBase∗, entry_ptr: LinearMapEntryBase∗) → void∗

  Returns a pointer to the key within an entry.
- func _lm_entry_by_hk_key(self: LinearMapBase∗, hkey: int, key_ptr: void∗) → LinearMapEntryBase∗?

  Finds an entry by its hash key and key pointer.
- func _lm_entry_by_hv_value(self: LinearMapBase∗, hval: int, value_ptr: void∗) → LinearMapEntryBase∗?

  Finds an entry by its hash value and value pointer.
- func _lm_new_entry(self: LinearMapBase∗) → void∗

  Allocates space for a new entry in the map.
- func _lm_set_entry(self: LinearMapBase∗, entry_ptr: LinearMapEntryBase∗, hkey: int, key_ptr: void∗, hval: int, value_ptr: void∗)

  Sets the key and value data for a given entry.
- func lm_create(key_size: int, value_size: int, initial_capacity: int) → LinearMapBase∗

  Creates a new LinearMapBase with the specified sizes and capacity.
- func lm_free(self: LinearMapBase∗)

  Free a `LinearMapBase` and its entry storage.
- func lm_len(self: LinearMapBase∗) → int

  Returns the number of entries currently in the map.
- func lm_contains_key(self: LinearMapBase∗, key_ptr: void∗) → bool

  Checks if the map contains the specified key.
- func lm_contains_value(self: LinearMapBase∗, value_ptr: void∗) → bool

  Checks if the map contains the specified value.
- func lm_remove(self: LinearMapBase∗, key_ptr: void∗) → bool

  Removes the entry with the specified key.
- func lm_set(self: LinearMapBase∗, key_ptr: void∗, value_ptr: void∗)

  Inserts or updates a key-value pair in the map.
- func lm_get(self: LinearMapBase∗, key_ptr: void∗) → void∗?

  Retrieves the value associated with the specified key.
- func _sslm_entry_by_index(self: StringStringLinearMap∗, index: int) → StringStringLinearMapEntry∗

  Returns a pointer to the string-string entry at the specified index.
- func _sslm_find_by_key(self: StringStringLinearMap∗, key: string) → int

  Finds the index of an entry by its string key.
- func sslm_create(initial_capacity: int) → StringStringLinearMap∗

  Creates a new string-to-string linear map.
- func sslm_free(self: StringStringLinearMap∗)

  Free a string-to-string linear map and release stored keys and values.
- func sslm_len(self: StringStringLinearMap∗) → int

  Returns the number of entries in the string-to-string map.
- func sslm_set(self: StringStringLinearMap∗, key: string, value: string)

  Inserts or updates a key-value pair in the string-to-string map.
- func sslm_get(self: StringStringLinearMap∗, key: string) → string?

  Retrieves the value associated with the specified key string.
- func sslm_contains(self: StringStringLinearMap∗, key: string) → bool

  Checks if the map contains the specified key string.

- func sslm_remove(self: StringStringLinearMap*, key: string) → bool

  Removes the entry with the specified key string.
- func sslm_key_at(self: StringStringLinearMap*, index: int) → string

  Returns the key string at the specified entry index.
- func sslm_value_at(self: StringStringLinearMap*, index: int) → string

  Returns the value string at the specified entry index.
- func _islm_entry_by_index(self: IntStringLinearMap*, index: int) → IntStringLinearMapEntry*

  Returns a pointer to the int-string entry at the specified index.
- func _islm_find_by_key(self: IntStringLinearMap*, key: int) → int

  Finds the index of an entry by its integer key.
- func islm_create(initial_capacity: int) → IntStringLinearMap*

  Creates a new integer-to-string linear map.
- func islm_free(self: IntStringLinearMap*)

  Free an integer-to-string linear map and release stored values.
- func islm_len(self: IntStringLinearMap*) → int

  Returns the number of entries in the integer-to-string map.
- func islm_set(self: IntStringLinearMap*, key: int, value: string)

  Inserts or updates a key-value pair in the integer-to-string map.
- func islm_get(self: IntStringLinearMap*, key: int) → string?

  Retrieves the value associated with the specified integer key.
- func islm_contains(self: IntStringLinearMap*, key: int) → bool

  Checks if the map contains the specified integer key.
- func islm_remove(self: IntStringLinearMap*, key: int) → bool

  Removes the entry with the specified integer key.
- func islm_key_at(self: IntStringLinearMap*, index: int) → int

  Returns the integer key at the specified entry index.
- func islm_value_at(self: IntStringLinearMap*, index: int) → string

  Returns the value string at the specified entry index.

8.8.1 Function Documentation

8.8.1.1 _islm_entry_by_index()

```
func _islm_entry_by_index(self:  IntStringLinearMap*, index:  int) → IntStringLinearMapEntry*
```

Returns a pointer to the int-string entry at the specified index.

Parameters

self	The pointer to the IntStringLinearMap.
index	The index of the entry.

Returns

A pointer to the IntStringLinearMapEntry.

8.8.1.2 _islm_find_by_key()

```
func _islm_find_by_key(self:  IntStringLinearMap*, key:  int) → int
```

Finds the index of an entry by its integer key.

Parameters

self	The pointer to the IntStringLinearMap.
key	The integer key to find.

Returns

The index of the entry, or -1 if not found.

8.8.1.3 _lm_entry_by_hk_key()

```
func _lm_entry_by_hk_key(self: LinearMapBase*, hkey: int, key_ptr: void*) → LinearMapEntryBase*?
```

Finds an entry by its hash key and key pointer.

Parameters

self	The pointer to the LinearMapBase.
hkey	The hash of the key.
key_ptr	The pointer to the key data to compare.

Returns

The found entry pointer, or null if not found.

8.8.1.4 _lm_entry_by_hv_value()

```
func _lm_entry_by_hv_value(self: LinearMapBase*, hval: int, value_ptr: void*) → LinearMapEntryBase*?
```

Finds an entry by its hash value and value pointer.

Parameters

self	The pointer to the LinearMapBase.
hval	The hash of the value.
value_ptr	The pointer to the value data to compare.

Returns

The found entry pointer, or null if not found.

8.8.1.5 _lm_entry_by_index()

```
func _lm_entry_by_index(self: LinearMapBase*, index: int) → LinearMapEntryBase*
```

Returns a pointer to the entry at the specified index.

Parameters

self	The pointer to the LinearMapBase.
index	The index of the entry.

Returns

A pointer to the LinearMapEntryBase.

8.8.1.6 _lm_grow()

```
func _lm_grow(self:  LinearMapBase*)
```

Increases the capacity of the map if needed and grows the length by one.

Parameters

self	The pointer to the LinearMapBase.

8.8.1.7 _lm_key_by_entry()

```
func _lm_key_by_entry(self:  LinearMapBase*, entry_ptr:  LinearMapEntryBase*) → void*
```

Returns a pointer to the key within an entry.

Parameters

self	The pointer to the LinearMapBase.
entry_ptr	The pointer to the LinearMapEntryBase.

Returns

A pointer to the key data.

8.8.1.8 _lm_new_entry()

```
func _lm_new_entry(self:  LinearMapBase*) → void*
```

Allocates space for a new entry in the map.

Parameters

self	The pointer to the LinearMapBase.

Returns

A pointer to the new entry.

8.8.1.9 _lm_set_entry()

```
func _lm_set_entry(self: LinearMapBase*, entry_ptr: LinearMapEntryBase*, hkey: int, key_ptr:
void*, hval: int, value_ptr: void*)
```

Sets the key and value data for a given entry.

Parameters

self	The pointer to the LinearMapBase.
entry_ptr	The entry to modify.
hkey	The hash of the key.
key_ptr	The pointer to the source key data.
hval	The hash of the value.
value_ptr	The pointer to the source value data.

8.8.1.10 _lm_value_by_entry()

```
func _lm_value_by_entry(self: LinearMapBase*, entry_ptr: LinearMapEntryBase*) → void*
```

Returns a pointer to the value within an entry.

Parameters

self	The pointer to the LinearMapBase.
entry_ptr	The pointer to the LinearMapEntryBase.

Returns

A pointer to the value data.

8.8.1.11 _sslm_entry_by_index()

```
func _sslm_entry_by_index(self: StringStringLinearMap*, index: int) → StringStringLinearMapEntry*
```

Returns a pointer to the string-string entry at the specified index.

Parameters

self	The pointer to the StringStringLinearMap.
index	The index of the entry.

Returns

A pointer to the StringStringLinearMapEntry.

8.8.1.12 _sslm_find_by_key()

```
func _sslm_find_by_key(self:  StringStringLinearMap*, key:  string) → int
```

Finds the index of an entry by its string key.

Parameters

self	The pointer to the StringStringLinearMap.
key	The key string to find.

Returns

The index of the entry, or -1 if not found.

8.8.1.13 islm_contains()

```
func islm_contains(self:  IntStringLinearMap*, key:  int) → bool
```

Checks if the map contains the specified integer key.

Parameters

self	The pointer to the map.
key	The integer key.

Returns

True if present, false otherwise.

8.8.1.14 islm_create()

```
func islm_create(initial_capacity:  int) → IntStringLinearMap*
```

Creates a new integer-to-string linear map.

Parameters

initial_capacity	The initial capacity of the map.

Returns

A pointer to the newly created map.

8.8.1.15 islm_free()

```
func islm_free(self:  IntStringLinearMap*)
```

Free an integer-to-string linear map and release stored values.

Parameters

self	Map to free.

8.8.1.16 islm_get()

```
func islm_get(self:  IntStringLinearMap*, key:  int) → string?
```

Retrieves the value associated with the specified integer key.

Parameters

self	The pointer to the map.
key	The integer key.

Returns

The associated value string, or null if not found.

8.8.1.17 islm_key_at()

```
func islm_key_at(self:  IntStringLinearMap*, index:  int) → int
```

Returns the integer key at the specified entry index.

Parameters

self	The pointer to the map.
index	The entry index.

Returns

The integer key.

8.8.1.18 islm_len()

```
func islm_len(self:  IntStringLinearMap*) → int
```

Returns the number of entries in the integer-to-string map.

Parameters

self	The pointer to the map.

Returns

The entry count.

8.8.1.19 islm_remove()

```
func islm_remove(self: IntStringLinearMap*, key: int) → bool
```

Removes the entry with the specified integer key.

Parameters

self	The pointer to the map.
key	The integer key to remove.

Returns

True if found and removed, false otherwise.

8.8.1.20 islm_set()

```
func islm_set(self: IntStringLinearMap*, key: int, value: string)
```

Inserts or updates a key-value pair in the integer-to-string map.

Parameters

self	The pointer to the map.
key	The integer key.
value	The value string.

8.8.1.21 islm_value_at()

```
func islm_value_at(self: IntStringLinearMap*, index: int) → string
```

Returns the value string at the specified entry index.

Parameters

self	The pointer to the map.
index	The entry index.

Returns

The value string.

8.8.1.22 lm_contains_key()

```
func lm_contains_key(self:  LinearMapBase*, key_ptr:  void*) → bool
```

Checks if the map contains the specified key.

Parameters

self	The pointer to the LinearMapBase.
key_ptr	The pointer to the key data.

Returns

True if the key is present, false otherwise.

8.8.1.23 lm_contains_value()

```
func lm_contains_value(self:  LinearMapBase*, value_ptr:  void*) → bool
```

Checks if the map contains the specified value.

Parameters

self	The pointer to the LinearMapBase.
value_ptr	The pointer to the value data.

Returns

True if the value is present, false otherwise.

8.8.1.24 lm_create()

```
func lm_create(key_size:  int, value_size:  int, initial_capacity:  int) → LinearMapBase*
```

Creates a new LinearMapBase with the specified sizes and capacity.

Parameters

key_size	Size of the key in bytes.
value_size	Size of the value in bytes.
initial_capacity	Initial number of entries.

Returns

A pointer to the newly created map.

8.8.1.25 lm_free()

```
func lm_free(self:  LinearMapBase*)
```

Free a `LinearMapBase` and its entry storage.

Parameters

self	Linear map to free.

8.8.1.26 lm_get()

```
func lm_get(self:  LinearMapBase*, key_ptr:  void*) → void*?
```

Retrieves the value associated with the specified key.

Parameters

self	The pointer to the LinearMapBase.
key_ptr	The pointer to the key data to look up.

Returns

A pointer to the value data, or null if not found.

8.8.1.27 lm_len()

```
func lm_len(self:  LinearMapBase*) → int
```

Returns the number of entries currently in the map.

Parameters

self	The pointer to the LinearMapBase.

Returns

The entry count.

8.8.1.28 lm_remove()

```
func lm_remove(self:  LinearMapBase*, key_ptr:  void*) → bool
```

Removes the entry with the specified key.

Parameters

self	The pointer to the LinearMapBase.
key_ptr	The pointer to the key data to remove.

Returns

True if the key was found and removed, false otherwise.

8.8.1.29 lm_set()

```
func lm_set(self: LinearMapBase*, key_ptr: void*, value_ptr: void*)
```

Inserts or updates a key-value pair in the map.

Parameters

self	The pointer to the LinearMapBase.
key_ptr	The pointer to the key data.
value_ptr	The pointer to the value data.

8.8.1.30 sslm_contains()

```
func sslm_contains(self: StringStringLinearMap*, key: string) → bool
```

Checks if the map contains the specified key string.

Parameters

self	The pointer to the map.
key	The key string.

Returns

True if present, false otherwise.

8.8.1.31 sslm_create()

```
func sslm_create(initial_capacity: int) → StringStringLinearMap*
```

Creates a new string-to-string linear map.

Parameters

initial_capacity	The initial capacity of the map.

Returns

A pointer to the newly created map.

8.8.1.32 sslm_free()

```
func sslm_free(self:  StringStringLinearMap*)
```

Free a string-to-string linear map and release stored keys and values.

Parameters

self	Map to free.

8.8.1.33 sslm_get()

```
func sslm_get(self:  StringStringLinearMap*, key:  string) → string?
```

Retrieves the value associated with the specified key string.

Parameters

self	The pointer to the map.
key	The key string.

Returns

The associated value string, or null if not found.

8.8.1.34 sslm_key_at()

```
func sslm_key_at(self:  StringStringLinearMap*, index:  int) → string
```

Returns the key string at the specified entry index.

Parameters

self	The pointer to the map.
index	The entry index.

Returns

The key string.

8.8.1.35 sslm_len()

```
func sslm_len(self:  StringStringLinearMap*) → int
```

Returns the number of entries in the string-to-string map.

Parameters

self	The pointer to the map.

Returns

The entry count.

8.8.1.36 sslm_remove()

```
func sslm_remove(self:  StringStringLinearMap*, key:  string) → bool
```

Removes the entry with the specified key string.

Parameters

self	The pointer to the map.
key	The key string to remove.

Returns

True if found and removed, false otherwise.

8.8.1.37 sslm_set()

```
func sslm_set(self:  StringStringLinearMap*, key:  string, value:  string)
```

Inserts or updates a key-value pair in the string-to-string map.

Parameters

self	The pointer to the map.
key	The key string.
value	The value string.

8.8.1.38 sslm_value_at()

```
func sslm_value_at(self:  StringStringLinearMap*, index:  int) → string
```

Returns the value string at the specified entry index.

Parameters

self	The pointer to the map.
index	The entry index.

Returns

The value string.

8.9 compiler/shared/l0/stdlib/std/optional.l0 File Reference

Module: `std.optional`

```
import std.assert;
```

Functions

- func unwrap_or_s(opt: string?, default: string) → string

  Unwraps an optional string, returning the contained value if present, otherwise returning the provided default string.
- func unwrap_or_i(opt: int?, default: int) → int

  Unwraps an optional int, returning the contained value if present, otherwise returning the provided default int.
- func unwrap_or_b(opt: bool?, default: bool) → bool

  Unwraps an optional bool, returning the contained value if present, otherwise returning the provided default bool.
- func expect_s(opt: string?, msg: string) → string

  Expects an optional string to be non-null, returning the contained value.
- func expect_i(opt: int?, msg: string) → int

  Expects an optional int to be non-null, returning the contained value.
- func expect_b(opt: bool?, msg: string) → bool

  Expects an optional bool to be non-null, returning the contained value.

8.9.1 Function Documentation

8.9.1.1 expect_b()

```
func expect_b(opt:  bool?, msg:  string) → bool
```

Expects an optional bool to be non-null, returning the contained value.

If the optional is null, it aborts with the provided message.

Parameters

opt	The optional boolean.
msg	The abort message if null.

Returns

The unwrapped boolean value.

8.9.1.2 expect_i()

```
func expect_i(opt:  int?, msg:  string) → int
```

Expects an optional int to be non-null, returning the contained value.

If the optional is null, it aborts with the provided message.

Parameters

opt	The optional integer.
msg	The abort message if null.

Returns

The unwrapped integer value.

8.9.1.3 expect_s()

```
func expect_s(opt:  string?, msg:  string) → string
```

Expects an optional string to be non-null, returning the contained value.

If the optional is null, it aborts with the provided message.

Parameters

opt	The optional string.
msg	The abort message if null.

Returns

The unwrapped string value.

8.9.1.4 unwrap_or_b()

```
func unwrap_or_b(opt:  bool?, default:  bool) → bool
```

Unwraps an optional bool, returning the contained value if present, otherwise returning the provided default bool.

Parameters

opt	The optional boolean to unwrap.
default	The default boolean value.

Returns

The unwrapped value or the default.

8.9.1.5 unwrap_or_i()

```
func unwrap_or_i(opt:  int?, default:  int) → int
```

Unwraps an optional int, returning the contained value if present, otherwise returning the provided default int.

Parameters

opt	The optional integer to unwrap.
default	The default integer value.

Returns

The unwrapped value or the default.

8.9.1.6 unwrap_or_s()

```
func unwrap_or_s(opt:  string?, default:  string) → string
```

Unwraps an optional string, returning the contained value if present, otherwise returning the provided default string.

Parameters

opt	The optional string to unwrap.
default	The default string value.

Returns

The unwrapped value or the default.

8.10 compiler/shared/l0/stdlib/std/rand.l0 File Reference

Module: `std.rand`

```
import sys.rt;
```

Functions

- func rand_seed(seed : int) → void

  Seeds the random number generator with the given seed.
- func rand_int(max : int) → int

  Returns a random integer in the range [0, max).
- func rand_int_range(min : int, max : int) → int

  Returns a random integer in the range [min, max).
- func rand_bool() → bool

  Returns a random boolean value.
- func rand_dice(sides : int, rolls : int) → int

  Rolls the specified number of dice with the given number of sides and returns the total.

8.10.1 Function Documentation

8.10.1.1 rand_bool()

```
func rand_bool() → bool
```

Returns a random boolean value.

Returns

A random boolean.

8.10.1.2 rand_dice()

```
func rand_dice(sides :  int, rolls :  int) → int
```

Rolls the specified number of dice with the given number of sides and returns the total.

Parameters

sides	The number of sides on each die.
rolls	The number of dice to roll.

Returns

The total sum of the rolls.

8.10.1.3 rand_int()

```
func rand_int(max :  int) → int
```

Returns a random integer in the range [0, max).

Parameters

max	The upper bound (exclusive).

Returns

A random integer.

8.10.1.4 rand_int_range()

```
func rand_int_range(min :  int, max :  int) → int
```

Returns a random integer in the range [min, max).

Parameters

min	The lower bound (inclusive).
max	The upper bound (exclusive).

Returns

A random integer.

8.10.1.5 rand_seed()

```
func rand_seed(seed :  int) → void
```

Seeds the random number generator with the given seed.

A non-zero seed produces a reproducible sequence of numbers for the same seed. 0 means use the current time as the seed (not reproducible sequence of numbers).

Parameters

seed	The integer seed value.

8.11 compiler/shared/l0/stdlib/std/string.l0 File Reference

Module: `std.string`

```
import std.assert;
import sys.rt;
```

Functions

- func len_s(s: string) → int

  Returns the length of the given string.
- func is_empty_s(s: string) → bool

  Checks if the given string is empty.
- func char_at_s(s: string, index: int) → byte

  Returns the character code (0-255) at the specified index in the string.
- func eq_s(a: string, b: string) → bool

  Compares two strings for equality.
- func cmp_s(a: string, b: string) → int

  Compares two strings lexicographically.
- func concat_s(a: string, b: string) → string

  Concatenates two strings and returns the result.
- func slice_s(s: string, start: int, end: int) → string

  Returns a slice of the string from start to end (exclusive).

- func byte_to_s(b: byte) → string

  Converts a single byte to a string.
- func bytes_to_s(bytes: byte*, len: int) → string

  Creates a string from a byte array of a given length.
- func find_s(haystack: string, needle: string) → int

  Returns the index of the first occurrence of 'needle' in 'haystack', or -1 if not found.
- func find_last_s(haystack: string, needle: string) → int

  Returns the index of the last occurrence of 'needle' in 'haystack', or -1 if not found.
- func find_from_s(haystack: string, needle: string, pos: int) → int

  Returns the index of the first occurrence of 'needle' in 'haystack' starting from 'pos', or -1 if not found.
- func contains_s(haystack: string, needle: string) → bool

  Returns true if 'haystack' contains 'needle'.
- func starts_with_s(s: string, prefix: string) → bool

  Returns true if 's' starts with 'prefix'.
- func ends_with_s(s: string, suffix: string) → bool

  Returns true if 's' ends with 'suffix'.
- func is_space(c: byte) → bool

  Checks if the given character is a whitespace character.
- func is_digit(c: byte) → bool

  Checks if the given character is a decimal digit.
- func is_digit_base(c: byte, base: int) → bool

  Checks if the character is a valid digit in the specified base.
- func is_alpha(c: byte) → bool

  Checks if the character is an alphabetic character.
- func is_alnum(c: byte) → bool

  Checks if the character is alphanumeric.
- func to_digit(c: byte) → int

  Converts a numeric character to its integer value.
- func to_digit_base(c: byte, base: int) → int?

  Converts a character to its integer value in the specified base.
- func to_upper(c: byte) → byte

  Converts a lowercase character to uppercase.
- func to_lower(c: byte) → byte

  Converts an uppercase character to lowercase.
- func trim_s(s: string) → string

  Removes leading and trailing whitespace from the input string.
- func digit_to_char(d: int) → byte

  Converts a digit (0-15) to its character representation.

8.11.1 Function Documentation

8.11.1.1 byte_to_s()

```
func byte_to_s(b:  byte) → string
```

Converts a single byte to a string.

Parameters

b	The byte to convert.

Returns

A string containing the byte.

8.11.1.2 bytes_to_s()

```
func bytes_to_s(bytes:  byte*, len:  int) → string
```

Creates a string from a byte array of a given length.

Parameters

bytes	The pointer to the byte array.
len	The number of bytes to include.

Returns

New string containing the provided bytes.

8.11.1.3 char_at_s()

```
func char_at_s(s:  string, index:  int) → byte
```

Returns the character code (0-255) at the specified index in the string.

Parameters

s	The string.
index	The index to access.

Returns

The character byte.

8.11.1.4 cmp_s()

```
func cmp_s(a:  string, b:  string) → int
```

Compares two strings lexicographically.

Parameters

a	The first string.
b	The second string.

Returns

A negative value if a < b, zero if a == b, positive if a > b.

8.11.1.5 concat_s()

```
func concat_s(a:  string, b:  string) → string
```

Concatenates two strings and returns the result.

Parameters

a	The first string.
b	The second string.

Returns

The combined string.

8.11.1.6 contains_s()

```
func contains_s(haystack:  string, needle:  string) → bool
```

Returns true if 'haystack' contains 'needle'.

Parameters

haystack	The string to search in.
needle	The string to search for.

Returns

True if found, false otherwise.

8.11.1.7 digit_to_char()

```
func digit_to_char(d:  int) → byte
```

Converts a digit (0-15) to its character representation.

Parameters

d	The digit value.

Returns

The character representation ('0'-'9', 'a'-'f').

8.11.1.8 ends_with_s()

```
func ends_with_s(s:  string, suffix:  string) → bool
```

Returns true if 's' ends with 'suffix'.

Parameters

s	The string.
suffix	The suffix string.

Returns

True if 's' ends with 'suffix', false otherwise.

8.11.1.9 eq_s()

```
func eq_s(a:  string, b:  string) → bool
```

Compares two strings for equality.

Parameters

a	The first string.
b	The second string.

Returns

True if equal, false otherwise.

8.11.1.10 find_from_s()

```
func find_from_s(haystack:  string, needle:  string, pos:  int) → int
```

Returns the index of the first occurrence of 'needle' in 'haystack' starting from 'pos', or -1 if not found.

Parameters

haystack	The string to search in.
needle	The string to search for.
pos	The starting position.

Returns

The index of the first occurrence, or -1.

8.11.1.11 find_last_s()

```
func find_last_s(haystack:  string, needle:  string) → int
```

Returns the index of the last occurrence of 'needle' in 'haystack', or -1 if not found.

For empty needle, returns len_s(haystack).

Parameters

haystack	The string to search in.
needle	The string to search for.

Returns

The index of the last occurrence, or -1.

8.11.1.12 find_s()

```
func find_s(haystack:  string, needle:  string) → int
```

Returns the index of the first occurrence of 'needle' in 'haystack', or -1 if not found.

Parameters

haystack	The string to search in.
needle	The string to search for.

Returns

The index of the first occurrence, or -1.

8.11.1.13 is_alnum()

```
func is_alnum(c:  byte) → bool
```

Checks if the character is alphanumeric.

Parameters

c	The character.

Returns

True if alphanumeric, false otherwise.

8.11.1.14 is_alpha()

```
func is_alpha(c:  byte) → bool
```

Checks if the character is an alphabetic character.

Parameters

c	The character.

Returns

True if alphabetic, false otherwise.

8.11.1.15 is_digit()

```
func is_digit(c:  byte) → bool
```

Checks if the given character is a decimal digit.

Parameters

c	The character.

Returns

True if a digit, false otherwise.

8.11.1.16 is_digit_base()

```
func is_digit_base(c:  byte, base:  int) → bool
```

Checks if the character is a valid digit in the specified base.

Parameters

c	The character.
base	The numeric base (2-16).

Returns

True if a valid digit, false otherwise.

8.11.1.17 is_empty_s()

```
func is_empty_s(s:  string) → bool
```

Checks if the given string is empty.

Parameters

s	The string to check.

Returns

True if empty, false otherwise.

8.11.1.18 is_space()

```
func is_space(c:  byte) → bool
```

Checks if the given character is a whitespace character.

Parameters

c	The character.

Returns

True if whitespace, false otherwise.

8.11.1.19 len_s()

```
func len_s(s:  string) → int
```

Returns the length of the given string.

Parameters

s	The string.

Returns

The length of the string.

8.11.1.20 slice_s()

```
func slice_s(s:  string, start:  int, end:  int) → string
```

Returns a slice of the string from start to end (exclusive).

Parameters

s	The string.
start	The starting index.
end	The ending index.

Returns

The string slice.

8.11.1.21 starts_with_s()

```
func starts_with_s(s:  string, prefix:  string) → bool
```

Returns true if 's' starts with 'prefix'.

Parameters

s	The string.
prefix	The prefix string.

Returns

True if 's' starts with 'prefix', false otherwise.

8.11.1.22 to_digit()

```
func to_digit(c:  byte) → int
```

Converts a numeric character to its integer value.

Parameters

c	The character.

Returns

The integer value.

8.11.1.23 to_digit_base()

```
func to_digit_base(c:  byte, base:  int) → int?
```

Converts a character to its integer value in the specified base.

Parameters

c	The character.
base	The numeric base (2-16).

Returns

The integer value, or null if invalid.

8.11.1.24 to_lower()

```
func to_lower(c:  byte) → byte
```

Converts an uppercase character to lowercase.

Parameters

c	The character.

Returns

The lowercase character.

8.11.1.25 to_upper()

```
func to_upper(c:  byte) → byte
```

Converts a lowercase character to uppercase.

Parameters

c	The character.

Returns

The uppercase character.

8.11.1.26 trim_s()

```
func trim_s(s:  string) → string
```

Removes leading and trailing whitespace from the input string.

Parameters

s	The string to trim.

Returns

The trimmed string.

8.12 compiler/shared/l0/stdlib/std/system.l0 File Reference

Module: `std.system`

```
import sys.rt;
```

Functions

- func exit(code: int) → void

  Exits the program with the given exit code.
- func env_get(var_name: string) → string?

  Retrieves the value of the specified environment variable.
- func argc() → int

  Retrieves the number of command-line arguments.
- func get_pid() → int

  Retrieves the current process identifier.
- func argv(index: int) → string

  Retrieves the command-line argument at the specified index.
- func abort(message: string) → void

  Aborts the program with the specified message.
- func errno() → int

  Retrieves the last runtime error number.
- func system(cmd: string) → int

  Execute a system command and return its exit code.

8.12.1 Function Documentation

8.12.1.1 abort()

```
func abort(message:  string) → void
```

Aborts the program with the specified message.

Parameters

message	The abort message.

8.12.1.2 argc()

```
func argc() → int
```

Retrieves the number of command-line arguments.

Returns

The number of arguments.

8.12.1.3 argv()

```
func argv(index:  int) → string
```

Retrieves the command-line argument at the specified index.

Parameters

index	The argument index.

Returns

The command-line argument string.

8.12.1.4 env_get()

```
func env_get(var_name:  string) → string?
```

Retrieves the value of the specified environment variable.

Parameters

var_name	The name of the environment variable.

Returns

The value of the environment variable, or null if not set.

8.12.1.5 errno()

```
func errno() → int
```

Retrieves the last runtime error number.

Returns

The error number (errno).

8.12.1.6 exit()

```
func exit(code:  int) → void
```

Exits the program with the given exit code.

Parameters

code	The integer exit code.

8.12.1.7 get_pid()

```
func get_pid() → int
```

Retrieves the current process identifier.

Returns

The current process identifier.

8.12.1.8 system()

```
func system(cmd: string) → int
```

Execute a system command and return its exit code.

Parameters

cmd	The command string to execute.

Returns

The exit status code.

8.13 compiler/shared/l0/stdlib/std/text.l0 File Reference

Module: `std.text`

```
import std.vector;
import std.assert;
import std.integer;
import std.string;
import sys.memory;
import sys.rt;
```

Classes

- struct StringBuffer

  StringBuffer allows for efficient construction of strings by accumulating parts in a vector.
- struct CharBuffer

  CharBuffer allows for efficient building of strings by accumulating individual characters.

Functions

- func sb_create() → StringBuffer*

  Creates a new, empty StringBuffer.
- func sb_append(self: StringBuffer*, s: string)

  Appends a string to the StringBuffer.
- func sb_append_int(self: StringBuffer*, value: int)

  Appends an integer to the StringBuffer.
- func sb_append_byte(self: StringBuffer*, b: byte)

  Appends a byte to the StringBuffer.
- func sb_to_string(self: StringBuffer*) → string

  Converts the accumulated contents of the StringBuffer into a single string.
- func sb_size(self: StringBuffer*) → int

  Returns the current total size of all strings in the StringBuffer.
- func sb_free(self: StringBuffer*)

  Free a `StringBuffer` and its accumulated string parts.
- func cb_create() → CharBuffer*

  Creates a new, empty CharBuffer.
- func cb_capacity(self: CharBuffer*) → int

  Returns the current capacity of the CharBuffer.
- func cb_size(self: CharBuffer*) → int

  Returns the number of characters currently in the CharBuffer.
- func cb_reserve(self: CharBuffer*, total_capacity: int)

  Ensures the CharBuffer has at least the specified capacity.
- func cb_append(self: CharBuffer*, c: byte)

  Appends a single character to the CharBuffer.
- func cb_append_s(self: CharBuffer*, s: string)

  Appends a string to the CharBuffer.
- func cb_append_slice(self: CharBuffer*, s: string, start: int, length: int)

  Appends a slice of a string to the CharBuffer.
- func cb_append_int(self: CharBuffer*, value: int)

  Appends an integer to the CharBuffer.
- func cb_reverse(cb: CharBuffer*)

  Reverses the contents of the CharBuffer in place.
- func cb_to_string(self: CharBuffer*) → string

  Converts the accumulated characters in the CharBuffer into a string.
- func cb_clear(self: CharBuffer*)

  Clears all characters from the CharBuffer without freeing memory.
- func cb_free(self: CharBuffer*)

  Free a `CharBuffer` and its backing character storage.
- func to_upper_s(s: string) → string

  Converts all characters in the input string to uppercase.
- func to_lower_s(s: string) → string

  Converts all characters in the input string to lowercase.
- func repeat_s(s: string, count: int) → string

  Returns a new string that is the input string `s` repeated `count` times.
- func split_s(s: string, sep: string) → StringVector*

  Splits a string into parts based on a separator string.
- func _line_strip_trailing_cr(s: string) → string

  Internal helper to strip a trailing CR from a line string.
- func lines_s(s: string) → StringVector*

Splits a string into lines, supporting both LF and CRLF.

- func join_s(parts: StringVector∗, sep: string) → string

  Joins parts of a string vector using a separator.
- func replace_s(s: string, old: string, replacement: string) → string

  Replaces all occurrences of an old substring with a new replacement string.
- func int_to_string_base(value: int, base: int) → string

  Converts an integer to a string representation in the specified base.
- func reverse_s(s: string) → string

  Reverses the characters in a string.
- func int_to_string(value: int) → string

  Converts an integer to its decimal string representation.
- func int_to_hex_string(value: int) → string

  Converts an integer to its hexadecimal string representation.
- func int_to_bin_string(value: int) → string

  Converts an integer to its binary string representation.
- func bool_to_string(value: bool) → string

  Converts a boolean to its string representation ("true" or "false").
- func string_to_bool(s: string) → bool?

  Parses a string representation of a boolean.
- func byte_to_string(value: byte) → string

  Converts a byte to its decimal string representation.
- func byte_to_string_base(value: byte, base: int) → string

  Converts a byte to its string representation in the specified base.
- func string_to_int(s: string) → int?

  Converts a decimal numeric string to an integer.
- func string_to_byte(s: string) → byte?

  Converts a decimal numeric string to a byte.
- func string_to_byte_base(s: string, base: int) → byte?

  Converts a numeric string in the specified base to a byte.
- func string_to_int_base(s: string, base: int) → int?

  Converts a numeric string 's' expressed in base 'base' to an integer.

8.13.1 Function Documentation

8.13.1.1 _line_strip_trailing_cr()

```
func _line_strip_trailing_cr(s:  string) → string
```

Internal helper to strip a trailing CR from a line string.

Parameters

s	The string.

Returns

The stripped string.

8.13.1.2 bool_to_string()

```
func bool_to_string(value:  bool) → string
```

Converts a boolean to its string representation ("true" or "false").

Parameters

value	The boolean value.

Returns

The string representation.

8.13.1.3 byte_to_string()

```
func byte_to_string(value:  byte) → string
```

Converts a byte to its decimal string representation.

Parameters

value	The byte value.

Returns

The decimal string.

8.13.1.4 byte_to_string_base()

```
func byte_to_string_base(value:  byte, base:  int) → string
```

Converts a byte to its string representation in the specified base.

Parameters

value	The byte value.
base	The numeric base (2-16).

Returns

The string representation.

8.13.1.5 cb_append()

```
func cb_append(self:  CharBuffer*, c:  byte)
```

Appends a single character to the CharBuffer.

Parameters

self	The pointer to the CharBuffer.
c	The character to append.

8.13.1.6 cb_append_int()

```
func cb_append_int(self: CharBuffer*, value: int)
```

Appends an integer to the CharBuffer.

Parameters

self	The pointer to the CharBuffer.
value	The integer value to append.

8.13.1.7 cb_append_s()

```
func cb_append_s(self: CharBuffer*, s: string)
```

Appends a string to the CharBuffer.

Parameters

self	The pointer to the CharBuffer.
s	The string to append.

8.13.1.8 cb_append_slice()

```
func cb_append_slice(self: CharBuffer*, s: string, start: int, length: int)
```

Appends a slice of a string to the CharBuffer.

Parameters

self	The pointer to the CharBuffer.
s	The string containing the slice.
start	The starting index of the slice.
length	The number of characters to append.

8.13.1.9 cb_capacity()

```
func cb_capacity(self: CharBuffer*) → int
```

Returns the current capacity of the CharBuffer.

Parameters

self	The pointer to the CharBuffer.

Returns

The total capacity.

8.13.1.10 cb_clear()

```
func cb_clear(self:  CharBuffer*)
```

Clears all characters from the CharBuffer without freeing memory.

Parameters

self	The pointer to the CharBuffer.

8.13.1.11 cb_create()

```
func cb_create() → CharBuffer*
```

Creates a new, empty CharBuffer.

Returns

A pointer to the newly created CharBuffer.

8.13.1.12 cb_free()

```
func cb_free(self:  CharBuffer*)
```

Free a `CharBuffer` and its backing character storage.

Parameters

self	Character buffer to free.

8.13.1.13 cb_reserve()

```
func cb_reserve(self:  CharBuffer*, total_capacity:  int)
```

Ensures the CharBuffer has at least the specified capacity.

Parameters

self	The pointer to the CharBuffer.
total_capacity	The desired capacity.

8.13.1.14 cb_reverse()

```
func cb_reverse(cb:  CharBuffer*)
```

Reverses the contents of the CharBuffer in place.

Parameters

cb	The pointer to the CharBuffer.

8.13.1.15 cb_size()

```
func cb_size(self:  CharBuffer*) → int
```

Returns the number of characters currently in the CharBuffer.

Parameters

self	The pointer to the CharBuffer.

Returns

The character count.

8.13.1.16 cb_to_string()

```
func cb_to_string(self:  CharBuffer*) → string
```

Converts the accumulated characters in the CharBuffer into a string.

Parameters

self	The pointer to the CharBuffer.

Returns

The final string.

8.13.1.17 int_to_bin_string()

```
func int_to_bin_string(value:  int) → string
```

Converts an integer to its binary string representation.

Parameters

value	The integer value.

Returns

The binary string.

8.13.1.18 int_to_hex_string()

```
func int_to_hex_string(value:  int) → string
```

Converts an integer to its hexadecimal string representation.

Parameters

value	The integer value.

Returns

The hex string.

8.13.1.19 int_to_string()

```
func int_to_string(value:  int) → string
```

Converts an integer to its decimal string representation.

Parameters

value	The integer value.

Returns

The decimal string.

8.13.1.20 int_to_string_base()

```
func int_to_string_base(value:  int, base:  int) → string
```

Converts an integer to a string representation in the specified base.

Parameters

value	The integer value.
base	The numeric base (2-16).

Returns

The string representation.

8.13.1.21 join_s()

```
func join_s(parts:  StringVector*, sep:  string) → string
```

Joins parts of a string vector using a separator.

Parameters

parts	The vector of string parts.
sep	The separator string.

Returns

The single joined string.

8.13.1.22 lines_s()

```
func lines_s(s:  string) → StringVector*
```

Splits a string into lines, supporting both LF and CRLF.

Parameters

s	The string to split into lines.

Returns

A new StringVector containing the lines.

8.13.1.23 repeat_s()

```
func repeat_s(s:  string, count:  int) → string
```

Returns a new string that is the input string `s` repeated `count` times.

Parameters

s	The string to repeat.
count	The number of repetitions.

Returns

The repeated string.

8.13.1.24 replace_s()

```
func replace_s(s: string, old: string, replacement: string) → string
```

Replaces all occurrences of an old substring with a new replacement string.

Parameters

s	The source string.
old	The substring to replace.
replacement	The replacement substring.

Returns

The modified string.

8.13.1.25 reverse_s()

```
func reverse_s(s: string) → string
```

Reverses the characters in a string.

Parameters

s	The string to reverse.

Returns

The reversed string.

8.13.1.26 sb_append()

```
func sb_append(self: StringBuffer*, s: string)
```

Appends a string to the StringBuffer.

Parameters

self	The pointer to the StringBuffer.
s	The string to append.

8.13.1.27 sb_append_byte()

```
func sb_append_byte(self:  StringBuffer*, b:  byte)
```

Appends a byte to the StringBuffer.

Parameters

self	The pointer to the StringBuffer.
b	The byte value to append.

8.13.1.28 sb_append_int()

```
func sb_append_int(self:  StringBuffer*, value:  int)
```

Appends an integer to the StringBuffer.

Parameters

self	The pointer to the StringBuffer.
value	The integer value to append.

8.13.1.29 sb_create()

```
func sb_create() → StringBuffer*
```

Creates a new, empty StringBuffer.

Returns

A pointer to the newly created StringBuffer.

8.13.1.30 sb_free()

```
func sb_free(self:  StringBuffer*)
```

Free a `StringBuffer` and its accumulated string parts.

Parameters

self	String buffer to free.

8.13.1.31 sb_size()

```
func sb_size(self:  StringBuffer*) → int
```

Returns the current total size of all strings in the StringBuffer.

Parameters

self	The pointer to the StringBuffer.

Returns

The total size in bytes.

8.13.1.32 sb_to_string()

```
func sb_to_string(self:  StringBuffer*) → string
```

Converts the accumulated contents of the StringBuffer into a single string.

Parameters

self	The pointer to the StringBuffer.

Returns

The final concatenated string.

8.13.1.33 split_s()

```
func split_s(s:  string, sep:  string) → StringVector*
```

Splits a string into parts based on a separator string.

Parameters

s	The string to split.
sep	The separator string.

Returns

A new StringVector containing the parts.

8.13.1.34 string_to_bool()

```
func string_to_bool(s:  string) → bool?
```

Parses a string representation of a boolean.

Parameters

s	The string to parse.

Returns

The boolean value, or null if invalid.

8.13.1.35 string_to_byte()

```
func string_to_byte(s:  string) → byte?
```

Converts a decimal numeric string to a byte.

Parameters

s	The string to parse.

Returns

The byte value, or null if invalid or out of range (0-255).

8.13.1.36 string_to_byte_base()

```
func string_to_byte_base(s:  string, base:  int) → byte?
```

Converts a numeric string in the specified base to a byte.

Parameters

s	The string to parse.
base	The numeric base (2-16).

Returns

The byte value, or null if invalid or out of range.

8.13.1.37 string_to_int()

```
func string_to_int(s:  string) → int?
```

Converts a decimal numeric string to an integer.

Returns null when parsing fails or overflows 32-bit signed range.

Parameters

s	The string to parse.

Returns

The integer value, or null.

8.13.1.38 string_to_int_base()

```
func string_to_int_base(s:  string, base:  int) → int?
```

Converts a numeric string 's' expressed in base 'base' to an integer.

Handles optional leading '-' for negative numbers and ignores leading zeros. No special prefixes like "0x" for hex or "0b" for binary are expected or allowed. Checks for overflow and invalid characters. Returns null if the string is not a valid 32-bit signed integer.

Parameters

s	The string to parse.
base	The numeric base (2-16).

Returns

The integer value, or null.

8.13.1.39 to_lower_s()

```
func to_lower_s(s:  string) → string
```

Converts all characters in the input string to lowercase.

Parameters

s	The string to convert.

Returns

The lowercase string.

8.13.1.40 to_upper_s()

```
func to_upper_s(s:  string) → string
```

Converts all characters in the input string to uppercase.

Parameters

s	The string to convert.

Returns

The uppercase string.

8.14 compiler/shared/l0/stdlib/std/time.l0 File Reference

Module: `std.time`

```
import std.integer;
import sys.rt;
```

Classes

- struct WallTime

  Represents wall clock time in seconds and nanoseconds since the Unix epoch.
- struct MonotonicTime

  Represents monotonic time in seconds and nanoseconds.
- struct Duration

  Represents a duration of time in seconds and nanoseconds.
- struct DateTime

  Represents a broken-down date and time.
- struct CivilDate

  Represents a civil date without time components.

Functions

- func is_valid_nsec(nsec: int) → bool

  Checks if the nanosecond component is within valid range [0, 1e9).
- func is_leap_year(year: int) → bool

  Determines if a given year is a leap year.
- func days_in_month(year: int, month: int) → int

  Returns the number of days in a given month of a given year.

- func day_of_year(year: int, month: int, day: int) → int

  Calculates the day of the year for a given date.
- func civil_from_unix_days(days: int) → CivilDate

  Calculates civil date components from Unix days.
- func datetime_from_unix_sec(unix_sec: int, nsec: int, utc_offset_sec: int, is_dst: bool) → DateTime?

  Creates a DateTime object from Unix seconds and other components.
- func wall_now() → WallTime?

  Returns the current wall clock time.
- func monotonic_supported() → bool

  Checks if the monotonic clock is supported by the runtime.
- func monotonic_now() → MonotonicTime?

  Returns the current monotonic time.
- func monotonic_diff(start: MonotonicTime, end: MonotonicTime) → Duration?

  Calculates the duration between two monotonic time points.
- func wall_to_utc_datetime(t: WallTime) → DateTime?

  Converts wall clock time to UTC DateTime.
- func wall_to_local_datetime(t: WallTime) → DateTime?

  Converts wall clock time to local DateTime.
- func utc_now_datetime() → DateTime?

  Returns the current date and time in UTC.
- func local_now_datetime() → DateTime?

  Returns the current date and time in the local time zone.

8.14.1 Function Documentation

8.14.1.1 civil_from_unix_days()

```
func civil_from_unix_days(days: int) → CivilDate
```

Calculates civil date components from Unix days.

Parameters

days	The number of days since the Unix epoch.

Returns

The broken-down CivilDate.

8.14.1.2 datetime_from_unix_sec()

```
func datetime_from_unix_sec(unix_sec: int, nsec: int, utc_offset_sec: int, is_dst: bool) →
DateTime?
```

Creates a DateTime object from Unix seconds and other components.

Parameters

unix_sec	Seconds since Unix epoch.
nsec	Nanoseconds component.
utc_offset_sec	Offset from UTC in seconds.
is_dst	True if daylight saving time is in effect.

Returns

The broken-down DateTime, or null if invalid.

8.14.1.3 day_of_year()

```
func day_of_year(year:  int, month:  int, day:  int) → int
```

Calculates the day of the year for a given date.

Parameters

year	The year.
month	The month.
day	The day of the month.

Returns

The day of the year (1-366).

8.14.1.4 days_in_month()

```
func days_in_month(year:  int, month:  int) → int
```

Returns the number of days in a given month of a given year.

Parameters

year	The year.
month	The month (1-12).

Returns

The number of days in the month.

8.14.1.5 is_leap_year()

```
func is_leap_year(year:  int) → bool
```

Determines if a given year is a leap year.

Parameters

year	The year to check.

Returns

True if leap year, false otherwise.

8.14.1.6 is_valid_nsec()

```
func is_valid_nsec(nsec:  int) → bool
```

Checks if the nanosecond component is within valid range [0, 1e9).

Parameters

nsec	The nanosecond value.

Returns

True if valid, false otherwise.

8.14.1.7 local_now_datetime()

```
func local_now_datetime() → DateTime?
```

Returns the current date and time in the local time zone.

Returns

The local DateTime, or null on error.

8.14.1.8 monotonic_diff()

```
func monotonic_diff(start:  MonotonicTime, end:  MonotonicTime) → Duration?
```

Calculates the duration between two monotonic time points.

Parameters

start	The starting time.
end	The ending time.

Returns

The Duration difference, or null if end < start or overflow.

8.14.1.9 monotonic_now()

```
func monotonic_now() → MonotonicTime?
```

Returns the current monotonic time.

Returns

The current MonotonicTime, or null if not supported or error.

8.14.1.10 monotonic_supported()

```
func monotonic_supported() → bool
```

Checks if the monotonic clock is supported by the runtime.

Returns

True if supported, false otherwise.

8.14.1.11 utc_now_datetime()

```
func utc_now_datetime() → DateTime?
```

Returns the current date and time in UTC.

Returns

The UTC DateTime, or null on error.

8.14.1.12 wall_now()

```
func wall_now() → WallTime?
```

Returns the current wall clock time.

Returns

The current WallTime, or null on error.

8.14.1.13 wall_to_local_datetime()

```
func wall_to_local_datetime(t:  WallTime) → DateTime?
```

Converts wall clock time to local DateTime.

Parameters

t	The wall clock time.

Returns

The local DateTime, or null on error.

8.14.1.14 wall_to_utc_datetime()

```
func wall_to_utc_datetime(t:  WallTime) → DateTime?
```

Converts wall clock time to UTC DateTime.

Parameters

t	The wall clock time.

Returns

The UTC DateTime, or null on error.

8.15 compiler/shared/l0/stdlib/std/unit.l0 File Reference

Module: `std.unit`

Classes

- struct Unit

  Definition of the Unit type, representing a value with no information.

Functions

- func unit() → Unit

  Returns a Unit value.
- func present() → Unit?

  Returns a non-null optional Unit value.

8.15.1 Function Documentation

8.15.1.1 present()

```
func present() → Unit?
```

Returns a non-null optional Unit value.

Returns

A present Unit optional.

8.15.1.2 unit()

```
func unit() → Unit
```

Returns a Unit value.

Returns

A Unit instance.

8.16 compiler/shared/l0/stdlib/std/vector.l0 File Reference

Module: `std.vector`

```
import std.array;
import std.string;
import std.assert;
import sys.memory;
import sys.rt;
```

Classes

- struct VectorBase

  VectorBase implements a dynamic array (vector) of elements of a given size.

Functions

- func vec_create(element_size: int, initial_capacity: int) → VectorBase∗

  Creates a new VectorBase with the specified element size and initial capacity.
- func vec_grow(self: VectorBase∗)

  Internal function to grow the vector's capacity if needed.
- func vec_reserve(self: VectorBase∗, total_capacity: int)

  Ensures the vector has at least the specified total capacity.
- func vec_check(self: VectorBase∗, index: int)

  Checks if the given index is within the logical bounds of the vector.
- func vec_get(self: VectorBase∗, index: int) → void∗

  Returns a pointer to the element at the specified index.
- func vec_push(self: VectorBase∗) → void∗

  Grows the vector by one element and returns a pointer to the new slot.
- func vec_size(self: VectorBase∗) → int

  Returns the number of elements currently in the vector.
- func vec_capacity(self: VectorBase∗) → int

  Returns the current total capacity of the vector.
- func vec_zap(self: VectorBase∗, index: int)

  Zeros out the element at the specified index.
- func vec_clear(self: VectorBase∗)

  Clears the vector, resetting its length to zero and shrinking its capacity.
- func vec_free(self: VectorBase∗)

Free a `VectorBase` and its backing array.

- func vec_push_int(self: VectorBase∗, val: int)

  Pushes an integer onto the vector.
- func vec_push_byte(self: VectorBase∗, val: byte)

  Pushes a byte onto the vector.
- func vec_push_bytes(self: VectorBase∗, src: byte∗, count: int)

  Bulk-push `count` bytes from `src` into a byte-element vector.
- func vec_push_bool(self: VectorBase∗, val: bool)

  Pushes a boolean onto the vector.
- func vec_push_ptr(self: VectorBase∗, val: void∗)

  Pushes a generic pointer onto the vector.
- func vi_sort(self: VectorBase∗)

  Sorts the elements of an integer vector in ascending order using insertion sort.
- type StringVector = VectorBase

  Creates a new StringVector with the specified initial capacity.
- func sv_push(self: StringVector∗, val: string)

  Pushes a string onto the StringVector.
- func sv_get(self: StringVector∗, index: int) → string

  Returns the string at the specified index.
- func sv_size(self: StringVector∗) → int

  Returns the number of strings currently in the vector.
- func sv_capacity(self: StringVector∗) → int

  Returns the current capacity of the string vector.
- func _sv_less(a: string, b: string) → bool

  Internal helper to compare two strings for sorting.
- func sv_sort(self: StringVector∗)

  Sorts the strings in the vector lexicographically using insertion sort.
- func sv_clear(self: StringVector∗)

  Clears the string vector and releases all held strings.
- func sv_free(self: StringVector∗)

  Free a `StringVector` and release every stored string.

Variables

- type StringVector = VectorBase

8.16.1 Function Documentation

8.16.1.1 _sv_less()

```
func _sv_less(a:  string, b:  string) → bool
```

Internal helper to compare two strings for sorting.

Parameters

a	The first string.
b	The second string.

Returns

True if a $<$ b lexicographically.

8.16.1.2 sv_capacity()

```
func sv_capacity(self:  StringVector*) → int
```

Returns the current capacity of the string vector.

Parameters

self	The pointer to the StringVector.

Returns

The total capacity.

8.16.1.3 sv_clear()

```
func sv_clear(self:  StringVector*)
```

Clears the string vector and releases all held strings.

Parameters

self	The pointer to the StringVector to clear.

8.16.1.4 sv_create()

```
func sv_create(initial_capacity:  int) → StringVector*
```

Creates a new StringVector with the specified initial capacity.

Parameters

initial_capacity	The initial capacity.

Returns

A pointer to the newly created StringVector.

8.16.1.5 sv_free()

```
func sv_free(self:  StringVector*)
```

Free a `StringVector` and release every stored string.

Parameters

self	String vector to free.

8.16.1.6 sv_get()

```
func sv_get(self:  StringVector*, index:  int) → string
```

Returns the string at the specified index.

Parameters

self	The pointer to the StringVector.
index	The index to access.

Returns

The string at that index.

8.16.1.7 sv_push()

```
func sv_push(self:  StringVector*, val:  string)
```

Pushes a string onto the StringVector.

Parameters

self	The pointer to the StringVector.
val	The string value to push.

8.16.1.8 sv_size()

```
func sv_size(self:  StringVector*) → int
```

Returns the number of strings currently in the vector.

Parameters

self	The pointer to the StringVector.

Returns

The count of elements.

8.16.1.9 sv_sort()

```
func sv_sort(self:  StringVector*)
```

Sorts the strings in the vector lexicographically using insertion sort.

Parameters

self	The pointer to the StringVector to sort.

8.16.1.10 vec_capacity()

```
func vec_capacity(self:  VectorBase*) → int
```

Returns the current total capacity of the vector.

Parameters

self	The pointer to the VectorBase.

Returns

The current capacity.

8.16.1.11 vec_check()

```
func vec_check(self:  VectorBase*, index:  int)
```

Checks if the given index is within the logical bounds of the vector.

Parameters

self	The pointer to the VectorBase.
index	The index to check.

8.16.1.12 vec_clear()

```
func vec_clear(self:  VectorBase*)
```

Clears the vector, resetting its length to zero and shrinking its capacity.

Parameters

self	The pointer to the VectorBase to clear.

8.16.1.13 vec_create()

```
func vec_create(element_size:  int, initial_capacity:  int) → VectorBase*
```

Creates a new VectorBase with the specified element size and initial capacity.

Parameters

element_size	The size of each element in bytes.
initial_capacity	The initial number of elements to allocate space for.

Returns

A pointer to the newly created VectorBase.

8.16.1.14 vec_free()

```
func vec_free(self:  VectorBase*)
```

Free a `VectorBase` and its backing array.

Parameters

self	Vector to free.

8.16.1.15 vec_get()

```
func vec_get(self:  VectorBase*, index:  int) → void*
```

Returns a pointer to the element at the specified index.

Parameters

self	The pointer to the VectorBase.
index	The index of the element.

Returns

A pointer to the element data.

8.16.1.16 vec_grow()

```
func vec_grow(self:  VectorBase*)
```

Internal function to grow the vector's capacity if needed.

Parameters

self	The pointer to the VectorBase.

8.16.1.17 vec_push()

```
func vec_push(self:  VectorBase*) → void*
```

Grows the vector by one element and returns a pointer to the new slot.

Parameters

self	The pointer to the VectorBase.

Returns

A pointer to the newly added element slot.

8.16.1.18 vec_push_bool()

```
func vec_push_bool(self:  VectorBase*, val:  bool)
```

Pushes a boolean onto the vector.

Parameters

self	The pointer to the VectorBase.
val	The boolean value to push.

8.16.1.19 vec_push_byte()

```
func vec_push_byte(self:  VectorBase*, val:  byte)
```

Pushes a byte onto the vector.

Parameters

self	The pointer to the VectorBase.
val	The byte value to push.

8.16.1.20 vec_push_bytes()

```
func vec_push_bytes(self:  VectorBase*, src:  byte*, count:  int)
```

Bulk-push `count` bytes from `src` into a byte-element vector.

Reserves capacity in one step and copies with `rt_memcpy`.

Parameters

self	The pointer to the VectorBase (must have element_size == 1).
src	Pointer to the source bytes.
count	Number of bytes to push.

8.16.1.21 vec_push_int()

```
func vec_push_int(self:  VectorBase*, val:  int)
```

Pushes an integer onto the vector.

Parameters

self	The pointer to the VectorBase.
val	The integer value to push.

8.16.1.22 vec_push_ptr()

```
func vec_push_ptr(self:  VectorBase*, val:  void*)
```

Pushes a generic pointer onto the vector.

Parameters

self	The pointer to the VectorBase.
val	The pointer value to push.

8.16.1.23 vec_reserve()

```
func vec_reserve(self:  VectorBase*, total_capacity:  int)
```

Ensures the vector has at least the specified total capacity.

Parameters

self	The pointer to the VectorBase.
total_capacity	The desired total capacity.

8.16.1.24 vec_size()

```
func vec_size(self:  VectorBase*) → int
```

Returns the number of elements currently in the vector.

Parameters

self	The pointer to the VectorBase.

Returns

The number of elements.

8.16.1.25 vec_zap()

```
func vec_zap(self:  VectorBase*, index:  int)
```

Zeros out the element at the specified index.

Parameters

self	The pointer to the VectorBase.
index	The index of the element to zap.

8.16.1.26 vi_sort()

```
func vi_sort(self:  VectorBase*)
```

Sorts the elements of an integer vector in ascending order using insertion sort.

Parameters

self	The pointer to the VectorBase containing integers.

8.16.2 Variable Documentation

8.16.2.1 StringVector

```
type StringVector = VectorBase
```

8.17 compiler/shared/l0/stdlib/sys/hash.l0 File Reference

Module: `sys.hash`

Functions

- extern func rt_hash_bool(value: bool) → int
  Hash a boolean value.
- extern func rt_hash_byte(value: byte) → int
  Hash a byte value.
- extern func rt_hash_int(value: int) → int
  Hash an integer value.
- extern func rt_hash_string(value: string) → int
  Hash a string value.
- extern func rt_hash_data(data: void*, size: int) → int
  Hash raw data given a pointer and its length.
- extern func rt_hash_opt_bool(opt: bool?) → int
  Hash an optional boolean value.
- extern func rt_hash_opt_byte(opt: byte?) → int
  Hash an optional byte value.
- extern func rt_hash_opt_int(opt: int?) → int
  Hash an optional integer value.
- extern func rt_hash_opt_string(opt: string?) → int
  Hash an optional string value.
- extern func rt_hash_ptr(ptr: void*) → int
  Hash a pointer value.
- extern func rt_hash_opt_ptr(opt: void*?) → int
  Hash an optional pointer value.

8.17.1 Function Documentation

8.17.1.1 rt_hash_bool()

```
extern func rt_hash_bool(value:  bool) → int
```

Hash a boolean value.

Parameters

value	The boolean to hash.

Returns

The hash code.

8.17.1.2 rt_hash_byte()

```
extern func rt_hash_byte(value:  byte) → int
```

Hash a byte value.

Parameters

value	The byte to hash.

Returns

The hash code.

8.17.1.3 rt_hash_data()

```
extern func rt_hash_data(data:  void*, size:  int) → int
```

Hash raw data given a pointer and its length.

Parameters

data	The pointer to the data.
size	The size of the data in bytes.

Returns

The hash code.

8.17.1.4 rt_hash_int()

```
extern func rt_hash_int(value:  int) → int
```

Hash an integer value.

Parameters

value	The integer to hash.

Returns

The hash code.

8.17.1.5 rt_hash_opt_bool()

```
extern func rt_hash_opt_bool(opt:  bool?)  → int
```

Hash an optional boolean value.

Parameters

opt	The optional boolean to hash.

Returns

The hash code.

8.17.1.6 rt_hash_opt_byte()

```
extern func rt_hash_opt_byte(opt:  byte?)  → int
```

Hash an optional byte value.

Parameters

opt	The optional byte to hash.

Returns

The hash code.

8.17.1.7 rt_hash_opt_int()

```
extern func rt_hash_opt_int(opt:  int?)  → int
```

Hash an optional integer value.

Parameters

opt	The optional integer to hash.

Returns

The hash code.

8.17.1.8 rt_hash_opt_ptr()

```
extern func rt_hash_opt_ptr(opt:  void*?)  → int
```

Hash an optional pointer value.

Note: this hashes the pointer address, not the data it points to. Panics if the optional is empty.

Parameters

opt	The optional pointer to hash.

Returns

The hash code.

8.17.1.9 rt_hash_opt_string()

```
extern func rt_hash_opt_string(opt:  string?)  → int
```

Hash an optional string value.

Parameters

opt	The optional string to hash.

Returns

The hash code.

8.17.1.10 rt_hash_ptr()

```
extern func rt_hash_ptr(ptr:  void*) → int
```

Hash a pointer value.

Note: this hashes the pointer address, not the data it points to.

Parameters

ptr	The pointer to hash.

Returns

The hash code.

8.17.1.11 rt_hash_string()

```
extern func rt_hash_string(value:  string) → int
```

Hash a string value.

Parameters

value	The string to hash.

Returns

The hash code.

8.18 compiler/shared/l0/stdlib/sys/memory.l0 File Reference

Module: `sys.memory`

Functions

- extern func rt_alloc(bytes: int) → void*?

  UNSAFE: Functions in this module are unsafe and may lead to undefined behavior if used incorrectly.
- extern func rt_realloc(ptr: void*?, new_bytes: int) → void*?

  Resize an existing raw allocation to a new size.
- extern func rt_free(ptr: void*?) → void

  Free a raw allocation returned by `rt_alloc`, `rt_calloc`, or `rt_realloc`.
- extern func rt_calloc(count: int, elem_size: int) → void*?

  Allocate zero-initialized memory for multiple elements.
- extern func rt_register_foreign(ptr: void*, bytes: int, read_only: bool) → void

  Register externally owned storage with the checked pointer runtime.
- extern func rt_unregister_foreign(ptr: void*) → void

  Remove a live foreign-storage registration without freeing its payload.
- extern func rt_memcpy(dest: void*, src: void*, bytes: int) → void*

  Copy bytes from one memory location to another.
- extern func rt_memset(dest: void*, value: int, bytes: int) → void*

  Set a block of memory to a specific byte value.
- extern func rt_memcmp(a: void*, b: void*, bytes: int) → int

  Compare two blocks of memory.
- extern func rt_array_element(array_data: void*, element_size: int, index: int) → void*

  Get a pointer to an element in an array given its index.
- extern func rt_stdin_read(buf: byte*?, capacity: int) → int

  Read up to `capacity` bytes from standard input into `buf`.
- extern func rt_stdout_write(buf: byte*?, len: int) → int

  Write up to `len` bytes from `buf` to standard output.
- extern func rt_stderr_write(buf: byte*?, len: int) → int

  Write up to `len` bytes from `buf` to standard error.

8.18.1 Function Documentation

8.18.1.1 rt_alloc()

```
extern func rt_alloc(bytes:  int) → void*?
```

UNSAFE: Functions in this module are unsafe and may lead to undefined behavior if used incorrectly.

Use with caution and ensure proper memory management.

Note: Allocation wrappers register returned storage with the runtime pointer access tracker. Dereferences performed by checked generated L0 code validate the target pointer before accessing memory.

Low-level byte operations in this module still operate on raw pointers and do not perform full semantic bounds checking on their own. It is the caller's responsibility to ensure that pointers are valid and that memory is managed correctly. Misuse of these functions can lead to memory corruption, leaks, or crashes. Always validate inputs and handle memory carefully when using these functions. Allocate a new block of memory of the given size.

Parameters

bytes	The number of bytes to allocate.

Returns

A pointer to the newly allocated memory, or null on failure.

8.18.1.2 rt_array_element()

```
extern func rt_array_element(array_data:  void*, element_size:  int, index:  int) → void*
```

Get a pointer to an element in an array given its index.

Parameters

array_data	The pointer to the base of the array.
element_size	The size of each element in bytes.
index	The element index.

Returns

A pointer to the requested element.

8.18.1.3 rt_calloc()

```
extern func rt_calloc(count:  int, elem_size:  int) → void*?
```

Allocate zero-initialized memory for multiple elements.

Parameters

count	The number of elements.
elem_size	The size of each element in bytes.

Returns

A pointer to the newly allocated memory, or null on failure.

8.18.1.4 rt_free()

```
extern func rt_free(ptr:  void*?)  → void
```

Free a raw allocation returned by `rt_alloc`, `rt_calloc`, or `rt_realloc`.

Pointers returned by `new` must use `drop` instead.

Parameters

ptr	The pointer to the memory block to free.

8.18.1.5 rt_memcmp()

```
extern func rt_memcmp(a:  void*, b:  void*, bytes:  int) → int
```

Compare two blocks of memory.

Parameters

a	The first pointer.
b	The second pointer.
bytes	The number of bytes to compare.

Returns

Zero if equal, non-zero otherwise.

8.18.1.6 rt_memcpy()

```
extern func rt_memcpy(dest:  void*, src:  void*, bytes:  int) → void*
```

Copy bytes from one memory location to another.

Parameters

dest	The destination pointer.
src	The source pointer.
bytes	The number of bytes to copy.

Returns

The destination pointer.

8.18.1.7 rt_memset()

```
extern func rt_memset(dest:  void*, value:  int, bytes:  int) → void*
```

Set a block of memory to a specific byte value.

Parameters

dest	The destination pointer.
value	The byte value to set.
bytes	The number of bytes to set.

Returns

The destination pointer.

8.18.1.8 rt_realloc()

```
extern func rt_realloc(ptr:  void*?, new_bytes:  int) → void*?
```

Resize an existing raw allocation to a new size.

Pointers returned by `new`, ARC storage, and registered foreign storage are rejected in checked builds. If `ptr` is null, this behaves like `rt_alloc`.

Parameters

ptr	The pointer to the existing memory block, or null to allocate fresh.
new_bytes	The new size in bytes.

Returns

A pointer to the resized memory block, or null on failure.

8.18.1.9 rt_register_foreign()

```
extern func rt_register_foreign(ptr:  void*, bytes:  int, read_only:  bool) → void
```

Register externally owned storage with the checked pointer runtime.

Repeating an identical live registration is a no-op. The caller must unregister the storage before its external lifetime ends.

Parameters

ptr	Exact base pointer of the external storage.
bytes	Accessible byte extent, which must be positive.
read_only	Whether checked writes must be rejected.

8.18.1.10 rt_stderr_write()

```
extern func rt_stderr_write(buf:  byte*?, len:  int) → int
```

Write up to `len` bytes from `buf` to standard error.

Parameters

buf	Source buffer. A null buffer is reported as `-1`.
len	Maximum number of bytes to write.

Returns

Bytes written, or `-1` on error (including a null `buf` when `len > 0`).

8.18.1.11 rt_stdin_read()

```
extern func rt_stdin_read(buf:  byte*?, capacity:  int) → int
```

Read up to `capacity` bytes from standard input into `buf`.

Parameters

buf	Destination buffer. A null buffer is reported as `-1`.
capacity	Maximum number of bytes to read.

Returns

Bytes read, `0` on EOF, or `-1` on error (including a null `buf` when `capacity > 0`).

8.18.1.12 rt_stdout_write()

```
extern func rt_stdout_write(buf:  byte*?, len:  int) → int
```

Write up to `len` bytes from `buf` to standard output.

Parameters

buf	Source buffer. A null buffer is reported as `-1`.
len	Maximum number of bytes to write.

Returns

Bytes written, or `-1` on error (including a null `buf` when `len > 0`).

8.18.1.13 rt_unregister_foreign()

```
extern func rt_unregister_foreign(ptr:  void*) → void
```

Remove a live foreign-storage registration without freeing its payload.

Parameters

ptr	Exact base pointer previously passed to `rt_register_foreign`.

8.19 compiler/shared/l0/stdlib/sys/rt.l0 File Reference

Module: `sys.rt`

Classes

- struct RtTimeParts

  Represents components of wall or monotonic time.
- struct RtFileInfo

  Represents basic file metadata used by bootstrap tooling.

Functions

- extern func rt_string_get(s: string, index: int) → byte

  Return the character at the given index in the string as a byte (0-255).
- extern func rt_string_bytes_ptr(s: string) → byte∗

  Return a pointer to the raw byte data of a string.
- extern func rt_strlen(str: string) → int

  Return the length of the string.
- extern func rt_string_equals(a: string, b: string) → bool

  Return whether two strings are equal.
- extern func rt_string_compare(a: string, b: string) → int

  Compare two strings lexicographically.
- extern func rt_string_concat(a: string, b: string) → string

  Concatenate two strings and return the result.
- extern func rt_string_slice(s: string, start: int, end: int) → string

  Return a slice of the string from start to end (exclusive).
- extern func rt_string_from_byte_array(bytes: byte∗, len: int) → string

  Create a string from a byte array of given length.
- extern func rt_string_from_byte(b: byte) → string

  Create a string from a single byte value.
- extern func rt_string_retain(s: string) → void

  Increment reference count for heap strings.
- extern func rt_string_release(s: string) → void

  Decrement reference count for heap strings and free if count reaches zero.
- extern func rt_read_file_all(path: string) → string?

  Read a file's entire contents into a string.
- extern func rt_write_file_all(path: string, data: string) → bool

  Write a string completely to a file.
- extern func rt_flush_stdout() → void

  Flush the standard output buffer.
- extern func rt_flush_stderr() → void

  Flush the standard error buffer.
- extern func rt_print(s: string) → void

  Print a string to standard output without newline.
- extern func rt_print_stderr(s: string) → void

  Print a string to standard error without newline.
- extern func rt_println() → void

  Print a newline to standard output.

- extern func rt_println_stderr() → void

  Print a newline to standard error.
- extern func rt_print_int(x: int) → void

  Print an integer to standard output.
- extern func rt_print_int_stderr(x: int) → void

  Print an integer to standard error.
- extern func rt_print_bool(x: bool) → void

  Print a boolean to standard output.
- extern func rt_print_bool_stderr(x: bool) → void

  Print a boolean to standard error.
- extern func rt_read_line() → string?

  Read a line from standard input, returning null if EOF.
- extern func rt_read_char() → int

  Read a single character from standard input.
- extern func rt_abort(message: string) → void

  Abort program execution with a message.
- extern func rt_exit(code: int) → void

  Exit the program with the provided code.
- extern func rt_srand(seed: int) → void

  Seed the runtime random number generator.
- extern func rt_rand(max: int) → int

  Generate a random integer less than the given maximum.
- extern func rt_errno() → int

  Get the last runtime error number.
- extern func rt_get_env_var(name: string) → string?

  Get the value of an environment variable, or null if not set.
- extern func rt_get_argc() → int

  Get the number of command-line arguments.
- extern func rt_get_pid() → int

  Get the current process identifier.
- extern func rt_get_argv(i: int) → string

  Get the command-line argument at the given index.
- extern func rt_time_unix(out: RtTimeParts∗) → bool

  Capture current unix wall time into `out`.
- extern func rt_time_monotonic(out: RtTimeParts∗) → bool

  Capture monotonic time into `out`.
- extern func rt_time_monotonic_supported() → bool

  Returns true if monotonic clock is supported by the runtime.
- extern func rt_time_local_offset_sec(unix_sec: int) → int?

  Returns local UTC offset in seconds for the given unix second, or null on error.
- extern func rt_time_local_is_dst(unix_sec: int) → bool?

  Returns whether local time is in DST for the given unix second, or null on error.
- extern func rt_system(cmd: string) → int

  Execute a system command and return its normalized status code.
- extern func rt_file_info(path: string) → RtFileInfo

  Return basic metadata for a path.
- extern func rt_delete_file(path: string) → bool

  Delete the file at the given path, returning true on success.

8.19.1 Function Documentation

8.19.1.1 rt_abort()

```
extern func rt_abort(message:  string) → void
```

Abort program execution with a message.

Parameters

message	The abort message.

8.19.1.2 rt_delete_file()

```
extern func rt_delete_file(path:  string) → bool
```

Delete the file at the given path, returning true on success.

Parameters

path	The file path.

Returns

True if deleted, false otherwise.

8.19.1.3 rt_errno()

```
extern func rt_errno() → int
```

Get the last runtime error number.

Returns

The errno value.

8.19.1.4 rt_exit()

```
extern func rt_exit(code:  int) → void
```

Exit the program with the provided code.

Parameters

code	The exit code.

8.19.1.5 rt_file_info()

```
extern func rt_file_info(path:  string) → RtFileInfo
```

Return basic metadata for a path.

Parameters

path	The file-system path.

Returns

File metadata with nullable size and timestamp fields.

8.19.1.6 rt_flush_stderr()

```
extern func rt_flush_stderr() → void
```

Flush the standard error buffer.

8.19.1.7 rt_flush_stdout()

```
extern func rt_flush_stdout() → void
```

Flush the standard output buffer.

8.19.1.8 rt_get_argc()

```
extern func rt_get_argc() → int
```

Get the number of command-line arguments.

Returns

The argc value.

8.19.1.9 rt_get_argv()

```
extern func rt_get_argv(i:  int) → string
```

Get the command-line argument at the given index.

Parameters

i	The index.

Returns

The argument string.

8.19.1.10 rt_get_env_var()

```
extern func rt_get_env_var(name:  string) → string?
```

Get the value of an environment variable, or null if not set.

Parameters

name	The name of the environment variable.

Returns

The value of the variable, or null.

8.19.1.11 rt_get_pid()

```
extern func rt_get_pid() → int
```

Get the current process identifier.

Returns

The process identifier.

8.19.1.12 rt_print()

```
extern func rt_print(s:  string) → void
```

Print a string to standard output without newline.

Parameters

s	The string to print.

8.19.1.13 rt_print_bool()

```
extern func rt_print_bool(x:  bool) → void
```

Print a boolean to standard output.

Parameters

x	The boolean to print.

8.19.1.14 rt_print_bool_stderr()

```
extern func rt_print_bool_stderr(x:  bool) → void
```

Print a boolean to standard error.

Parameters

x	The boolean to print.

8.19.1.15 rt_print_int()

```
extern func rt_print_int(x:  int) → void
```

Print an integer to standard output.

Parameters

x	The integer to print.

8.19.1.16 rt_print_int_stderr()

```
extern func rt_print_int_stderr(x:  int) → void
```

Print an integer to standard error.

Parameters

x	The integer to print.

8.19.1.17 rt_print_stderr()

```
extern func rt_print_stderr(s:  string) → void
```

Print a string to standard error without newline.

Parameters

s	The string to print.

8.19.1.18 rt_println()

```
extern func rt_println() → void
```

Print a newline to standard output.

8.19.1.19 rt_println_stderr()

```
extern func rt_println_stderr() → void
```

Print a newline to standard error.

8.19.1.20 rt_rand()

```
extern func rt_rand(max:  int) → int
```

Generate a random integer less than the given maximum.

Parameters

max	The upper bound (exclusive).

Returns

A random integer.

8.19.1.21 rt_read_char()

```
extern func rt_read_char() → int
```

Read a single character from standard input.

Returns

The character read as an integer, or -1 on EOF.

8.19.1.22 rt_read_file_all()

```
extern func rt_read_file_all(path:  string) → string?
```

Read a file's entire contents into a string.

Parameters

path	The path to the file.

Returns

The file content, or null on error.

8.19.1.23 rt_read_line()

```
extern func rt_read_line() → string?
```

Read a line from standard input, returning null if EOF.

Returns

The line read, or null.

8.19.1.24 rt_srand()

```
extern func rt_srand(seed:  int) → void
```

Seed the runtime random number generator.

Parameters

seed	The integer seed.

8.19.1.25 rt_string_bytes_ptr()

```
extern func rt_string_bytes_ptr(s:  string) → byte*
```

Return a pointer to the raw byte data of a string.

Parameters

s	The string.

Returns

Pointer to the first byte.

8.19.1.26 rt_string_compare()

```
extern func rt_string_compare(a:  string, b:  string) → int
```

Compare two strings lexicographically.

Parameters

a	The first string.
b	The second string.

Returns

A negative value if a $<$ b, zero if a == b, positive if a $>$ b.

8.19.1.27 rt_string_concat()

```
extern func rt_string_concat(a:  string, b:  string) → string
```

Concatenate two strings and return the result.

Parameters

a	The first string.
b	The second string.

Returns

The combined string.

8.19.1.28 rt_string_equals()

```
extern func rt_string_equals(a:  string, b:  string) → bool
```

Return whether two strings are equal.

Parameters

a	The first string.
b	The second string.

Returns

True if equal, false otherwise.

8.19.1.29 rt_string_from_byte()

```
extern func rt_string_from_byte(b:  byte) → string
```

Create a string from a single byte value.

Parameters

b	The byte value.

Returns

A string containing the byte.

8.19.1.30 rt_string_from_byte_array()

```
extern func rt_string_from_byte_array(bytes:  byte*, len:  int) → string
```

Create a string from a byte array of given length.

Parameters

bytes	The pointer to the byte array.
len	The length of the array.

Returns

New string containing the provided bytes.

8.19.1.31 rt_string_get()

```
extern func rt_string_get(s:  string, index:  int) → byte
```

Return the character at the given index in the string as a byte (0-255).

Parameters

s	The string.
index	The index.

Returns

The character byte.

8.19.1.32 rt_string_release()

```
extern func rt_string_release(s:  string) → void
```

Decrement reference count for heap strings and free if count reaches zero.

No-op for static or non-reference-counted strings.

Parameters

s	The string to release.

8.19.1.33 rt_string_retain()

```
extern func rt_string_retain(s:  string) → void
```

Increment reference count for heap strings.

No-op for static or non-reference-counted strings.

Parameters

s	The string to retain.

8.19.1.34 rt_string_slice()

```
extern func rt_string_slice(s:  string, start:  int, end:  int) → string
```

Return a slice of the string from start to end (exclusive).

Parameters

s	The string.
start	The starting index.
end	The ending index.

Returns

The string slice.

8.19.1.35 rt_strlen()

```
extern func rt_strlen(str:  string) → int
```

Return the length of the string.

Parameters

str	The string.

Returns

The length.

8.19.1.36 rt_system()

```
extern func rt_system(cmd:  string) → int
```

Execute a system command and return its normalized status code.

Parameters

cmd	The command string to execute.

Returns

The command exit code, or a normalized signal-style status.

8.19.1.37 rt_time_local_is_dst()

```
extern func rt_time_local_is_dst(unix_sec:  int) → bool?
```

Returns whether local time is in DST for the given unix second, or null on error.

Parameters

unix_sec	The Unix timestamp in seconds.

Returns

True if DST, false if not, or null.

8.19.1.38 rt_time_local_offset_sec()

```
extern func rt_time_local_offset_sec(unix_sec:  int) → int?
```

Returns local UTC offset in seconds for the given unix second, or null on error.

Parameters

unix_sec	The Unix timestamp in seconds.

Returns

The offset in seconds, or null.

8.19.1.39 rt_time_monotonic()

```
extern func rt_time_monotonic(out:  RtTimeParts*) → bool
```

Capture monotonic time into `out`.

Parameters

out	Pointer to the RtTimeParts to fill.

Returns

True on success, false otherwise.

8.19.1.40 rt_time_monotonic_supported()

```
extern func rt_time_monotonic_supported() → bool
```

Returns true if monotonic clock is supported by the runtime.

Returns

True if supported.

8.19.1.41 rt_time_unix()

```
extern func rt_time_unix(out:  RtTimeParts*) → bool
```

Capture current unix wall time into `out`.

Parameters

out	Pointer to the RtTimeParts to fill.

Returns

True on success, false otherwise.

8.19.1.42 rt_write_file_all()

```
extern func rt_write_file_all(path:  string, data:  string) → bool
```

Write a string completely to a file.

Parameters

path	The path to the file.
data	The string content.

Returns

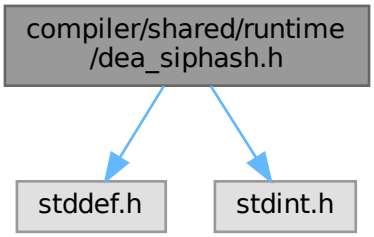
True on success, false otherwise.

8.20 compiler/shared/runtime/dea_siphash.h File Reference

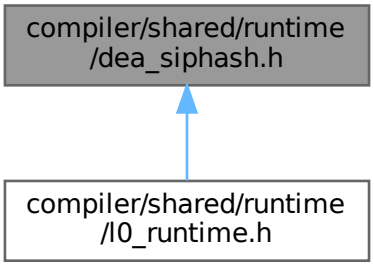
SipHash-1-3 and SipHash-2-4 (64-bit tag), portable C99.

```
#include <stddef.h>
#include <stdint.h>
```

Include dependency graph for dea_siphash.h:



This graph shows which files directly or indirectly include this file:



Functions

- uint64_t siphash24 (const void *data, size_t len, const uint8_t key[16])

  SipHash-2-4 (64-bit tag).
- uint64_t siphash13 (const void *data, size_t len, const uint8_t key[16])

  SipHash-1-3 (64-bit tag).
- uint64_t siphash24_tag8_bytes (const uint8_t tag8[8], const void *data, size_t len, const uint8_t key[16])

Hash (tag8 || data) where tag8 is a raw 8-byte array using SipHash-2-4.

- uint64_t siphash13_tag8_bytes (const uint8_t tag8[8], const void *data, size_t len, const uint8_t key[16])

  Hash (tag8 || data) where tag8 is a raw 8-byte array using SipHash-1-3.
- uint64_t siphash24_tag8_bf (const uint8_t tag8[8], const uint8_t flags, const void *data, size_t len, const uint8_t key[16])

  Hash (tag8^ flags || data) where tag8 is a raw 8-byte array and flags is one byte (XORed into tag8[0]).
- uint64_t siphash13_tag8_bf (const uint8_t tag8[8], const uint8_t flags, const void *data, size_t len, const uint8_t key[16])

  Hash (tag8^ flags || data) where tag8 is a raw 8-byte array and flags is one byte (XORed into tag8[0]).

8.20.1 Detailed Description

SipHash-1-3 and SipHash-2-4 (64-bit tag), portable C99.

Header-only / single-header pattern:

- In one translation unit: `#define SIPHASH_IMPLEMENTATION` before including this header.
- Everywhere else: include this header without `SIPHASH_IMPLEMENTATION`.

8.20.2 Function Documentation

8.20.2.1 siphash13()

```
uint64_t siphash13 (
            const void * data,
            size_t len,
            const uint8_t key[16] )
```

SipHash-1-3 (64-bit tag).

Parameters

data	Pointer to the data to hash.
len	Length of the data in bytes.
key	16-byte key, interpreted as little-endian (k0 = key[0..7], k1 = key[8..15]).

Returns

64-bit hash value.

8.20.2.2 siphash13_tag8_bf()

```
uint64_t siphash13_tag8_bf (
            const uint8_t tag8[8],
            const uint8_t flags,
            const void * data,
```

```
            size_t len,
            const uint8_t key[16] )
```

Hash (tag8^flags || data) where tag8 is a raw 8-byte array and flags is one byte (XORed into tag8[0]).

Uses SipHash-1-3.

Parameters

tag8	8-byte tag/prefix.
flags	One byte flags XORed into tag8[0].
data	Pointer to the data to hash.
len	Length of the data in bytes.
key	16-byte key.

Returns

64-bit hash value.

8.20.2.3 siphash13_tag8_bytes()

```
uint64_t siphash13_tag8_bytes (
            const uint8_t tag8[8],
            const void * data,
            size_t len,
            const uint8_t key[16] )
```

Hash (tag8 || data) where tag8 is a raw 8-byte array using SipHash-1-3.

Parameters

tag8	8-byte tag/prefix.
data	Pointer to the data to hash.
len	Length of the data in bytes.
key	16-byte key.

Returns

64-bit hash value.

8.20.2.4 siphash24()

```
uint64_t siphash24 (
            const void * data,
            size_t len,
            const uint8_t key[16] )
```

SipHash-2-4 (64-bit tag).

Parameters

data	Pointer to the data to hash.
len	Length of the data in bytes.
key	16-byte key, interpreted as little-endian (k0 = key[0..7], k1 = key[8..15]).

Returns

64-bit hash value.

8.20.2.5 siphash24_tag8_bf()

```
uint64_t siphash24_tag8_bf (
            const uint8_t tag8[8],
            const uint8_t flags,
            const void * data,
            size_t len,
            const uint8_t key[16] )
```

Hash (tag8^flags || data) where tag8 is a raw 8-byte array and flags is one byte (XORed into tag8[0]).

Uses SipHash-2-4.

Parameters

tag8	8-byte tag/prefix.
flags	One byte flags XORed into tag8[0].
data	Pointer to the data to hash.
len	Length of the data in bytes.
key	16-byte key.

Returns

64-bit hash value.

8.20.2.6 siphash24_tag8_bytes()

```
uint64_t siphash24_tag8_bytes (
            const uint8_t tag8[8],
            const void * data,
            size_t len,
            const uint8_t key[16] )
```

Hash (tag8 || data) where tag8 is a raw 8-byte array using SipHash-2-4.

Parameters

tag8	8-byte tag/prefix.
data	Pointer to the data to hash.
len	Length of the data in bytes.
key	16-byte key.

Returns

64-bit hash value.

8.21 dea_siphash.h

Go to the documentation of this file.

```
00001 #ifndef SIPHASH_C99_SIPHASH_H
00002 #define SIPHASH_C99_SIPHASH_H
00003
00004 /*
00005  * SPDX-License-Identifier: MIT OR Apache-2.0
00006  * Copyright (c) 2025-2026 gwz
00007  */
00008
00018 #include <stddef.h>
00019 #include <stdint.h>
00020
00021 #ifdef __cplusplus
00022 extern "C" {
00023 #endif
00024
00033 uint64_t siphash24(const void *data, size_t len, const uint8_t key[16]);
00034
00043 uint64_t siphash13(const void *data, size_t len, const uint8_t key[16]);
00044
00054 uint64_t siphash24_tag8_bytes(const uint8_t tag8[8], const void *data, size_t len, const uint8_t
      key[16]);
00055
00065 uint64_t siphash13_tag8_bytes(const uint8_t tag8[8], const void *data, size_t len, const uint8_t
      key[16]);
00066
00078 uint64_t siphash24_tag8_bf(const uint8_t tag8[8], const uint8_t flags, const void *data, size_t len,
      const uint8_t key[16]);
00079
00091 uint64_t siphash13_tag8_bf(const uint8_t tag8[8], const uint8_t flags, const void *data, size_t len,
      const uint8_t key[16]);
00092
00093
00094 #ifdef __cplusplus
00095 }
00096 #endif
00097
00098 #ifdef SIPHASH_IMPLEMENTATION
00099
00100 #ifdef __cplusplus
00101 extern "C" {
00102 #endif
00103
00111 static uint64_t sh_rotl64(uint64_t x, unsigned b) {
00112   return x << b | x >> (64u - b);
00113 }
00114
00121 static uint64_t sh_load64_le(const uint8_t *p) {
00122   return (uint64_t)p[0]       |
00123          (uint64_t)p[1] << 8  |
00124          (uint64_t)p[2] << 16 |
00125          (uint64_t)p[3] << 24 |
00126          (uint64_t)p[4] << 32 |
00127          (uint64_t)p[5] << 40 |
00128          (uint64_t)p[6] << 48 |
00129          (uint64_t)p[7] << 56;
00130 }
00131
00135 #define SH_SIPROUND(v0, v1, v2, v3)   \
00136   do {                                \
00137     (v0) += (v1);                     \
00138     (v2) += (v3);                     \
00139     (v1) = sh_rotl64((v1), 13);       \
00140     (v3) = sh_rotl64((v3), 16);       \
00141     (v1) ^= (v0);                     \
00142     (v3) ^= (v2);                     \
00143     (v0) = sh_rotl64((v0), 32);       \
00144     (v2) += (v1);                     \
00145     (v0) += (v3);                     \
00146     (v1) = sh_rotl64((v1), 17);       \
00147     (v3) = sh_rotl64((v3), 21);       \
00148     (v1) ^= (v2);                     \
00149     (v3) ^= (v0);                     \
00150     (v2) = sh_rotl64((v2), 32);       \
00151   } while (0)
```

```
00152
00165 static uint64_t sh_siphash_cd(const void *data, size_t len,
00166                              uint64_t k0, uint64_t k1,
00167                              int c_rounds, int d_rounds)
00168 {
00169   const uint8_t *in = (const uint8_t *)data;
00170   const uint8_t *end = in + (len & ~(size_t)7);
00171
00172   /* SipHash initialization vectors (from the SipHash specification). */
00173   uint64_t v0 = 0x736f6d6570736575ULL ^ k0;
00174   uint64_t v1 = 0x646f72616e646f6dULL ^ k1;
00175   uint64_t v2 = 0x6c7967656e657261ULL ^ k0;
00176   uint64_t v3 = 0x7465646279746573ULL ^ k1;
00177
00178   for (; in != end; in += 8) {
00179     uint64_t m = sh_load64_le(in);
00180     v3 ^= m;
00181     for (int i = 0; i < c_rounds; i++) SH_SIPROUND(v0, v1, v2, v3);
00182     v0 ^= m;
00183   }
00184
00185   /* Final block: top byte is message length mod 256, lower bytes are remaining tail (per SipHash
      spec). */
00186   /* In other words, b = (len << 56) | tail */
00187   uint64_t b = ((uint64_t)len) << 56;
00188   switch (len & 7) {
00189   case 7: b |= (uint64_t)in[6] << 48; /* fallthrough */
00190   case 6: b |= (uint64_t)in[5] << 40; /* fallthrough */
00191   case 5: b |= (uint64_t)in[4] << 32; /* fallthrough */
00192   case 4: b |= (uint64_t)in[3] << 24; /* fallthrough */
00193   case 3: b |= (uint64_t)in[2] << 16; /* fallthrough */
00194   case 2: b |= (uint64_t)in[1] << 8;  /* fallthrough */
00195   case 1: b |= (uint64_t)in[0]; /* fallthrough */
00196   default: break;
00197   }
00198
00199   v3 ^= b;
00200   for (int i = 0; i < c_rounds; i++) SH_SIPROUND(v0, v1, v2, v3);
00201   v0 ^= b;
00202
00203   /* Finalization marker (per SipHash spec). */
00204   v2 ^= 0xff;
00205   for (int i = 0; i < d_rounds; i++) SH_SIPROUND(v0, v1, v2, v3);
00206
00207   return (v0 ^ v1) ^ (v2 ^ v3);
00208 }
00209
00217 static void sh_key_to_k01(const uint8_t key[16], uint64_t *k0, uint64_t *k1) {
00218   *k0 = sh_load64_le(&key[0]);
00219   *k1 = sh_load64_le(&key[8]);
00220 }
00221
00225 uint64_t siphash24(const void *data, size_t len, const uint8_t key[16]) {
00226   uint64_t k0, k1;
00227   sh_key_to_k01(key, &k0, &k1);
00228   return sh_siphash_cd(data, len, k0, k1, 2, 4);
00229 }
00230
00234 uint64_t siphash13(const void *data, size_t len, const uint8_t key[16]) {
00235   uint64_t k0, k1;
00236   sh_key_to_k01(key, &k0, &k1);
00237   return sh_siphash_cd(data, len, k0, k1, 1, 3);
00238 }
00239
00252 static uint64_t sh_siphash_cd_tag8(const void *data, size_t len,
00253                                    uint64_t k0, uint64_t k1,
00254                                    uint64_t tag8_le,
00255                                    int c_rounds, int d_rounds) {
00256   const uint8_t *in = (const uint8_t *)data;
00257   const uint8_t *end = in + (len & ~(size_t)7);
00258
00259   /* SipHash initialization vectors (from the SipHash specification). */
00260   uint64_t v0 = 0x736f6d6570736575ULL ^ k0;
00261   uint64_t v1 = 0x646f72616e646f6dULL ^ k1;
00262   uint64_t v2 = 0x6c7967656e657261ULL ^ k0;
00263   uint64_t v3 = 0x7465646279746573ULL ^ k1;
00264
00265   /* First message block: the 8-byte tag/prefix. */
00266   {
00267     uint64_t m = tag8_le;
00268     v3 ^= m;
00269     for (int i = 0; i < c_rounds; i++) SH_SIPROUND(v0, v1, v2, v3);
00270     v0 ^= m;
00271   }
00272
00273   /* Process data blocks. */
00274   for (; in != end; in += 8) {
```

```
00275     uint64_t m = sh_load64_le(in);
00276     v3 ^= m;
00277     for (int i = 0; i < c_rounds; i++) SH_SIPROUND(v0, v1, v2, v3);
00278     v0 ^= m;
00279   }
00280
00281   /* Final block length includes the 8-byte tag. */
00282   uint64_t total_len = (uint64_t)len + 8u;
00283   uint64_t b = total_len << 56;
00284
00285   switch (len & 7) {
00286     case 7: b |= (uint64_t)in[6] << 48; /* fallthrough */
00287     case 6: b |= (uint64_t)in[5] << 40; /* fallthrough */
00288     case 5: b |= (uint64_t)in[4] << 32; /* fallthrough */
00289     case 4: b |= (uint64_t)in[3] << 24; /* fallthrough */
00290     case 3: b |= (uint64_t)in[2] << 16; /* fallthrough */
00291     case 2: b |= (uint64_t)in[1] << 8;  /* fallthrough */
00292     case 1: b |= (uint64_t)in[0]; break;
00293     default: break;
00294   }
00295
00296   v3 ^= b;
00297   for (int i = 0; i < c_rounds; i++) SH_SIPROUND(v0, v1, v2, v3);
00298   v0 ^= b;
00299
00300   v2 ^= 0xff;
00301   for (int i = 0; i < d_rounds; i++) SH_SIPROUND(v0, v1, v2, v3);
00302
00303   return (v0 ^ v1) ^ (v2 ^ v3);
00304 }
00305
00309 uint64_t siphash24_tag8_u64(uint64_t tag8_le, const void *data, size_t len, const uint8_t key[16]) {
00310   uint64_t k0, k1;
00311   sh_key_to_k01(key, &k0, &k1);
00312   return sh_siphash_cd_tag8(data, len, k0, k1, tag8_le, 2, 4);
00313 }
00314
00318 uint64_t siphash13_tag8_u64(uint64_t tag8_le, const void *data, size_t len, const uint8_t key[16]) {
00319   uint64_t k0, k1;
00320   sh_key_to_k01(key, &k0, &k1);
00321   return sh_siphash_cd_tag8(data, len, k0, k1, tag8_le, 1, 3);
00322 }
00323
00327 uint64_t siphash24_tag8_bytes(const uint8_t tag8[8], const void *data, size_t len, const uint8_t
      key[16]) {
00328   return siphash24_tag8_u64(sh_load64_le(tag8), data, len, key);
00329 }
00330
00334 uint64_t siphash13_tag8_bytes(const uint8_t tag8[8], const void *data, size_t len, const uint8_t
      key[16]) {
00335   return siphash13_tag8_u64(sh_load64_le(tag8), data, len, key);
00336 }
00337
00341 uint64_t siphash24_tag8_bf(const uint8_t tag8[8], const uint8_t flags, const void *data, size_t len,
      const uint8_t key[16]) {
00342   return siphash24_tag8_u64(sh_load64_le(tag8)^flags, data, len, key);
00343 }
00344
00348 uint64_t siphash13_tag8_bf(const uint8_t tag8[8], const uint8_t flags, const void *data, size_t len,
      const uint8_t key[16]) {
00349   return siphash13_tag8_u64(sh_load64_le(tag8)^flags, data, len, key);
00350 }
00351
00352 #ifdef __cplusplus
00353 }
00354 #endif
00355
00356 #endif /* SIPHASH_IMPLEMENTATION */
00357
00358 #endif /* SIPHASH_C99_SIPHASH_H */
```

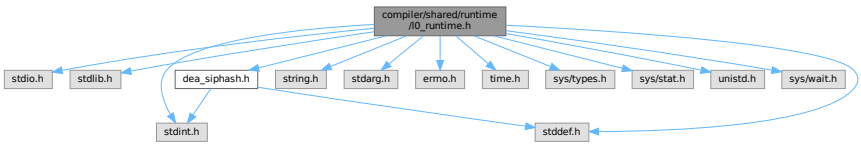
8.22 compiler/shared/runtime/l0_runtime.h File Reference

L0 Runtime Library (K0 - Kernel Layer)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stddef.h>
```

```
#include <string.h>
#include <stdarg.h>
#include <errno.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/wait.h>
#include "dea_siphash.h"
```

Include dependency graph for l0_runtime.h:



Classes

- struct _l0_h_string

  Heap-allocated L0 string header.
- struct l0_string

  Unified L0 string type (static or heap-allocated).
- struct l0_opt_bool

  Optional boolean wrapper.
- struct l0_opt_byte

  Optional byte wrapper.
- struct l0_opt_int

  Optional integer wrapper.
- struct l0_opt_string

  Optional string wrapper.
- struct _l0_base_opt

  Base structure for optional types to access `has_value`.
- struct l0_sys_rt_RtTimeParts

  Definition for `sys.rt::RtTimeParts`.
- struct l0_sys_rt_RtFileInfo

  Definition for `sys.rt::RtFileInfo`.
- struct _rt_alloc_record

  Hot allocation fields read by checked pointer cache hits and allocation tracker mutation paths.
- struct _rt_alloc_record_cold

  Cold allocation fields used by diagnostics and rare metadata reads.
- struct _rt_ptr_site

  Per-call-site pointer check cache.

Macros

- #define DEA_RUNTIME_FCLOSE fclose
- #define DEA_RUNTIME_FILE_INFO_STAT stat
- #define DEA_RUNTIME_REMOVE remove
- #define _RT_ALIGNOF(type) (sizeof(struct { char _rt_align_c; type _rt_align_v; }) - sizeof(type))
- #define L0_UNREACHABLE(_s) rt_panic(_s)
- #define _RT_TRACE_ARC(...) ((void)0)
- #define _RT_TRACE_ARC_LOC(loc_file, loc_line, ...) ((void)0)
- #define _RT_TRACE_MEM(...) ((void)0)
- #define _RT_TRACE_MEM_LOC(loc_file, loc_line, ...) ((void)0)
- #define L0_STRING_K_STATIC 0
- #define L0_STRING_K_HEAP 1
- #define L0_STRING_CONST(str_data, str_len) { .kind = L0_STRING_K_STATIC, .data.s_str = { .len = (str↩ _len), .bytes = (str_data) } }

  String literal construction macro.
- #define L0_OPT_BOOL_DEFINED
- #define L0_OPT_BYTE_DEFINED
- #define L0_OPT_INT_DEFINED
- #define L0_OPT_STRING_DEFINED
- #define L0_DEFINED_l0_sys_rt_RtTimeParts
- #define L0_DEFINED_l0_sys_rt_RtFileInfo
- #define _RT_ALLOC_INIT_CAP 256
- #define _RT_ALLOC_LIVE 1
- #define _RT_ALLOC_QUARANTINED 2
- #define _RT_ALLOC_POOLED 3
- #define _RT_MEM_RAW 0
- #define _RT_MEM_NEW 1
- #define _RT_MEM_ARC 2
- #define _RT_MEM_STATIC 3
- #define _RT_MEM_FOREIGN 4
- #define _RT_ACCESS_READ 0
- #define _RT_ACCESS_WRITE 1
- #define _RT_ACCESS_UNTRACKED_OK 2
- #define _RT_QUARANTINE_MAX_BYTES ((size_t)16 * 1024 * 1024)
- #define _RT_QUARANTINE_MAX_COUNT ((size_t)4096)
- #define _RT_REC_POOL_CHUNK 256
- #define _RT_ALLOC_TOMBSTONE ((_rt_alloc_record*)(uintptr_t)1)
- #define _rt_rec_cold(rec) (&_rt_cold_chunks[(rec)→cold_index >> 8][(rec)→cold_index & 255u])
- #define _L0_TAG_OPT 0x80 /* option */
- #define _L0_TAG_PTR 0x40 /* pointer */
- #define _L0_TAG_ENUM 0x20 /* enum */
- #define _L0_TAG_STRUCT 0x10 /* struct */

Typedefs

- typedef uint8_t l0_bool
- typedef int8_t l0_tiny

  future use
- typedef int16_t l0_short
- typedef int32_t l0_int
- typedef int64_t l0_long
- typedef uint8_t l0_byte
- typedef uint16_t l0_ushort

- typedef uint32_t l0_uint
- typedef uint64_t l0_ulong
- typedef float l0_float
- typedef double l0_double
- typedef struct _rt_alloc_record _rt_alloc_record
- typedef struct _rt_alloc_record_cold _rt_alloc_record_cold
- typedef struct _rt_ptr_site _rt_ptr_site
- typedef char _rt_alloc_record_size_check[(sizeof(_rt_alloc_record)==64) ? 1 :-1]
- typedef uint8_t _rt_siphash_key_t[16]
- typedef uint8_t _rt_siphash_tag8_t[8]

Functions

- void _rt_init_args (int argc, char **argv)

  Initialize command-line arguments.
- static void _rt_panic (const char *message)

  Abort the program with a message.
- static void _rt_panic_fmt (const char *fmt,...)

  Abort the program with a formatted message.
- static l0_int _rt_idiv (l0_int a, l0_int b)

  Safe integer division.
- static l0_int _rt_imod (l0_int a, l0_int b)

  Safe integer modulo.
- static l0_int _rt_iadd (l0_int a, l0_int b)

  Safe integer addition with overflow check.
- static l0_int _rt_isub (l0_int a, l0_int b)

  Safe integer subtraction with overflow check.
- static l0_int _rt_imul (l0_int a, l0_int b)

  Safe integer multiplication with overflow check.
- l0_byte _rt_narrow_l0_byte (l0_int value)

  Narrow l0_int to l0_byte with range check.
- static void * _unwrap_ptr (void *opt, const char *type_name)

  Unwrap a pointer, panicking if NULL.
- static void * _unwrap_opt (void *opt_ptr, const char *type_name)

  Unwrap an optional type structure, panicking if it has no value.
- static void _rt_track_alloc_record (void *ptr, size_t size, size_t align, uint32_t type_id, const char *loc_file, int loc_line)
- static void _rt_promote_new_alloc (void *ptr)
- static void _rt_release_tracked_alloc (void *ptr, const char *loc_file, int loc_line, const char *op_name)
- static void * _rt_realloc_tracked_alloc (void *ptr, size_t new_size, const char *loc_file, int loc_line)
- static void * _rt_check_ptr_site_slow (_rt_ptr_site *site, void *ptr, l0_int required_size, l0_int required_align, int access_mode, const char *loc_file, int loc_line)
- static void * _rt_check_index_ptr_site_slow (_rt_ptr_site *site, void *base_ptr, l0_int index, l0_int element↩ _size, l0_int required_align, int access_mode, const char *loc_file, int loc_line)
- static void * _rt_drop_begin_impl (void *ptr, l0_int required_size, l0_int required_align, const char *loc_file, int loc_line)
- static void _rt_drop_finish_impl (void *ptr, const char *loc_file, int loc_line)
- static void * _rt_validate_derived_ptr (void *derived, void *parent_base, l0_int size, l0_int align, const char *loc_file, int loc_line)
- static void _rt_track_arc_bytes (void *ptr, size_t size)

  Lazily register ARC-managed string storage as a read-only tracked record.
- static void _rt_untrack_arc_alloc (void *ptr)

- static void _rt_track_static_bytes (const void *ptr, size_t size)
- static void * _rt_check_ptr_site (_rt_ptr_site *site, void *ptr, l0_int required_size, l0_int required_align, int access_mode, const char *loc_file, int loc_line)

  Validate one pointer access through a call-site cache.
- static void * _rt_check_index_ptr_site (_rt_ptr_site *site, void *base_ptr, l0_int index, l0_int element_size, l0_int required_align, int access_mode, const char *loc_file, int loc_line)

  Validate and derive one indexed pointer access through a call-site cache.
- static l0_string _rt_l0_string_from_const_literal (const char *c_str)

  Create an L0 string from a constant C string.
- static l0_string _rt_init_heap_string (void *mem, l0_int s_len)

  Initialize a heap-allocated L0_string in the given memory.
- static l0_string _rt_alloc_string (l0_int len)

  Allocate a new reference counted L0_string of the given length.
- static void _rt_free_string (l0_string str)

  Free a string's allocated data, if applicable.
- static l0_string _rt_realloc_string (l0_string s, l0_int new_len)

  Reallocate a heap string to a new length.
- static l0_string _rt_new_l0_string (const char *c_str)

  Create a new reference counted L0_string from a null-terminated C string.
- static char * _rt_string_bytes (l0_string s)

  Gets the null-terminated C string underlying an L0 string.
- static l0_int rt_strlen (l0_string str)

  Get the length of a string.
- static l0_byte rt_string_get (l0_string a, l0_int index)

  Bounds-checked character access.
- static l0_byte * rt_string_bytes_ptr (l0_string s)

  Return a pointer to the raw byte data of a string.
- static l0_bool rt_string_equals (l0_string a, l0_string b)

  Check if two strings are equal.
- static l0_int rt_string_compare (l0_string a, l0_string b)

  Compare two strings lexicographically.
- static l0_string rt_string_concat (l0_string a, l0_string b)

  Concatenate two strings (allocates new memory).
- static l0_string rt_string_slice (l0_string s, l0_int start, l0_int end)

  Create a substring (allocates new memory).
- static l0_string rt_string_from_byte (l0_byte b)

  Create an L0 string from a single character (byte).
- static l0_string rt_string_from_byte_array (l0_byte *bytes, l0_int len)

  Create an L0 string from a byte array and a length.
- static void rt_string_retain (l0_string s)

  Increment reference count for heap strings (no-op for static).
- static void rt_string_release (l0_string s)

  Decrement reference count, freeing if zero.
- static l0_int rt_system (l0_string cmd)

  Execute a system command and return its normalized status.
- static l0_opt_string rt_get_env_var (l0_string name)

  Get an environment variable as an L0 optional string.
- static l0_int rt_get_argc (void)

  Get the number of command-line arguments.
- static l0_bool _rt_pid_to_l0_int (intmax_t value, l0_int *out)

  Convert a native process identifier into `l0_int`.

- static l0_int rt_get_pid (void)

  Get the current process identifier.
- static l0_string rt_get_argv (l0_int i)

  Get the command-line argument at the given index.
- static l0_bool _rt_time_to_l0_int_sec (time_t value, l0_int *out)

  Internal helper to convert time_t to l0_int seconds.
- static l0_bool _rt_time_to_l0_int_nsec (long value, l0_int *out)

  Internal helper to convert long to l0_int nanoseconds.
- static l0_bool _rt_time_write_parts (struct l0_sys_rt_RtTimeParts *out, l0_int sec, l0_int nsec)

  Internal helper to write time parts to struct.
- static l0_bool rt_time_unix (struct l0_sys_rt_RtTimeParts *out)

  Capture current unix wall clock into `out`.
- static l0_bool rt_time_monotonic (struct l0_sys_rt_RtTimeParts *out)

  Capture current monotonic clock into `out`.
- static l0_bool rt_time_monotonic_supported (void)

  Returns whether a monotonic clock source is available.
- static l0_opt_int rt_time_local_offset_sec (l0_int unix_sec)

  Returns local UTC offset in seconds for `unix_sec`.
- static l0_opt_bool rt_time_local_is_dst (l0_int unix_sec)

  Returns whether local time is daylight-saving time for `unix_sec`.
- static l0_opt_string rt_read_file_all (l0_string path)

  Read entire file contents into a string.
- static l0_bool rt_write_file_all (l0_string path, l0_string data)

  Write string data to a file.
- static struct l0_sys_rt_RtFileInfo rt_file_info (l0_string path)

  Return basic metadata for a path.
- static l0_bool rt_delete_file (l0_string path)

  Delete the file at the given path.
- static l0_int _rt_stream_write_some (FILE *stream, const l0_byte *buf, l0_int len)

  Write raw bytes to one standard stream.
- static l0_int rt_stdin_read (l0_byte *buf, l0_int capacity)

  Read raw bytes from standard input.
- static l0_int rt_stdout_write (l0_byte *buf, l0_int len)

  Write raw bytes to standard output.
- static l0_int rt_stderr_write (l0_byte *buf, l0_int len)

  Write raw bytes to standard error.
- static void rt_flush_stdout (void)

  Flush stdout.
- static void rt_flush_stderr (void)

  Flush stderr.
- void _rt_print (l0_string s, FILE *stream)

  Internal helper to print an l0_string to a given stream.
- static void rt_print (l0_string s)

  Print a string to stdout.
- static void rt_print_stderr (l0_string s)

  Print a string to stderr.
- static void rt_println (void)

  Print a newline to stdout.
- static void rt_println_stderr (void)

  Print a newline to stderr.
- static void rt_print_int (l0_int x)

Print an integer to stdout.

- static void rt_print_int_stderr (l0_int x)

  Print an integer to stderr.
- static void rt_print_bool (l0_bool x)

  Print a bool to stdout.
- static void rt_print_bool_stderr (l0_bool x)

  Print a bool to stderr.
- static l0_opt_string rt_read_line (void)

  Read a line from stdin into a dynamically allocated buffer.
- static l0_int rt_read_char (void)

  Read one character from stdin.
- static void rt_abort (l0_string message)

  Abort the program with a panic message.
- static void rt_exit (l0_int code)

  Exit the program with the given exit code.
- static void rt_srand (l0_int seed)

  Seed the random number generator.
- static l0_int rt_rand (l0_int max)

  Generate a random integer in the range [0, max).
- static l0_int rt_errno (void)

  Get the current errno value.
- static void * rt_alloc (l0_int bytes)

  Allocate memory of the given size in bytes.
- static void * rt_realloc (void *ptr, l0_int new_bytes)

  Reallocate a raw runtime allocation to a new size.
- static void rt_free (void *ptr)

  Free a raw runtime allocation.
- static void * rt_calloc (l0_int count, l0_int elem_size)

  Allocate zeroed memory for an array of elements.
- static void * rt_memset (void *dest, l0_int value, l0_int bytes)

  Set memory to a specific byte value.
- static void * rt_memcpy (void *dest, void *src, l0_int bytes)

  Copy memory from source to destination.
- static l0_int rt_memcmp (void *a, void *b, l0_int bytes)

  Compare two memory regions.
- static void * rt_array_element (void *array_data, l0_int element_size, l0_int index)

  Get a pointer to an element in an array.
- static _rt_alloc_record * _rt_rec_new (_rt_alloc_record_cold **cold_out)

  Take one record from the pool, refilling it chunk-wise.
- static void _rt_rec_recycle (_rt_alloc_record *rec)
- static size_t _rt_alloc_hash (void *ptr, size_t cap)
- static void _rt_alloc_table_rehash (void)

  Rebuild the hash table sized from the live record count.
- static _rt_alloc_record * _rt_alloc_table_lookup (void *ptr)
- static void _rt_alloc_table_insert (_rt_alloc_record *rec)
- static void _rt_alloc_table_remove (_rt_alloc_record *target)
- static void _rt_panic_invalid_access (const char *reason, void *ptr, const char *loc_file, int loc_line)
- static void _rt_panic_invalid_drop (const char *reason, void *ptr, const char *loc_file, int loc_line)
- static void _rt_panic_invalid_release (const char *op_name, const char *reason, void *ptr, const char *loc↩ _file, int loc_line)
- static void _rt_trace_invalid_drop (void *ptr, const char *loc_file, int loc_line)

- static int _rt_ptr_is_aligned (void *ptr, size_t align)
- static int _rt_range_contains (void *base, size_t total_size, void *ptr, size_t need_size, size_t *offset_out)
- static uint32_t _rt_tree_prio_for (void *base)
- static _rt_alloc_record * _rt_tree_insert_at (_rt_alloc_record *node, _rt_alloc_record *rec)
- static _rt_alloc_record * _rt_tree_merge (_rt_alloc_record *a, _rt_alloc_record *b)

  Merge two treaps where every base in `a` is below every base in `b`.
- static _rt_alloc_record * _rt_tree_remove_at (_rt_alloc_record *node, void *base)
- static void _rt_alloc_tree_insert (_rt_alloc_record *rec)
- static void _rt_alloc_tree_remove (_rt_alloc_record *rec)
- static _rt_alloc_record * _rt_alloc_tree_glb (void *ptr)

  Return the tracked record with the greatest base at or below `ptr`.
- static _rt_alloc_record * _rt_alloc_tree_lub (void *ptr)

  Return the tracked record with the lowest base at or above `ptr`.
- static _rt_alloc_record * _rt_alloc_tree_find_containing (void *ptr, size_t need_size)
- static void _rt_ptr_site_store (_rt_ptr_site *site, _rt_alloc_record *owner)
- static size_t _rt_required_size (l0_int required_size)
- static size_t _rt_required_align (l0_int required_align)
- static size_t _rt_index_element_size (l0_int element_size)
- static size_t _rt_index_offset (l0_int index, size_t element_size)
- static int _rt_required_access_mode (int access_mode)
- static int _rt_index_access_mode (int access_mode)
- static int _rt_index_allows_untracked (int access_mode)
- static void _rt_check_ptr_align (void *ptr, size_t need_align, const char *loc_file, int loc_line)

  Panic when a validated access target misses its required alignment.
- static void _rt_check_record_writeable (_rt_alloc_record *rec, void *ptr, int access_mode, const char *loc↩ _file, int loc_line)
- static void _rt_track_alloc_record_kind (void *ptr, size_t size, size_t align, uint32_t type_id, int mem_kind, int read_only, const char *loc_file, int loc_line)
- static void rt_register_foreign (void *ptr, l0_int bytes, l0_bool read_only)

  Register externally owned storage for checked generated pointer accesses.
- static void rt_unregister_foreign (void *ptr)

  Unregister externally owned storage without releasing its payload.
- static void _rt_evict_quarantine (void)
- static void _rt_quarantine_alloc_record (_rt_alloc_record *rec, _rt_alloc_record_cold *cold, const char *loc↩ _file, int loc_line)
- static void * _rt_alloc_obj (l0_int bytes)
- static uint32_t _rt_fmix32 (uint32_t x)

  Final mixing function for 32-bit hashes (MurmurHash3 fmix32).
- static uint32_t _rt_fold_u64_to_u32_fmix (uint64_t h)

  Fold a 64-bit hash into a 32-bit hash with final mixing.
- static l0_int _rt_hash_tag8 (const _rt_siphash_tag8_t tag8, const uint8_t flags, const void *data, size_t len, const _rt_siphash_key_t key)

  Internal helper to hash data with a given 8-byte tag and flags.
- static l0_int _rt_hash_bool (l0_bool value, const uint8_t flags)

  Hash a boolean value with the runtime bool tag.
- static l0_int _rt_hash_byte (l0_byte value, const uint8_t flags)

  Hash a byte value with the runtime byte tag.
- static l0_int _rt_hash_int (l0_int value, const uint8_t flags)

  Hash an integer value with the runtime int tag.
- static l0_int _rt_hash_string (l0_string str, const uint8_t flags)

  Hash a string's byte contents with the runtime string tag.
- static l0_int _rt_hash_data (void *data, l0_int size, const uint8_t flags)

  Hash an arbitrary byte sequence with the runtime data tag.

- static l0_int rt_hash_bool (l0_bool value)
  Hash a boolean value.
- static l0_int rt_hash_byte (l0_byte value)
  Hash a byte value.
- static l0_int rt_hash_int (l0_int value)
  Hash an integer value.
- static l0_int rt_hash_string (l0_string value)
  Hash a string value.
- static l0_int rt_hash_data (void *data, l0_int size)
  Hash raw data.
- static l0_int rt_hash_opt_bool (l0_opt_bool opt)
  Hash an optional boolean value.
- static l0_int rt_hash_opt_byte (l0_opt_byte opt)
  Hash an optional byte value.
- static l0_int rt_hash_opt_int (l0_opt_int opt)
  Hash an optional integer value.
- static l0_int rt_hash_opt_string (l0_opt_string opt)
  Hash an optional string value.
- static l0_int rt_hash_ptr (void *ptr)
  Hash a pointer value.
- static l0_int rt_hash_opt_ptr (void *opt)
  Hash an optional pointer value.

Variables

- static const l0_int _RT_MEM_SENTINEL = 0xF00DB10C
  Sentinel value for memory checks.
- static l0_string L0_STRING_EMPTY = { 0, { .s_str = { 0, NULL } } }
  Static empty string instance.
- static l0_opt_string L0_OPT_STRING_NULL = { .has_value = 0, .value = { 0 } }
  Static instance for null optional string.
- static l0_opt_string L0_OPT_STRING_EMPTY = { .has_value = 1, .value = { 0 } }
  Static instance for empty optional string.
- static int _rt_argc = 0
- static char ** _rt_argv = NULL
- static _rt_alloc_record ** _rt_alloc_table = NULL
- static size_t _rt_alloc_table_cap = 0
- static size_t _rt_alloc_table_cnt = 0
- static size_t _rt_alloc_table_tombstones = 0
- static uint64_t _rt_alloc_next_generation = 1
- static _rt_alloc_record * _rt_alloc_tree_root = NULL
- static _rt_alloc_record * _rt_rec_free_list = NULL
- static _rt_alloc_record_cold ** _rt_cold_chunks = NULL
- static size_t _rt_rec_pool_chunks = 0
- static _rt_alloc_record * _rt_quarantine_head = NULL
- static _rt_alloc_record * _rt_quarantine_tail = NULL
- static size_t _rt_quarantine_bytes = 0
- static size_t _rt_quarantine_count = 0
- static const _rt_siphash_tag8_t _l0_sh_tag_bool = { 0, 'b', 'o', 'o', 'l' }
- static const _rt_siphash_tag8_t _l0_sh_tag_byte = { 0, 'i', 'n', 't', 8 }
- static const _rt_siphash_tag8_t _l0_sh_tag_int = { 0, 'i', 'n', 't', 32 }
- static const _rt_siphash_tag8_t _l0_sh_tag_string = { 0, 's', 't', 'r', 'i', 'n', 'g' }
- static const _rt_siphash_tag8_t _l0_sh_tag_data = { 0, 'd', 'a', 't', 'a' }
- static _rt_siphash_key_t _rt_sh_key
  Default (debug) SipHash key for L0 runtime.

8.22.1 Detailed Description

L0 Runtime Library (K0 - Kernel Layer)

Header-only C99 runtime providing:

- Memory allocation and deallocation
- Whole-file I/O operations
- Basic printing to stdout/stderr
- Panic mechanism for defined runtime aborts
- UB-free integer operations
- String type and operations
- Optional type support
- Random number generation
- Wall/monotonic time snapshots and local-time metadata
- Support for L0 `new` and `drop` semantics
- Environment variable access
- Reading from stdin
- Errno access

Design principles:

- All UB and platform quirks are confined to this file
- L0 programs use l0_int (int32_t); this layer handles size_t conversion
- Portable C99 code, no compiler-specific extensions

8.22.2 Macro Definition Documentation

8.22.2.1 _L0_TAG_ENUM

```
#define _L0_TAG_ENUM 0x20 /* enum */
```

8.22.2.2 _L0_TAG_OPT

```
#define _L0_TAG_OPT 0x80 /* option */
```

8.22.2.3 _L0_TAG_PTR

```
#define _L0_TAG_PTR 0x40 /* pointer */
```

8.22.2.4 _L0_TAG_STRUCT

```
#define _L0_TAG_STRUCT 0x10 /* struct */
```

8.22.2.5 _RT_ACCESS_READ

```
#define _RT_ACCESS_READ 0
```

8.22.2.6 _RT_ACCESS_UNTRACKED_OK

```
#define _RT_ACCESS_UNTRACKED_OK 2
```

8.22.2.7 _RT_ACCESS_WRITE

```
#define _RT_ACCESS_WRITE 1
```

8.22.2.8 _RT_ALIGNOF

```
#define _RT_ALIGNOF(
             type ) (sizeof(struct { char _rt_align_c; type _rt_align_v; }) - sizeof(type))
```

8.22.2.9 _RT_ALLOC_INIT_CAP

```
#define _RT_ALLOC_INIT_CAP 256
```

8.22.2.10 _RT_ALLOC_LIVE

```
#define _RT_ALLOC_LIVE 1
```

8.22.2.11 _RT_ALLOC_POOLED

```
#define _RT_ALLOC_POOLED 3
```

8.22.2.12 _RT_ALLOC_QUARANTINED

```
#define _RT_ALLOC_QUARANTINED 2
```

8.22.2.13 _RT_ALLOC_TOMBSTONE

```
#define _RT_ALLOC_TOMBSTONE ((_rt_alloc_record*)(uintptr_t)1)
```

8.22.2.14 _RT_MEM_ARC

```
#define _RT_MEM_ARC 2
```

8.22.2.15 _RT_MEM_FOREIGN

```
#define _RT_MEM_FOREIGN 4
```

8.22.2.16 _RT_MEM_NEW

```
#define _RT_MEM_NEW 1
```

8.22.2.17 _RT_MEM_RAW

```
#define _RT_MEM_RAW 0
```

8.22.2.18 _RT_MEM_STATIC

```
#define _RT_MEM_STATIC 3
```

8.22.2.19 _RT_QUARANTINE_MAX_BYTES

```
#define _RT_QUARANTINE_MAX_BYTES ((size_t)16 * 1024 * 1024)
```

8.22.2.20 _RT_QUARANTINE_MAX_COUNT

```
#define _RT_QUARANTINE_MAX_COUNT ((size_t)4096)
```

8.22.2.21 _rt_rec_cold

```
#define _rt_rec_cold(
              rec ) (&_rt_cold_chunks[(rec)→cold_index >> 8][(rec)→cold_index & 255u])
```

8.22.2.22 _RT_REC_POOL_CHUNK

```
#define _RT_REC_POOL_CHUNK 256
```

8.22.2.23 _RT_TRACE_ARC

```
#define _RT_TRACE_ARC(
              ... ) ((void)0)
```

8.22.2.24 _RT_TRACE_ARC_LOC

```
#define _RT_TRACE_ARC_LOC(
              loc_file,
              loc_line,
              ...  ) ((void)0)
```

8.22.2.25 _RT_TRACE_MEM

```
#define _RT_TRACE_MEM(
              ...  ) ((void)0)
```

8.22.2.26 _RT_TRACE_MEM_LOC

```
#define _RT_TRACE_MEM_LOC(
              loc_file,
              loc_line,
              ...  ) ((void)0)
```

8.22.2.27 DEA_RUNTIME_FCLOSE

```
#define DEA_RUNTIME_FCLOSE fclose
```

8.22.2.28 DEA_RUNTIME_FILE_INFO_STAT

```
#define DEA_RUNTIME_FILE_INFO_STAT stat
```

8.22.2.29 DEA_RUNTIME_REMOVE

```
#define DEA_RUNTIME_REMOVE remove
```

8.22.2.30 L0_DEFINED_l0_sys_rt_RtFileInfo

```
#define L0_DEFINED_l0_sys_rt_RtFileInfo
```

8.22.2.31 L0_DEFINED_l0_sys_rt_RtTimeParts

```
#define L0_DEFINED_l0_sys_rt_RtTimeParts
```

8.22.2.32 L0_OPT_BOOL_DEFINED

```
#define L0_OPT_BOOL_DEFINED
```

8.22.2.33 L0_OPT_BYTE_DEFINED

```
#define L0_OPT_BYTE_DEFINED
```

8.22.2.34 L0_OPT_INT_DEFINED

```
#define L0_OPT_INT_DEFINED
```

8.22.2.35 L0_OPT_STRING_DEFINED

```
#define L0_OPT_STRING_DEFINED
```

8.22.2.36 L0_STRING_CONST

```
#define L0_STRING_CONST(
              str_data,
              str_len ) { .kind = L0_STRING_K_STATIC, .data.s_str = { .len = (str_len), .bytes
= (str_data) } }
```

String literal construction macro.

8.22.2.37 L0_STRING_K_HEAP

```
#define L0_STRING_K_HEAP 1
```

8.22.2.38 L0_STRING_K_STATIC

```
#define L0_STRING_K_STATIC 0
```

8.22.2.39 L0_UNREACHABLE

```
#define L0_UNREACHABLE(
             _s ) rt_panic(_s)
```

8.22.3 Typedef Documentation

8.22.3.1 _rt_alloc_record

```
typedef struct _rt_alloc_record _rt_alloc_record
```

8.22.3.2 _rt_alloc_record_cold

```
typedef struct _rt_alloc_record_cold _rt_alloc_record_cold
```

8.22.3.3 _rt_alloc_record_size_check

```
typedef char _rt_alloc_record_size_check[(sizeof(_rt_alloc_record)==64) ?  1 :-1]
```

8.22.3.4 _rt_ptr_site

```
typedef struct _rt_ptr_site _rt_ptr_site
```

8.22.3.5 _rt_siphash_key_t

```
typedef uint8_t _rt_siphash_key_t[16]
```

8.22.3.6 _rt_siphash_tag8_t

```
typedef uint8_t _rt_siphash_tag8_t[8]
```

8.22.3.7 l0_bool

```
typedef uint8_t l0_bool
```

8.22.3.8 l0_byte

```
typedef uint8_t l0_byte
```

8.22.3.9 l0_double

```
typedef double l0_double
```

8.22.3.10 l0_float

```
typedef float l0_float
```

8.22.3.11 l0_int

```
typedef int32_t l0_int
```

8.22.3.12 l0_long

```
typedef int64_t l0_long
```

8.22.3.13 l0_short

```
typedef int16_t l0_short
```

8.22.3.14 l0_tiny

```
typedef int8_t l0_tiny
```

future use

8.22.3.15 l0_uint

```
typedef uint32_t l0_uint
```

8.22.3.16 l0_ulong

```
typedef uint64_t l0_ulong
```

8.22.3.17 l0_ushort

```
typedef uint16_t l0_ushort
```

8.22.4 Function Documentation

8.22.4.1 _rt_alloc_hash()

```
static size_t _rt_alloc_hash (
            void * ptr,
            size_t cap )  [inline], [static]
```

8.22.4.2 _rt_alloc_obj()

```
static void * _rt_alloc_obj (
            l0_int bytes )  [static]
```

8.22.4.3 _rt_alloc_string()

```
static l0_string _rt_alloc_string (
            l0_int len )  [static]
```

Allocate a new reference counted L0_string of the given length.

Character data (bytes[]) is uninitialized; caller must fill it in. Panics on allocation failure or negative length. Size of allocated memory is: string header + len + 1 for null terminator.

The returned string is of kind L0_STRING_K_HEAP and its data is null-terminated in advance.

Parameters

len	Length of the string.

Returns

Allocated L0 string.

8.22.4.4 _rt_alloc_table_insert()

```
static void _rt_alloc_table_insert (
            _rt_alloc_record * rec )  [static]
```

8.22.4.5 _rt_alloc_table_lookup()

```
static _rt_alloc_record * _rt_alloc_table_lookup (
            void * ptr )  [static]
```

8.22.4.6 _rt_alloc_table_rehash()

```
static void _rt_alloc_table_rehash (
            void  )  [static]
```

Rebuild the hash table sized from the live record count.

Sizing from the live count (not the old capacity) keeps sustained alloc/free churn from ratcheting the table up↩ : a tombstone-triggered rebuild purges tombstones at a stable or smaller capacity instead of doubling it, and a shrunken live set lets the table contract.

8.22.4.7 _rt_alloc_table_remove()

```
static void _rt_alloc_table_remove (
            _rt_alloc_record * target )  [static]
```

8.22.4.8 _rt_alloc_tree_find_containing()

```
static _rt_alloc_record * _rt_alloc_tree_find_containing (
            void * ptr,
            size_t need_size )  [static]
```

8.22.4.9 _rt_alloc_tree_glb()

```
static _rt_alloc_record * _rt_alloc_tree_glb (
            void * ptr )  [static]
```

Return the tracked record with the greatest base at or below `ptr`.

8.22.4.10 _rt_alloc_tree_insert()

```
static void _rt_alloc_tree_insert (
            _rt_alloc_record * rec )  [static]
```

8.22.4.11 _rt_alloc_tree_lub()

```
static _rt_alloc_record * _rt_alloc_tree_lub (
            void * ptr )  [static]
```

Return the tracked record with the lowest base at or above `ptr`.

8.22.4.12 _rt_alloc_tree_remove()

```
static void _rt_alloc_tree_remove (
            _rt_alloc_record * rec )  [static]
```

8.22.4.13 _rt_check_index_ptr_site()

```
static void * _rt_check_index_ptr_site (
            _rt_ptr_site * site,
            void * base_ptr,
            l0_int index,
            l0_int element_size,
            l0_int required_align,
            int access_mode,
            const char * loc_file,
            int loc_line )  [inline], [static]
```

Validate and derive one indexed pointer access through a call-site cache.

The checked path validates the base pointer and target byte range before forming the target pointer value, avoiding out-of-object C pointer arithmetic in generated code.

Parameters

site	Per-call-site cache slot owned by the generated access site.
base_ptr	Base pointer expression before indexing.
index	Element index.
element_size	Size in bytes of one indexed element.
required_align	Required target alignment, or 0 for no alignment check.
access_mode	`_RT_ACCESS_READ` or `_RT_ACCESS_WRITE`.
loc_file	Source file of the access site.
loc_line	Source line of the access site.

Returns

Target pointer when the access is valid; panics otherwise.

8.22.4.14 _rt_check_index_ptr_site_slow()

```
static void * _rt_check_index_ptr_site_slow (
            _rt_ptr_site * site,
            void * base_ptr,
            l0_int index,
            l0_int element_size,
            l0_int required_align,
            int access_mode,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.15 _rt_check_ptr_align()

```
static void _rt_check_ptr_align (
            void * ptr,
            size_t need_align,
            const char * loc_file,
            int loc_line )  [static]
```

Panic when a validated access target misses its required alignment.

Parameters

ptr	Access target to check.
need_align	Required alignment; values below 2 always pass.
loc_file	Source file of the access site.
loc_line	Source line of the access site.

8.22.4.16 _rt_check_ptr_site()

```
static void * _rt_check_ptr_site (
            _rt_ptr_site * site,
            void * ptr,
            l0_int required_size,
            l0_int required_align,
            int access_mode,
            const char * loc_file,
            int loc_line )  [inline], [static]
```

Validate one pointer access through a call-site cache.

Parameters

site	Per-call-site cache slot owned by the generated access site.
ptr	Pointer about to be dereferenced.
required_size	Size in bytes of the access.
required_align	Required alignment, or 0 for no alignment check.
access_mode	`_RT_ACCESS_READ` or `_RT_ACCESS_WRITE`.
loc_file	Source file of the access site.
loc_line	Source line of the access site.

Returns

`ptr` when the access is valid; panics otherwise.

8.22.4.17 _rt_check_ptr_site_slow()

```
static void * _rt_check_ptr_site_slow (
            _rt_ptr_site * site,
            void * ptr,
            l0_int required_size,
            l0_int required_align,
            int access_mode,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.18 _rt_check_record_writeable()

```
static void _rt_check_record_writeable (
            _rt_alloc_record * rec,
            void * ptr,
            int access_mode,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.19 _rt_drop_begin_impl()

```
static void * _rt_drop_begin_impl (
            void * ptr,
            l0_int required_size,
            l0_int required_align,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.20 _rt_drop_finish_impl()

```
static void _rt_drop_finish_impl (
            void * ptr,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.21 _rt_evict_quarantine()

```
static void _rt_evict_quarantine (
            void  )  [static]
```

8.22.4.22 _rt_fmix32()

```
static uint32_t _rt_fmix32 (
            uint32_t x )  [inline], [static]
```

Final mixing function for 32-bit hashes (MurmurHash3 fmix32).

Parameters

x	Current hash.

Returns

Mixed hash.

8.22.4.23 _rt_fold_u64_to_u32_fmix()

```
static uint32_t _rt_fold_u64_to_u32_fmix (
            uint64_t h )  [inline], [static]
```

Fold a 64-bit hash into a 32-bit hash with final mixing.

Parameters

h	64-bit hash.

Returns

32-bit hash.

8.22.4.24 _rt_free_string()

```
static void _rt_free_string (
            l0_string str )  [static]
```

Free a string's allocated data, if applicable.

If reference counted, decrements reference count and frees when it reaches zero.

Parameters

str	L0 string to free.

8.22.4.25 _rt_hash_bool()

```
static l0_int _rt_hash_bool (
            l0_bool value,
            const uint8_t flags )  [static]
```

Hash a boolean value with the runtime bool tag.

Parameters

value	Boolean value to hash.
flags	Type-shaping flags mixed into the hash domain.

Returns

32-bit hash.

8.22.4.26 _rt_hash_byte()

```
static l0_int _rt_hash_byte (
            l0_byte value,
            const uint8_t flags )  [static]
```

Hash a byte value with the runtime byte tag.

Parameters

value	Byte value to hash.
flags	Type-shaping flags mixed into the hash domain.

Returns

32-bit hash.

8.22.4.27 _rt_hash_data()

```
static l0_int _rt_hash_data (
            void * data,
            l0_int size,
            const uint8_t flags )  [static]
```

Hash an arbitrary byte sequence with the runtime data tag.

Parameters

data	Pointer to the byte sequence.
size	Size of `data` in bytes.
flags	Type-shaping flags mixed into the hash domain.

Returns

32-bit hash.

8.22.4.28 _rt_hash_int()

```
static l0_int _rt_hash_int (
            l0_int value,
            const uint8_t flags )  [static]
```

Hash an integer value with the runtime int tag.

Parameters

value	Integer value to hash.
flags	Type-shaping flags mixed into the hash domain.

Returns

32-bit hash.

8.22.4.29 _rt_hash_string()

```
static l0_int _rt_hash_string (
            l0_string str,
            const uint8_t flags )  [static]
```

Hash a string's byte contents with the runtime string tag.

Parameters

str	String value to hash.
flags	Type-shaping flags mixed into the hash domain.

Returns

32-bit hash.

8.22.4.30 _rt_hash_tag8()

```
static l0_int _rt_hash_tag8 (
            const _rt_siphash_tag8_t tag8,
            const uint8_t flags,
            const void * data,
            size_t len,
            const _rt_siphash_key_t key )  [static]
```

Internal helper to hash data with a given 8-byte tag and flags.

Parameters

tag8	8-byte tag.
flags	Flags.
data	Pointer to data.
len	Data length.
key	SipHash key.

Returns

32-bit hash.

8.22.4.31 _rt_iadd()

```
static l0_int _rt_iadd (
            l0_int a,
            l0_int b )  [static]
```

Safe integer addition with overflow check.

Parameters

a	First operand.
b	Second operand.

Returns

Sum.

8.22.4.32 _rt_idiv()

```
static l0_int _rt_idiv (
            l0_int a,
            l0_int b )  [static]
```

Safe integer division.

Parameters

a	Dividend.
b	Divisor.

Returns

Quotient.

8.22.4.33 _rt_imod()

```
static l0_int _rt_imod (
            l0_int a,
            l0_int b )  [static]
```

Safe integer modulo.

Parameters

a	Dividend.
b	Divisor.

Returns

Remainder.

8.22.4.34 _rt_imul()

```
static l0_int _rt_imul (
            l0_int a,
            l0_int b )  [static]
```

Safe integer multiplication with overflow check.

Parameters

a	First operand.
b	Second operand.

Returns

Product.

8.22.4.35 _rt_index_access_mode()

```
static int _rt_index_access_mode (
            int access_mode )  [static]
```

8.22.4.36 _rt_index_allows_untracked()

```
static int _rt_index_allows_untracked (
            int access_mode )  [static]
```

8.22.4.37 _rt_index_element_size()

```
static size_t _rt_index_element_size (
            l0_int element_size )  [static]
```

8.22.4.38 _rt_index_offset()

```
static size_t _rt_index_offset (
            l0_int index,
            size_t element_size )  [static]
```

8.22.4.39 _rt_init_args()

```
void _rt_init_args (
            int argc,
            char ** argv )
```

Initialize command-line arguments.

Parameters

argc	Number of arguments.
argv	Argument vector.

8.22.4.40 _rt_init_heap_string()

```
static l0_string _rt_init_heap_string (
            void * mem,
            l0_int s_len )  [static]
```

Initialize a heap-allocated L0_string in the given memory.

Character data (bytes[]) is uninitialized; caller must fill it in. Length is assumed to be already validated by the caller. Size of mem MUST be at least sizeof(_l0_h_string) + s_len + 1.

The returned string is of kind L0_STRING_K_HEAP and its data is null-terminated in advance.

Parameters

mem	Allocated memory block.
s_len	Length of the string.

Returns

Initialized L0 string.

8.22.4.41 _rt_isub()

```
static l0_int _rt_isub (
            l0_int a,
            l0_int b )  [static]
```

Safe integer subtraction with overflow check.

Parameters

a	First operand.
b	Second operand.

Returns

Difference.

8.22.4.42 _rt_l0_string_from_const_literal()

```
static l0_string _rt_l0_string_from_const_literal (
            const char * c_str )  [static]
```

Create an L0 string from a constant C string.

Returns a string with len=0 if c_str is NULL.

Note: Does NOT allocate or copy - just wraps the existing C string. Use only for string literals or static const data.

Parameters

c_str	Constant C string.

Returns

L0 string.

8.22.4.43 _rt_narrow_l0_byte()

```
l0_byte _rt_narrow_l0_byte (
            l0_int value )
```

Narrow l0_int to l0_byte with range check.

Parameters

value	Integer value.

Returns

Byte value.

8.22.4.44 _rt_new_l0_string()

```
static l0_string _rt_new_l0_string (
            const char * c_str )  [static]
```

Create a new reference counted L0_string from a null-terminated C string.

Allocates new memory and copies data.

Parameters

c_str	Null-terminated C string.

Returns

L0 string.

8.22.4.45 _rt_panic()

```
static void _rt_panic (
            const char * message )  [static]
```

Abort the program with a message.

Parameters

message	The panic message.

8.22.4.46 _rt_panic_fmt()

```
static void _rt_panic_fmt (
            const char * fmt,
             ...  )  [static]
```

Abort the program with a formatted message.

Parameters

fmt	Format string.

8.22.4.47 _rt_panic_invalid_access()

```
static void _rt_panic_invalid_access (
            const char * reason,
            void * ptr,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.48 _rt_panic_invalid_drop()

```
static void _rt_panic_invalid_drop (
            const char * reason,
            void * ptr,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.49 _rt_panic_invalid_release()

```
static void _rt_panic_invalid_release (
            const char * op_name,
            const char * reason,
            void * ptr,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.50 _rt_pid_to_l0_int()

```
static l0_bool _rt_pid_to_l0_int (
            intmax_t value,
            l0_int * out )  [static]
```

Convert a native process identifier into `l0_int`.

Parameters

value	Native process identifier.
out	Output location.

Returns

1 when `value` fits in `l0_int`, otherwise 0.

8.22.4.51 _rt_print()

```
void _rt_print (
            l0_string s,
            FILE * stream )
```

Internal helper to print an l0_string to a given stream.

Parameters

s	String to print.
stream	Target stream.

8.22.4.52 _rt_promote_new_alloc()

```
static void _rt_promote_new_alloc (
            void * ptr )  [static]
```

8.22.4.53 _rt_ptr_is_aligned()

```
static int _rt_ptr_is_aligned (
            void * ptr,
            size_t align )  [static]
```

8.22.4.54 _rt_ptr_site_store()

```
static void _rt_ptr_site_store (
            _rt_ptr_site * site,
            _rt_alloc_record * owner )  [static]
```

8.22.4.55 _rt_quarantine_alloc_record()

```
static void _rt_quarantine_alloc_record (
            _rt_alloc_record * rec,
            _rt_alloc_record_cold * cold,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.56 _rt_range_contains()

```
static int _rt_range_contains (
            void * base,
            size_t total_size,
            void * ptr,
            size_t need_size,
            size_t * offset_out )  [static]
```

8.22.4.57 _rt_realloc_string()

```
static l0_string _rt_realloc_string (
            l0_string s,
            l0_int new_len )  [static]
```

Reallocate a heap string to a new length.

Parameters

s	Current L0 string.
new_len	New length.

Returns

Updated L0 string.

8.22.4.58 _rt_realloc_tracked_alloc()

```
static void * _rt_realloc_tracked_alloc (
            void * ptr,
            size_t new_size,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.59 _rt_rec_new()

```
static _rt_alloc_record * _rt_rec_new (
            _rt_alloc_record_cold ** cold_out )  [static]
```

Take one record from the pool, refilling it chunk-wise.

Pool memory is never returned to the C allocator, so stale record pointers held by call-site caches remain safe to dereference.

8.22.4.60 _rt_rec_recycle()

```
static void _rt_rec_recycle (
            _rt_alloc_record * rec )  [static]
```

8.22.4.61 _rt_release_tracked_alloc()

```
static void _rt_release_tracked_alloc (
            void * ptr,
            const char * loc_file,
            int loc_line,
            const char * op_name )  [static]
```

8.22.4.62 _rt_required_access_mode()

```
static int _rt_required_access_mode (
            int access_mode )  [static]
```

8.22.4.63 _rt_required_align()

```
static size_t _rt_required_align (
            l0_int required_align )  [static]
```

8.22.4.64 _rt_required_size()

```
static size_t _rt_required_size (
            l0_int required_size )  [static]
```

8.22.4.65 _rt_stream_write_some()

```
static l0_int _rt_stream_write_some (
            FILE * stream,
            const l0_byte * buf,
            l0_int len )  [static]
```

Write raw bytes to one standard stream.

Parameters

stream	Target stream.
buf	Source bytes.
len	Maximum number of bytes to write.

Returns

Bytes written, or `-1` on error.

8.22.4.66 _rt_string_bytes()

```
static char * _rt_string_bytes (
            l0_string s )  [static]
```

Gets the null-terminated C string underlying an L0 string.

or NULL if not available, e.g. for static empty strings. Useful when interfacing with C APIs that require null-terminated strings.

Note: This is an internal helper, not exposed to L0 code.

Parameters

s	L0 string.

Returns

Pointer to character data.

8.22.4.67 _rt_time_to_l0_int_nsec()

```
static l0_bool _rt_time_to_l0_int_nsec (
            long value,
            l0_int * out )  [static]
```

Internal helper to convert long to l0_int nanoseconds.

8.22.4.68 _rt_time_to_l0_int_sec()

```
static l0_bool _rt_time_to_l0_int_sec (
            time_t value,
            l0_int * out )  [static]
```

Internal helper to convert time_t to l0_int seconds.

8.22.4.69 _rt_time_write_parts()

```
static l0_bool _rt_time_write_parts (
            struct l0_sys_rt_RtTimeParts * out,
            l0_int sec,
            l0_int nsec )  [static]
```

Internal helper to write time parts to struct.

8.22.4.70 _rt_trace_invalid_drop()

```
static void _rt_trace_invalid_drop (
            void * ptr,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.71 _rt_track_alloc_record()

```
static void _rt_track_alloc_record (
            void * ptr,
            size_t size,
            size_t align,
            uint32_t type_id,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.72 _rt_track_alloc_record_kind()

```
static void _rt_track_alloc_record_kind (
            void * ptr,
            size_t size,
            size_t align,
            uint32_t type_id,
            int mem_kind,
            int read_only,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.73 _rt_track_arc_bytes()

```
static void _rt_track_arc_bytes (
            void * ptr,
            size_t size )  [static]
```

Lazily register ARC-managed string storage as a read-only tracked record.

Idempotent: repeated exposure of the same live block is a no-op, so `rt_string_bytes_ptr` may call this on every invocation.

8.22.4.74 _rt_track_static_bytes()

```
static void _rt_track_static_bytes (
            const void * ptr,
            size_t size )  [static]
```

8.22.4.75 _rt_tree_insert_at()

```
static _rt_alloc_record * _rt_tree_insert_at (
            _rt_alloc_record * node,
            _rt_alloc_record * rec )  [static]
```

8.22.4.76 _rt_tree_merge()

```
static _rt_alloc_record * _rt_tree_merge (
            _rt_alloc_record * a,
            _rt_alloc_record * b )  [static]
```

Merge two treaps where every base in `a` is below every base in `b`.

8.22.4.77 _rt_tree_prio_for()

```
static uint32_t _rt_tree_prio_for (
            void * base )  [inline], [static]
```

8.22.4.78 _rt_tree_remove_at()

```
static _rt_alloc_record * _rt_tree_remove_at (
            _rt_alloc_record * node,
            void * base )  [static]
```

8.22.4.79 _rt_untrack_arc_alloc()

```
static void _rt_untrack_arc_alloc (
            void * ptr )  [static]
```

8.22.4.80 _rt_validate_derived_ptr()

```
static void * _rt_validate_derived_ptr (
            void * derived,
            void * parent_base,
            l0_int size,
            l0_int align,
            const char * loc_file,
            int loc_line )  [static]
```

8.22.4.81 _unwrap_opt()

```
static void * _unwrap_opt (
            void * opt_ptr,
            const char * type_name )  [inline], [static]
```

Unwrap an optional type structure, panicking if it has no value.

Parameters

opt_ptr	Pointer to the optional structure.
type_name	Name of the type for error reporting.

Returns

Pointer to the optional structure.

8.22.4.82 _unwrap_ptr()

```
static void * _unwrap_ptr (
            void * opt,
            const char * type_name )  [inline], [static]
```

Unwrap a pointer, panicking if NULL.

Parameters

opt	Pointer to unwrap.
type_name	Name of the type for error reporting.

Returns

Unwrapped pointer.

8.22.4.83 rt_abort()

```
static void rt_abort (
            l0_string message )  [static]
```

Abort the program with a panic message.

Parameters

message	Panic message.

L0 signature: `extern func rt_abort(message: string) → void;`

8.22.4.84 rt_alloc()

```
static void * rt_alloc (
            l0_int bytes )  [static]
```

Allocate memory of the given size in bytes.

Returns NULL on allocation failure or if bytes is zero. Panics if bytes is negative, or too large to fit in size_t (platform-dependent).

Parameters

bytes	Size in bytes.

Returns

Pointer to allocated memory or NULL.

L0 signature: `extern func rt_alloc(bytes: int) → void*?;`

8.22.4.85 rt_array_element()

```
static void * rt_array_element (
            void * array_data,
            l0_int element_size,
            l0_int index ) [static]
```

Get a pointer to an element in an array.

Panics if array_data is NULL, element_size is non-positive, or index is negative.

Parameters

array_data	Pointer to array data.
element_size	Size of one element.
index	Element index.

Returns

Pointer to the element.

L0 signature: `extern func rt_array_element(array_data: void*, element_size: int, index: int) → void*;`

8.22.4.86 rt_calloc()

```
static void * rt_calloc (
            l0_int count,
            l0_int elem_size ) [static]
```

Allocate zeroed memory for an array of elements.

Returns NULL on allocation failure or if count/elem_size is negative.

Parameters

count	Number of elements.
elem_size	Element size.

Returns

Pointer to zeroed memory or NULL.

L0 signature: `extern func rt_calloc(count: int, elem_size: int) → void*?;`

8.22.4.87 rt_delete_file()

```
static l0_bool rt_delete_file (
            l0_string path )  [static]
```

Delete the file at the given path.

Returns 1 (true) on success, 0 (false) on error.

Parameters

path	File path.

Returns

1 on success, 0 on failure.

L0 signature: `extern func rt_delete_file(path: string) → bool;`

8.22.4.88 rt_errno()

```
static l0_int rt_errno (
            void )  [static]
```

Get the current errno value.

Returns

errno value.

L0 signature: `extern func rt_errno() → int;`

8.22.4.89 rt_exit()

```
static void rt_exit (
            l0_int code )  [static]
```

Exit the program with the given exit code.

Parameters

code	Exit code.

L0 signature: `extern func rt_exit(code: int) → void;`

8.22.4.90 rt_file_info()

```
static struct l0_sys_rt_RtFileInfo rt_file_info (
            l0_string path )  [static]
```

Return basic metadata for a path.

Parameters

path	File path.

Returns

Metadata record with nullable size and mtime fields.

L0 signature: `extern func rt_file_info(path: string) → RtFileInfo;`

8.22.4.91 rt_flush_stderr()

```
static void rt_flush_stderr (
            void )  [static]
```

Flush stderr.

L0 signature: `extern func rt_flush_stdout() → void;`

L0 signature: `extern func rt_flush_stderr() → void;`

8.22.4.92 rt_flush_stdout()

```
static void rt_flush_stdout (
            void )  [static]
```

Flush stdout.

8.22.4.93 rt_free()

```
static void rt_free (
            void * ptr )  [static]
```

Free a raw runtime allocation.

Parameters

ptr	Pointer to free.

L0 signature: `extern func rt_free(ptr: void*?) → void;`

8.22.4.94 rt_get_argc()

```
static l0_int rt_get_argc (
            void ) [static]
```

Get the number of command-line arguments.

Returns

Argument count.

L0 signature: `extern func rt_get_argc() → int;`

8.22.4.95 rt_get_argv()

```
static l0_string rt_get_argv (
            l0_int i ) [static]
```

Get the command-line argument at the given index.

Panics if index is out of bounds.

Parameters

i	Index.

Returns

Argument string.

L0 signature: `extern func rt_get_argv(i: int) → string;`

8.22.4.96 rt_get_env_var()

```
static l0_opt_string rt_get_env_var (
            l0_string name ) [static]
```

Get an environment variable as an L0 optional string.

Returns null (empty optional) if the variable is not set.

Parameters

name	Variable name.

Returns

Optional string value.

L0 signature: `extern func rt_get_env_var(name: string) → string?;`

8.22.4.97 rt_get_pid()

```
static l0_int rt_get_pid (
            void ) [static]
```

Get the current process identifier.

Returns

Process identifier.

L0 signature: `extern func rt_get_pid() → int;`

8.22.4.98 rt_hash_bool()

```
static l0_int rt_hash_bool (
            l0_bool value ) [static]
```

Hash a boolean value.

Parameters

value	Boolean value.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_bool(value: bool) → int;`

8.22.4.99 rt_hash_byte()

```
static l0_int rt_hash_byte (
            l0_byte value ) [static]
```

Hash a byte value.

Parameters

value	Byte value.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_byte(value: byte) → int;`

8.22.4.100 rt_hash_data()

```
static l0_int rt_hash_data (
            void * data,
            l0_int size )  [static]
```

Hash raw data.

Panics if data is null or size is negative.

Parameters

data	Pointer to data.
size	Data size.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_data(data: void*, size: int) → int;`

8.22.4.101 rt_hash_int()

```
static l0_int rt_hash_int (
            l0_int value )  [static]
```

Hash an integer value.

Parameters

value	Integer value.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_int(value: int) → int;`

8.22.4.102 rt_hash_opt_bool()

```
static l0_int rt_hash_opt_bool (
            l0_opt_bool opt )  [static]
```

Hash an optional boolean value.

Parameters

opt	Optional bool.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_opt_bool(opt: bool?) → int;`

8.22.4.103 rt_hash_opt_byte()

```
static l0_int rt_hash_opt_byte (
            l0_opt_byte opt ) [static]
```

Hash an optional byte value.

Parameters

opt	Optional byte.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_opt_byte(opt: byte?) → int;`

8.22.4.104 rt_hash_opt_int()

```
static l0_int rt_hash_opt_int (
            l0_opt_int opt ) [static]
```

Hash an optional integer value.

Parameters

opt	Optional int.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_opt_int(opt: int?) → int;`

8.22.4.105 rt_hash_opt_ptr()

```
static l0_int rt_hash_opt_ptr (
            void * opt ) [static]
```

Hash an optional pointer value.

Note: this hashes the pointer value (address), not the data it points to. Panics if opt is empty (null pointer).

Parameters

opt	Pointer.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_opt_ptr(opt: void*?) → int;`

8.22.4.106 rt_hash_opt_string()

```
static l0_int rt_hash_opt_string (
            l0_opt_string opt ) [static]
```

Hash an optional string value.

If opt is empty, hashes as an empty string with the optional flag.

Parameters

opt	Optional string.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_opt_string(opt: string?) → int;`

8.22.4.107 rt_hash_ptr()

```
static l0_int rt_hash_ptr (
            void * ptr ) [static]
```

Hash a pointer value.

Note: this hashes the pointer value (address), not the data it points to. Panics if ptr is null.

Parameters

ptr	Pointer.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_ptr(ptr: void*) → int;`

8.22.4.108 rt_hash_string()

```
static l0_int rt_hash_string (
            l0_string value )  [static]
```

Hash a string value.

Parameters

value	L0 string.

Returns

32-bit hash.

L0 signature: `extern func rt_hash_string(value: string) → int;`

8.22.4.109 rt_memcmp()

```
static l0_int rt_memcmp (
            void * a,
            void * b,
            l0_int bytes )  [static]
```

Compare two memory regions.

Returns 0 if equal, <0 if a < b, >0 if a > b.

Parameters

a	First region.
b	Second region.
bytes	Number of bytes.

Returns

Comparison result.

L0 signature: `extern func rt_memcmp(a: void*, b: void*, bytes: int) → int;`

8.22.4.110 rt_memcpy()

```
static void * rt_memcpy (
            void * dest,
```

```
            void * src,
            l0_int bytes )  [static]
```

Copy memory from source to destination.

Returns destination pointer.

Parameters

dest	Destination.
src	Source.
bytes	Number of bytes.

Returns

dest.

L0 signature: `extern func rt_memcpy(dest: void*, src: void*, bytes: int) → void*;`

8.22.4.111 rt_memset()

```
static void * rt_memset (
            void * dest,
            l0_int value,
            l0_int bytes )  [static]
```

Set memory to a specific byte value.

Returns destination pointer.

Parameters

dest	Destination pointer.
value	Byte value.
bytes	Number of bytes.

Returns

dest.

L0 signature: `extern func rt_memset(dest: void*, value: int, bytes: int) → void*;`

8.22.4.112 rt_print()

```
static void rt_print (
            l0_string s )  [static]
```

Print a string to stdout.

Parameters

s	String to print.

L0 signature: `extern func rt_print(s: string) → void;`

8.22.4.113 rt_print_bool()

```
static void rt_print_bool (
            l0_bool x )  [static]
```

Print a bool to stdout.

Parameters

x	Boolean value.

L0 signature: `extern func rt_print_bool(x: bool) → void;`

8.22.4.114 rt_print_bool_stderr()

```
static void rt_print_bool_stderr (
            l0_bool x )  [static]
```

Print a bool to stderr.

Parameters

x	Boolean value.

L0 signature: `extern func rt_print_bool_stderr(x: bool) → void;`

8.22.4.115 rt_print_int()

```
static void rt_print_int (
            l0_int x )  [static]
```

Print an integer to stdout.

Parameters

x	Integer value.

L0 signature: `extern func rt_print_int(x: int) → void;`

8.22.4.116 rt_print_int_stderr()

```
static void rt_print_int_stderr (
            l0_int x )  [static]
```

Print an integer to stderr.

Parameters

x	Integer value.

L0 signature: `extern func rt_print_int_stderr(x: int) → void;`

8.22.4.117 rt_print_stderr()

```
static void rt_print_stderr (
            l0_string s )  [static]
```

Print a string to stderr.

Parameters

s	String to print.

L0 signature: `extern func rt_print_stderr(s: string) → void;`

8.22.4.118 rt_println()

```
static void rt_println (
            void  )  [static]
```

Print a newline to stdout.

8.22.4.119 rt_println_stderr()

```
static void rt_println_stderr (
            void  )  [static]
```

Print a newline to stderr.

L0 signature: `extern func rt_println() → void;`

L0 signature: `extern func rt_println_stderr() → void;`

8.22.4.120 rt_rand()

```
static l0_int rt_rand (
            l0_int max )  [static]
```

Generate a random integer in the range [0, max).

Returns 0 if max <= 0.

Parameters

max	Upper bound (exclusive).

Returns

Random value.

L0 signature: `extern func rt_rand(max: int) → int;`

8.22.4.121 rt_read_char()

```
static l0_int rt_read_char (
            void ) [static]
```

Read one character from stdin.

Returns -1 on EOF or error.

Returns

Character value or -1.

L0 signature: `extern func rt_read_char() → int;`

8.22.4.122 rt_read_file_all()

```
static l0_opt_string rt_read_file_all (
            l0_string path ) [static]
```

Read entire file contents into a string.

Returns empty string on error (file not found, read error, allocation failure).

Parameters

path	File path.

Returns

Optional string containing file contents.

L0 signature: `extern func rt_read_file_all(path: string) → string?;`

8.22.4.123 rt_read_line()

```
static l0_opt_string rt_read_line (
            void ) [static]
```

Read a line from stdin into a dynamically allocated buffer.

Returns None on EOF (no characters read).

- Ownership: on Some(s), s.data is heap-allocated and must be freed by calling rt_string_release(s) (directly or indirectly via stdlib).

Returns

Optional string containing the line.

L0 signature: `extern func rt_read_line() → string?;`

8.22.4.124 rt_realloc()

```
static void * rt_realloc (
            void * ptr,
            l0_int new_bytes )  [static]
```

Reallocate a raw runtime allocation to a new size.

Returns NULL on failure. Panics if new_bytes is negative or too large to fit in size_t (platform-dependent). If ptr is NULL, behaves like rt_alloc.

Parameters

ptr	Pointer to memory, or NULL to allocate fresh.
new_bytes	New size.

Returns

Pointer to reallocated memory or NULL.

L0 signature: `extern func rt_realloc(ptr: void*?, new_bytes: int) → void*?;`

8.22.4.125 rt_register_foreign()

```
static void rt_register_foreign (
            void * ptr,
            l0_int bytes,
            l0_bool read_only )  [static]
```

Register externally owned storage for checked generated pointer accesses.

8.22.4.126 rt_srand()

```
static void rt_srand (
            l0_int seed )  [static]
```

Seed the random number generator.

Uses current time if seed is 0.

Parameters

seed	Seed value.

L0 signature: `extern func rt_srand(seed: int) → void;`

8.22.4.127 rt_stderr_write()

```
static l0_int rt_stderr_write (
            l0_byte * buf,
            l0_int len ) [static]
```

Write raw bytes to standard error.

Parameters

buf	Source bytes. A NULL buffer is reported as `-1`.
len	Maximum number of bytes to write.

Returns

Bytes written, or `-1` on error (including a NULL `buf` when `len > 0`).

L0 signature: `extern func rt_stderr_write(buf: byte*?, len: int) → int;`

8.22.4.128 rt_stdin_read()

```
static l0_int rt_stdin_read (
            l0_byte * buf,
            l0_int capacity ) [static]
```

Read raw bytes from standard input.

Parameters

buf	Destination bytes. A NULL buffer is reported as `-1`.
capacity	Maximum number of bytes to read.

Returns

Bytes read, `0` on EOF, or `-1` on error (including a NULL `buf` when `capacity > 0`).

L0 signature: `extern func rt_stdin_read(buf: byte*?, capacity: int) → int;`

8.22.4.129 rt_stdout_write()

```
static l0_int rt_stdout_write (
            l0_byte * buf,
            l0_int len ) [static]
```

Write raw bytes to standard output.

Parameters

buf	Source bytes. A NULL buffer is reported as `-1`.
len	Maximum number of bytes to write.

Returns

Bytes written, or `-1` on error (including a NULL `buf` when `len > 0`).

L0 signature: `extern func rt_stdout_write(buf: byte*?, len: int) → int;`

8.22.4.130 rt_string_bytes_ptr()

```
static l0_byte * rt_string_bytes_ptr (
            l0_string s )  [static]
```

Return a pointer to the raw byte data of a string.

Parameters

s	L0 string.

Returns

Pointer to the first byte.

L0 signature: `extern func rt_string_bytes_ptr(s: string) → byte*;`

Heap and static string storage is registered with the pointer access tracker lazily here, at first raw-byte exposure, so the returned pointer stays dereferenceable by checked generated code while strings that never hand out raw bytes stay out of the tracker. Both are runtime-managed: passing the returned pointer to `drop` or `rt_free` is a runtime error.

8.22.4.131 rt_string_compare()

```
static l0_int rt_string_compare (
            l0_string a,
            l0_string b )  [static]
```

Compare two strings lexicographically.

Returns 0 if equal, <0 if a < b, >0 if a > b.

Parameters

a	First string.
b	Second string.

Returns

Comparison result.

L0 signature: `extern func rt_string_compare(a: string, b: string) → int;`

8.22.4.132 rt_string_concat()

```
static l0_string rt_string_concat (
            l0_string a,
            l0_string b )  [static]
```

Concatenate two strings (allocates new memory).

Returns a heap-allocated string containing a + b.

Parameters

a	First string.
b	Second string.

Returns

Concatenated string.

L0 signature: `extern func rt_string_concat(a: string, b: string) → string;`

8.22.4.133 rt_string_equals()

```
static l0_bool rt_string_equals (
            l0_string a,
            l0_string b )  [static]
```

Check if two strings are equal.

Parameters

a	First string.
b	Second string.

Returns

1 if equal, 0 otherwise.

L0 signature: `extern func rt_string_equals(a: string, b: string) → bool;`

8.22.4.134 rt_string_from_byte()

```
static l0_string rt_string_from_byte (
            l0_byte b )  [static]
```

Create an L0 string from a single character (byte).

Allocates a new heap string of length 1. Note: Caller must free the returned string using _rt_free_string.

Parameters

b	Character.

Returns

L0 string.

L0 signature: `extern func rt_string_from_byte(b: byte) → string;`

8.22.4.135 rt_string_from_byte_array()

```
static l0_string rt_string_from_byte_array (
            l0_byte * bytes,
            l0_int len )  [static]
```

Create an L0 string from a byte array and a length.

Allocates a new heap string of the given length and copies data. The array does not need to be a null-terminated C string: all bytes are copied and a null terminator is added for C interoperability. Panics if len is negative.

Parameters

bytes	Pointer to bytes.
len	Length.

Returns

L0 string.

L0 signature: `extern func rt_string_from_byte_array(bytes: byte*, len: int) → string;`

8.22.4.136 rt_string_get()

```
static l0_byte rt_string_get (
            l0_string a,
            l0_int index )  [static]
```

Bounds-checked character access.

Returns the character at the given index, or panics if out of bounds.

Parameters

a	L0 string.
index	Index.

Returns

Byte value.

L0 signature: `extern func rt_string_get(s: string, index: int) → byte;`

8.22.4.137 rt_string_release()

```
static void rt_string_release (
            l0_string s )  [static]
```

Decrement reference count, freeing if zero.

Parameters

s	L0 string.

L0 signature: `extern func rt_string_release(s: string) → void;`

8.22.4.138 rt_string_retain()

```
static void rt_string_retain (
            l0_string s )  [static]
```

Increment reference count for heap strings (no-op for static).

Panics if the string is heap-allocated but has an invalid refcount state (e.g. double free detected).

Parameters

s	L0 string.

L0 signature: `extern func rt_string_retain(s: string) → void;`

8.22.4.139 rt_string_slice()

```
static l0_string rt_string_slice (
            l0_string s,
            l0_int start,
            l0_int end )  [static]
```

Create a substring (allocates new memory).

Panics if start/end are out of bounds or start > end.

Parameters

s	Source string.
start	Start index.
end	End index.

Returns

Slice string.

L0 signature: `extern func rt_string_slice(s: string, start: int, end: int) → string;`

8.22.4.140 rt_strlen()

```
static l0_int rt_strlen (
            l0_string str )  [static]
```

Get the length of a string.

Parameters

str	L0 string.

Returns

Length in bytes.

L0 signature: `extern func rt_strlen(str: string) → int;`

8.22.4.141 rt_system()

```
static l0_int rt_system (
            l0_string cmd )  [static]
```

Execute a system command and return its normalized status.

Returns the command exit code, `128 + signal` when terminated by a signal, or a negative value on error launching the shell.

Parameters

cmd	Command string.

Returns

Normalized status.

L0 signature: `extern func rt_system(cmd: string) → int;`

8.22.4.142 rt_time_local_is_dst()

```
static l0_opt_bool rt_time_local_is_dst (
            l0_int unix_sec )  [static]
```

Returns whether local time is daylight-saving time for `unix_sec`.

Parameters

unix_sec	Unix timestamp.

Returns

Optional boolean.

L0 signature: `extern func rt_time_local_is_dst(unix_sec: int) → bool?;`

8.22.4.143 rt_time_local_offset_sec()

```
static l0_opt_int rt_time_local_offset_sec (
            l0_int unix_sec ) [static]
```

Returns local UTC offset in seconds for `unix_sec`.

Computes the offset by comparing `gmtime` and `localtime` breakdowns directly, avoiding `mktime` which rejects pre-epoch values on some platforms.

Parameters

unix_sec	Unix timestamp.

Returns

Optional integer offset.

L0 signature: `extern func rt_time_local_offset_sec(unix_sec: int) → int?;`

8.22.4.144 rt_time_monotonic()

```
static l0_bool rt_time_monotonic (
            struct l0_sys_rt_RtTimeParts * out ) [static]
```

Capture current monotonic clock into `out`.

Parameters

out	Pointer to RtTimeParts.

Returns

1 on success, 0 on failure.

L0 signature: `extern func rt_time_monotonic(out: RtTimeParts*) → bool;`

8.22.4.145 rt_time_monotonic_supported()

```
static l0_bool rt_time_monotonic_supported (
            void  )  [static]
```

Returns whether a monotonic clock source is available.

Returns

1 if supported, 0 otherwise.

L0 signature: `extern func rt_time_monotonic_supported() → bool;`

8.22.4.146 rt_time_unix()

```
static l0_bool rt_time_unix (
            struct l0_sys_rt_RtTimeParts * out )  [static]
```

Capture current unix wall clock into `out`.

Parameters

out	Pointer to RtTimeParts.

Returns

1 on success, 0 on failure.

L0 signature: `extern func rt_time_unix(out:  RtTimeParts*) → bool;`

8.22.4.147 rt_unregister_foreign()

```
static void rt_unregister_foreign (
            void * ptr )  [static]
```

Unregister externally owned storage without releasing its payload.

8.22.4.148 rt_write_file_all()

```
static l0_bool rt_write_file_all (
            l0_string path,
            l0_string data )  [static]
```

Write string data to a file.

Returns 1 (true) on success, 0 (false) on error.

Parameters

path	File path.
data	Data string.

Returns

1 on success, 0 on failure.

L0 signature: `extern func rt_write_file_all(path: string, data: string) → bool;`

8.22.5 Variable Documentation

8.22.5.1 _l0_sh_tag_bool

```
const _rt_siphash_tag8_t _l0_sh_tag_bool = { 0, 'b', 'o', 'o', 'l' }  [static]
```

8.22.5.2 _l0_sh_tag_byte

```
const _rt_siphash_tag8_t _l0_sh_tag_byte = { 0, 'i', 'n', 't', 8 }  [static]
```

8.22.5.3 _l0_sh_tag_data

```
const _rt_siphash_tag8_t _l0_sh_tag_data = { 0, 'd', 'a', 't', 'a' }  [static]
```

8.22.5.4 _l0_sh_tag_int

```
const _rt_siphash_tag8_t _l0_sh_tag_int = { 0, 'i', 'n', 't', 32 }  [static]
```

8.22.5.5 _l0_sh_tag_string

```
const _rt_siphash_tag8_t _l0_sh_tag_string = { 0, 's', 't', 'r', 'i', 'n', 'g' }  [static]
```

8.22.5.6 _rt_alloc_next_generation

```
uint64_t _rt_alloc_next_generation = 1  [static]
```

8.22.5.7 _rt_alloc_table

```
_rt_alloc_record** _rt_alloc_table = NULL  [static]
```

8.22.5.8 _rt_alloc_table_cap

```
size_t _rt_alloc_table_cap = 0  [static]
```

8.22.5.9 _rt_alloc_table_cnt

```
size_t _rt_alloc_table_cnt = 0  [static]
```

8.22.5.10 _rt_alloc_table_tombstones

```
size_t _rt_alloc_table_tombstones = 0  [static]
```

8.22.5.11 _rt_alloc_tree_root

```
_rt_alloc_record* _rt_alloc_tree_root = NULL  [static]
```

8.22.5.12 _rt_argc

```
int _rt_argc = 0  [static]
```

8.22.5.13 _rt_argv

```
char** _rt_argv = NULL  [static]
```

8.22.5.14 _rt_cold_chunks

```
_rt_alloc_record_cold** _rt_cold_chunks = NULL  [static]
```

8.22.5.15 _RT_MEM_SENTINEL

```
const l0_int _RT_MEM_SENTINEL = 0xF00DB10C  [static]
```

Sentinel value for memory checks.

8.22.5.16 _rt_quarantine_bytes

```
size_t _rt_quarantine_bytes = 0  [static]
```

8.22.5.17 _rt_quarantine_count

```
size_t _rt_quarantine_count = 0  [static]
```

8.22.5.18 _rt_quarantine_head

```
_rt_alloc_record* _rt_quarantine_head = NULL  [static]
```

8.22.5.19 _rt_quarantine_tail

```
_rt_alloc_record* _rt_quarantine_tail = NULL  [static]
```

8.22.5.20 _rt_rec_free_list

```
_rt_alloc_record* _rt_rec_free_list = NULL  [static]
```

8.22.5.21 _rt_rec_pool_chunks

```
size_t _rt_rec_pool_chunks = 0  [static]
```

8.22.5.22 _rt_sh_key

```
_rt_siphash_key_t _rt_sh_key  [static]
```

Initial value:

```
= {
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
}
```

Default (debug) SipHash key for L0 runtime.

In production, it will be randomized at runtime to prevent hash-flooding attacks.

8.22.5.23 L0_OPT_STRING_EMPTY

```
l0_opt_string L0_OPT_STRING_EMPTY = { .has_value = 1, .value = { 0 } }  [static]
```

Static instance for empty optional string.

8.22.5.24 L0_OPT_STRING_NULL

```
l0_opt_string L0_OPT_STRING_NULL = { .has_value = 0, .value = { 0 } }  [static]
```

Static instance for null optional string.

8.22.5.25 L0_STRING_EMPTY

```
l0_string L0_STRING_EMPTY = { 0, { .s_str = { 0, NULL } } }  [static]
```

Static empty string instance.

8.23 l0_runtime.h

Go to the documentation of this file.

```
00001 #ifndef L0_RUNTIME_H
00002 #define L0_RUNTIME_H
00003
00004 /*
00005  * SPDX-License-Identifier: MIT OR Apache-2.0
00006  * Copyright (c) 2025-2026 gwz
00007  */
00008
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <stdint.h>
00037 #include <stddef.h>
00038 #include <string.h>
00039 #include <stdarg.h>
00040 #include <errno.h>
00041 #include <time.h>
00042 #include <sys/types.h>
00043 #include <sys/stat.h>
00044
00045 #ifndef DEA_RUNTIME_FCLOSE
00046 #define DEA_RUNTIME_FCLOSE fclose
00047 #endif
00048
00049 #ifndef DEA_RUNTIME_FILE_INFO_STAT
00050 #if defined(_WIN32)
00051 #define DEA_RUNTIME_FILE_INFO_STAT _stat64
00052 #else
00053 #define DEA_RUNTIME_FILE_INFO_STAT stat
00054 #endif
00055 #endif
00056
00057 #ifndef DEA_RUNTIME_REMOVE
00058 #define DEA_RUNTIME_REMOVE remove
00059 #endif
00060
00061 #if defined(_WIN32)
00062 #include <process.h>
00063 #else
00064 #include <unistd.h>
00065 #include <sys/wait.h>
00066 #endif
00067
00068 #include "dea_siphash.h"
00069
00070 #ifndef _RT_ALIGNOF
00071 #if defined(__clang__) || (defined(__GNUC__) && !defined(__TINYC__))
00072 #define _RT_ALIGNOF(type) __alignof__(type)
00073 #else
00074 #define _RT_ALIGNOF(type) (sizeof(struct { char _rt_align_c; type _rt_align_v; }) - sizeof(type))
00075 #endif
00076 #endif
00077
00078 #if defined(L0_RT_CHECK_BASIC) && defined(L0_RT_UNCHECKED)
00079 #error "L0_RT_CHECK_BASIC cannot be combined with L0_RT_UNCHECKED"
00080 #endif
00081
00082 /* ============================================================================
00083  * Compiler-specific builtins and attributes
00084  * ============================================================================ */
00085
00086 #if defined(__TINYC__) && __TINYC__ >= 928
00087 /* __builtin_unreachable added in mob branch post-0.9.27 */
00088 #   define L0_UNREACHABLE(_s) __builtin_unreachable()
00089 #elif defined(__GNUC__) || defined(__clang__)
00090 #   define L0_UNREACHABLE(_s) __builtin_unreachable()
00091 #else
00092 #   define L0_UNREACHABLE(_s) rt_panic(_s)
00093 #endif
00094
00095 /* ============================================================================
00096  * Optional tracing support (compile-time toggles)
00097  * ============================================================================ */
00098
00099 #if defined(L0_TRACE_ARC) || defined(L0_TRACE_MEMORY)
00100 #define _RT_TRACE_STDERR_BUFFER_SIZE (64u * 1024u)
00101
00102 static int _rt_trace_flush_each_event = 1;
00103 static int _rt_trace_output_started = 0;
00104 static char _rt_trace_stderr_buffer[_RT_TRACE_STDERR_BUFFER_SIZE];
00105
00107 static void _rt_trace_event_end(void) {
00108     _rt_trace_output_started = 1;
```

```
00109     if (_rt_trace_flush_each_event) {
00110         fflush(stderr);
00111     }
00112 }
00113
00115 static void _rt_trace_init(void) {
00116     const char *policy = getenv("DEA_TRACE_FLUSH");
00117
00118     if (_rt_trace_output_started) {
00119         return;
00120     }
00121     if (policy != NULL && strcmp(policy, "block") == 0) {
00122         if (setvbuf(
00123                 stderr,
00124                 _rt_trace_stderr_buffer,
00125                 _IOFBF,
00126                 sizeof(_rt_trace_stderr_buffer)
00127             ) == 0) {
00128             _rt_trace_flush_each_event = 0;
00129         }
00130     }
00131 }
00132 #endif
00133
00134 #ifdef L0_TRACE_ARC
00138 #define _RT_TRACE_ARC(...) \
00139     do { \
00140         fprintf(stderr, "[l0][arc] "); \
00141         fprintf(stderr, __VA_ARGS__); \
00142         fprintf(stderr, "\n"); \
00143         _rt_trace_event_end(); \
00144     } while (0)
00148 #define _RT_TRACE_ARC_LOC(loc_file, loc_line, ...) \
00149     do { \
00150         fprintf(stderr, "[l0][arc] "); \
00151         fprintf(stderr, __VA_ARGS__); \
00152         fprintf(stderr, " loc=\"%s\":%d", loc_file, loc_line); \
00153         fprintf(stderr, "\n"); \
00154         _rt_trace_event_end(); \
00155     } while (0)
00156 #else
00157 #define _RT_TRACE_ARC(...) ((void)0)
00158 #define _RT_TRACE_ARC_LOC(loc_file, loc_line, ...) ((void)0)
00159 #endif
00160
00161 #ifdef L0_TRACE_MEMORY
00165 #define _RT_TRACE_MEM(...) \
00166     do { \
00167         fprintf(stderr, "[l0][mem] "); \
00168         fprintf(stderr, __VA_ARGS__); \
00169         fprintf(stderr, "\n"); \
00170         _rt_trace_event_end(); \
00171     } while (0)
00175 #define _RT_TRACE_MEM_LOC(loc_file, loc_line, ...) \
00176     do { \
00177         fprintf(stderr, "[l0][mem] "); \
00178         fprintf(stderr, __VA_ARGS__); \
00179         fprintf(stderr, " loc=\"%s\":%d", loc_file, loc_line); \
00180         fprintf(stderr, "\n"); \
00181         _rt_trace_event_end(); \
00182     } while (0)
00183 #else
00184 #define _RT_TRACE_MEM(...) ((void)0)
00185 #define _RT_TRACE_MEM_LOC(loc_file, loc_line, ...) ((void)0)
00186 #endif
00187
00188 /* ============================================================================
00189  * Core type definitions
00190  * ============================================================================ */
00191
00192 typedef uint8_t  l0_bool;
00193
00194 typedef int8_t   l0_tiny;
00195 typedef int16_t  l0_short;
00196 typedef int32_t  l0_int;
00197 typedef int64_t  l0_long;
00198
00199 typedef uint8_t  l0_byte;
00200 typedef uint16_t l0_ushort;
00201 typedef uint32_t l0_uint;
00202 typedef uint64_t l0_ulong;
00203
00204 typedef float    l0_float;
00205 typedef double   l0_double;
00206
00222 typedef struct {
00223     l0_int refcount;
```

```
00224     l0_int len;
00225     char bytes[];
00226 } _l0_h_string;
00227 
00228 #define L0_STRING_K_STATIC  0
00229 #define L0_STRING_K_HEAP    1
00230 
00234 static const l0_int _RT_MEM_SENTINEL = 0xF00DB10C;
00235 
00240 typedef struct {
00241     unsigned int kind : 1;
00242     unsigned int : 0;
00243     union {
00244         struct {
00245             l0_int len;
00246             const char* bytes;
00247         } s_str;
00248         _l0_h_string *h_str;
00249     } data;
00250 } l0_string;
00251 
00255 static l0_string L0_STRING_EMPTY = { 0, { .s_str = { 0, NULL } } };
00256 
00260 #define L0_STRING_CONST(str_data, str_len) { .kind = L0_STRING_K_STATIC, .data.s_str = { .len =
      (str_len), .bytes = (str_data) } }
00261 
00262 /* ============================================================================
00263  * Optional type wrappers (T? as {has_value, value})
00264  * ============================================================================ */
00265 
00266 #ifndef L0_OPT_BOOL_DEFINED
00267 #define L0_OPT_BOOL_DEFINED
00269 typedef struct { l0_bool has_value; l0_bool value; } l0_opt_bool;
00270 #endif /* L0_OPT_BOOL_DEFINED */
00271 
00272 #ifndef L0_OPT_BYTE_DEFINED
00273 #define L0_OPT_BYTE_DEFINED
00275 typedef struct { l0_bool has_value; l0_byte value; } l0_opt_byte;
00276 #endif /* L0_OPT_BYTE_DEFINED */
00277 
00278 #ifndef L0_OPT_INT_DEFINED
00279 #define L0_OPT_INT_DEFINED
00281 typedef struct { l0_bool has_value; l0_int value; } l0_opt_int;
00282 #endif /* L0_OPT_INT_DEFINED */
00283 
00284 #ifndef L0_OPT_STRING_DEFINED
00285 #define L0_OPT_STRING_DEFINED
00287 typedef struct { l0_bool has_value; l0_string value; } l0_opt_string;
00288 #endif /* L0_OPT_STRING_DEFINED */
00289 
00291 typedef struct { l0_bool has_value; } _l0_base_opt;
00292 
00294 static l0_opt_string L0_OPT_STRING_NULL = { .has_value = 0, .value = { 0 } };
00296 static l0_opt_string L0_OPT_STRING_EMPTY = { .has_value = 1, .value = { 0 } };
00297 
00302 #ifndef L0_DEFINED_l0_sys_rt_RtTimeParts
00303 #define L0_DEFINED_l0_sys_rt_RtTimeParts
00304 struct l0_sys_rt_RtTimeParts {
00305     l0_int sec;
00306     l0_int nsec;
00307 };
00308 #endif
00309 
00314 #ifndef L0_DEFINED_l0_sys_rt_RtFileInfo
00315 #define L0_DEFINED_l0_sys_rt_RtFileInfo
00316 struct l0_sys_rt_RtFileInfo {
00317     l0_bool exists;
00318     l0_bool is_file;
00319     l0_bool is_dir;
00320     l0_opt_int size;
00321     l0_opt_int mtime_sec;
00322     l0_opt_int mtime_nsec;
00323 };
00324 #endif
00325 
00326 /* ============================================================================
00327  * Argument handling
00328  * ============================================================================ */
00329 
00330 static int _rt_argc = 0;
00331 static char** _rt_argv = NULL;
00332 
00339 void _rt_init_args(int argc, char** argv) {
00340 #if defined(L0_TRACE_ARC) || defined(L0_TRACE_MEMORY)
00341     _rt_trace_init();
00342 #endif
00343     _rt_argc = argc;
```

```
00344     _rt_argv = argv;
00345 }
00346
00347 /* ============================================================================
00348  * Panic mechanism
00349  * ============================================================================ */
00350
00356 static void _rt_panic(const char* message) {
00357     if (message == NULL) {
00358         message = "Guru Meditation";
00359     }
00360     fflush(stdout);
00361 #if defined(L0_TRACE_ARC) || defined(L0_TRACE_MEMORY)
00362     fflush(stderr);
00363 #endif
00364     fprintf(stderr, "Software Failure: %s\n", message);
00365     fflush(stderr);
00366     abort();
00367 }
00368
00374 static void _rt_panic_fmt(const char* fmt, ...) {
00375     va_list args;
00376     fflush(stdout);
00377 #if defined(L0_TRACE_ARC) || defined(L0_TRACE_MEMORY)
00378     fflush(stderr);
00379 #endif
00380     fprintf(stderr, "Software Failure: ");
00381     va_start(args, fmt);
00382     vfprintf(stderr, fmt, args);
00383     va_end(args);
00384     fprintf(stderr, "\n");
00385     fflush(stderr);
00386     abort();
00387 }
00388
00389 /* ============================================================================
00390  * UB-free integer helpers
00391  * ============================================================================ */
00392
00400 static l0_int _rt_idiv(l0_int a, l0_int b) {
00401     if (b == 0) {
00402         _rt_panic("division by zero");
00403     }
00404     if (a == INT32_MIN && b == -1) {
00405         _rt_panic("division overflow: INT32_MIN / -1");
00406     }
00407     return a / b;
00408 }
00409
00417 static l0_int _rt_imod(l0_int a, l0_int b) {
00418     if (b == 0) {
00419         _rt_panic("modulo by zero");
00420     }
00421     if (a == INT32_MIN && b == -1) {
00422         _rt_panic("modulo overflow: INT32_MIN % -1");
00423     }
00424     return a % b;
00425 }
00426
00434 static l0_int _rt_iadd(l0_int a, l0_int b) {
00435     if ((b > 0 && a > INT32_MAX - b) || (b < 0 && a < INT32_MIN - b)) {
00436         _rt_panic("integer addition overflow");
00437     }
00438     return a + b;
00439 }
00440
00448 static l0_int _rt_isub(l0_int a, l0_int b) {
00449     if ((b < 0 && a > INT32_MAX + b) || (b > 0 && a < INT32_MIN + b)) {
00450         _rt_panic("integer subtraction overflow");
00451     }
00452     return a - b;
00453 }
00454
00462 static l0_int _rt_imul(l0_int a, l0_int b) {
00463     /* Zero multiplication always succeeds */
00464     if (a == 0 || b == 0) {
00465         return 0;
00466     }
00467
00468     /* Special case: -1 * INT32_MIN or INT32_MIN * -1 = 2147483648, which overflows int32_t */
00469     if ((a == -1 && b == INT32_MIN) || (b == -1 && a == INT32_MIN)) {
00470         _rt_panic("integer multiplication overflow");
00471     }
00472
00473     /* Both operands positive: overflow if a > INT32_MAX / b */
00474     if (a > 0 && b > 0) {
00475         if (a > INT32_MAX / b) {
```

```
00476             _rt_panic("integer multiplication overflow");
00477         }
00478     }
00479     /* Both operands negative: result is positive, overflow if a < INT32_MAX / b
00480      * Note: We already handled the a=-1,b=INT32_MIN case above, so b != INT32_MIN here
00481      * and INT32_MAX / b is safe */
00482     else if (a < 0 && b < 0) {
00483         if (a < INT32_MAX / b) {
00484             _rt_panic("integer multiplication overflow");
00485         }
00486     }
00487     /* Mixed signs: result is negative or zero */
00488     else {
00489         /* We already handled the special cases where b=-1,a=INT32_MIN or a=-1,b=INT32_MIN
00490          * So all divisions below are safe from overflow */
00491         if (a > 0) {
00492             /* a > 0, b < 0: underflow if a > INT32_MIN / b
00493              * Since b != -1 (handled above), INT32_MIN / b is safe */
00494             if (b != -1 && a > INT32_MIN / b) {
00495                 _rt_panic("integer multiplication overflow");
00496             }
00497         } else {
00498             /* a < 0, b > 0: underflow if a < INT32_MIN / b */
00499             if (a < INT32_MIN / b) {
00500                 _rt_panic("integer multiplication overflow");
00501             }
00502         }
00503     }
00504
00505     return a * b;
00506 }
00507
00514 l0_byte _rt_narrow_l0_byte(l0_int value) {
00515     if (value < 0 || value > 255) {
00516         _rt_panic("int to byte cast overflow");
00517     }
00518     return (l0_byte)value;
00519 }
00520
00521 /* ============================================================================
00522  * UB-free optional type helpers
00523  * ============================================================================ */
00524
00532 static inline void *_unwrap_ptr(void *opt, const char *type_name) {
00533     if (opt == NULL) {
00534         _rt_panic_fmt("unwrap of empty optional: '%s'", type_name);
00535     }
00536     return opt;
00537 }
00538
00546 static inline void *_unwrap_opt(void *opt_ptr, const char *type_name) {
00547     _l0_base_opt *base = (_l0_base_opt*)opt_ptr;
00548     if (!base->has_value) {
00549         _rt_panic_fmt("unwrap of empty optional: '%s'", type_name);
00550     }
00551     return opt_ptr;
00552 }
00553
00554 /* ============================================================================
00555  * Runtime pointer access validation: types, configuration, and fast path
00556  * ----------------------------------------------------------------------------
00557  * Checked builds (the default) validate every generated pointer dereference
00558  * against the allocation tracker. Define L0_RT_UNCHECKED to compile all
00559  * pointer access validation and allocation tracking out; generated code is
00560  * identical in both modes.
00561  * ============================================================================ */
00562
00563 #ifndef _RT_ALLOC_INIT_CAP
00564 #define _RT_ALLOC_INIT_CAP 256
00565 #endif
00566 #define _RT_ALLOC_LIVE 1
00567 #define _RT_ALLOC_QUARANTINED 2
00568 #define _RT_ALLOC_POOLED 3
00569 #define _RT_MEM_RAW 0
00570 #define _RT_MEM_NEW 1
00571 #define _RT_MEM_ARC 2
00572 #define _RT_MEM_STATIC 3
00573 #define _RT_MEM_FOREIGN 4
00574 #define _RT_ACCESS_READ 0
00575 #define _RT_ACCESS_WRITE 1
00576 #define _RT_ACCESS_UNTRACKED_OK 2
00577 #ifndef _RT_QUARANTINE_MAX_BYTES
00578 #define _RT_QUARANTINE_MAX_BYTES ((size_t)16 * 1024 * 1024)
00579 #endif
00580 #ifndef _RT_QUARANTINE_MAX_COUNT
00581 #define _RT_QUARANTINE_MAX_COUNT ((size_t)4096)
00582 #endif
```

```
00583 #ifndef _RT_REC_POOL_CHUNK
00584 #define _RT_REC_POOL_CHUNK 256
00585 #endif
00586
00587 typedef struct _rt_alloc_record _rt_alloc_record;
00588 typedef struct _rt_alloc_record_cold _rt_alloc_record_cold;
00589 typedef struct _rt_ptr_site _rt_ptr_site;
00590
00597 struct _rt_alloc_record {
00598     void *base;
00599     size_t size;
00600     uint64_t generation;
00601 #ifdef L0_RT_CHECK_BASIC
00602     /* Sized from the pointer width so the hot layout matches the treap
00603      * fields of full checked mode on both 32-bit and 64-bit targets. */
00604     uint8_t tree_pad[2 * sizeof(_rt_alloc_record *)];
00605 #else
00606     _rt_alloc_record *tree_left;
00607     _rt_alloc_record *tree_right;
00608 #endif
00609     _rt_alloc_record *q_next;
00610 #ifdef L0_RT_CHECK_BASIC
00611     uint32_t tree_prio_pad;
00612 #else
00613     uint32_t tree_prio;
00614 #endif
00615     uint32_t cold_index;
00616     uint8_t state;
00617     uint8_t read_only;
00618     uint8_t mem_kind;
00619 #if UINTPTR_MAX == UINT32_MAX && SIZE_MAX == UINT32_MAX
00620     uint8_t hot_pad[25];
00621 #else
00622     uint8_t hot_pad[5];
00623 #endif
00624 };
00625
00626 typedef char _rt_alloc_record_size_check[(sizeof(_rt_alloc_record) == 64) ? 1 : -1];
00627
00631 struct _rt_alloc_record_cold {
00632     size_t align;
00633     uint32_t type_id; /* Reserved for a future runtime type-identity initiative. */
00634     int alloc_line;
00635     int drop_line;
00636     uint32_t reserved;
00637     const char *alloc_file;
00638     const char *drop_file;
00639 };
00640
00646 struct _rt_ptr_site {
00647     _rt_alloc_record *owner;
00648     uint64_t generation;
00649 };
00650
00651 static void _rt_track_alloc_record(
00652     void *ptr,
00653     size_t size,
00654     size_t align,
00655     uint32_t type_id,
00656     const char *loc_file,
00657     int loc_line
00658 );
00659 static void _rt_promote_new_alloc(void *ptr);
00660 static void _rt_release_tracked_alloc(void *ptr, const char *loc_file, int loc_line, const char
      *op_name);
00661 static void *_rt_realloc_tracked_alloc(void *ptr, size_t new_size, const char *loc_file, int
      loc_line);
00662 #ifndef L0_RT_UNCHECKED
00663 static void *_rt_check_ptr_site_slow(_rt_ptr_site *site, void *ptr, l0_int required_size, l0_int
      required_align, int access_mode, const char *loc_file, int loc_line);
00664 static void *_rt_check_index_ptr_site_slow(_rt_ptr_site *site, void *base_ptr, l0_int index, l0_int
      element_size, l0_int required_align, int access_mode, const char *loc_file, int loc_line);
00665 #endif
00666 static void *_rt_drop_begin_impl(
00667     void *ptr,
00668     l0_int required_size,
00669     l0_int required_align,
00670     const char *loc_file,
00671     int loc_line
00672 );
00673 static void _rt_drop_finish_impl(void *ptr, const char *loc_file, int loc_line);
00674 static void *_rt_validate_derived_ptr(void *derived, void *parent_base, l0_int size, l0_int align,
      const char *loc_file, int loc_line);
00675 static void _rt_track_arc_bytes(void *ptr, size_t size);
00676 static void _rt_untrack_arc_alloc(void *ptr);
00677 static void _rt_track_static_bytes(const void *ptr, size_t size);
00678
```

```
00691 static inline void *_rt_check_ptr_site(_rt_ptr_site *site, void *ptr, l0_int required_size, l0_int
      required_align, int access_mode, const char *loc_file, int loc_line) {
00692 #ifdef L0_RT_UNCHECKED
00693     (void)site; (void)required_size; (void)required_align; (void)access_mode; (void)loc_file;
      (void)loc_line;
00694     return ptr;
00695 #else
00696     _rt_alloc_record *owner = site != NULL ? site->owner : NULL;
00697     if (owner != NULL && owner->generation == site->generation &&
00698         owner->state == _RT_ALLOC_LIVE && required_size > 0 &&
00699         required_align >= 0 &&
00700         (access_mode == _RT_ACCESS_READ || access_mode == _RT_ACCESS_WRITE) &&
00701         (access_mode != _RT_ACCESS_WRITE || !owner->read_only)) {
00702         uintptr_t base = (uintptr_t)owner->base;
00703         uintptr_t addr = (uintptr_t)ptr;
00704         if (addr >= base) {
00705             size_t offset = (size_t)(addr - base);
00706             if (offset <= owner->size && (size_t)required_size <= owner->size - offset &&
00707                 (required_align <= 1 || (addr % (size_t)required_align) == 0)) {
00708                 return ptr;
00709             }
00710         }
00711     }
00712     return _rt_check_ptr_site_slow(site, ptr, required_size, required_align, access_mode, loc_file,
      loc_line);
00713 #endif
00714 }
00715
00733 static inline void *_rt_check_index_ptr_site(
00734     _rt_ptr_site *site,
00735     void *base_ptr,
00736     l0_int index,
00737     l0_int element_size,
00738     l0_int required_align,
00739     int access_mode,
00740     const char *loc_file,
00741     int loc_line
00742 ) {
00743 #ifdef L0_RT_UNCHECKED
00744     uint64_t offset = (uint64_t)((int64_t)index * (int64_t)element_size);
00745     (void)site; (void)required_align; (void)access_mode; (void)loc_file; (void)loc_line;
00746     return (void *)((uintptr_t)base_ptr + (uintptr_t)offset);
00747 #else
00748     _rt_alloc_record *owner = site != NULL ? site->owner : NULL;
00749     int index_mode = access_mode & _RT_ACCESS_WRITE;
00750     int index_flags = access_mode & ~_RT_ACCESS_WRITE;
00751     if (owner != NULL && owner->generation == site->generation &&
00752         owner->state == _RT_ALLOC_LIVE && index >= 0 && element_size > 0 &&
00753         required_align >= 0 &&
00754         (index_flags == 0 || index_flags == _RT_ACCESS_UNTRACKED_OK) &&
00755         (index_mode == _RT_ACCESS_READ || index_mode == _RT_ACCESS_WRITE) &&
00756         (index_mode != _RT_ACCESS_WRITE || !owner->read_only)) {
00757         /* `l0_int` is 32-bit, so the 64-bit product of two non-negative
00758          * operands cannot wrap and the SIZE_MAX comparison is exact. */
00759         uint64_t offset64 = (uint64_t)index * (uint64_t)element_size;
00760         if (offset64 <= SIZE_MAX) {
00761             uintptr_t owner_addr = (uintptr_t)owner->base;
00762             uintptr_t base_addr = (uintptr_t)base_ptr;
00763             if (base_addr >= owner_addr) {
00764                 size_t base_offset = (size_t)(base_addr - owner_addr);
00765                 if (base_offset < owner->size) {
00766                     size_t offset = (size_t)offset64;
00767                     size_t available = owner->size - base_offset;
00768                     if (offset <= available && (size_t)element_size <= available - offset) {
00769                         size_t target_offset = base_offset + offset;
00770                         if (target_offset <= UINTPTR_MAX - owner_addr) {
00771                             uintptr_t target_addr = owner_addr + target_offset;
00772                             if (required_align <= 1 || (target_addr % (size_t)required_align) == 0) {
00773                                 return (void *)target_addr;
00774                             }
00775                         }
00776                     }
00777                 }
00778             }
00779         }
00780     }
00781     return _rt_check_index_ptr_site_slow(site, base_ptr, index, element_size, required_align,
      access_mode, loc_file, loc_line);
00782 #endif
00783 }
00784
00785 /* ============================================================================
00786  * String construction and operations
00787  * ============================================================================ */
00788
00799 static l0_string _rt_l0_string_from_const_literal(const char *c_str) {
00800     l0_string s;
```

```
00801     if (c_str == NULL) {
00802         return L0_STRING_EMPTY;
00803     } else {
00804         size_t len = strlen(c_str);
00805         if (len > INT32_MAX) {
00806             _rt_panic("_rt_l0_string_from_const_literal: string too long for l0_int");
00807         }
00808         s.kind = L0_STRING_K_STATIC;
00809         s.data.s_str.len = (l0_int)len;
00810         s.data.s_str.bytes = c_str;
00811     }
00812     return s;
00813 }
00814
00828 static l0_string _rt_init_heap_string(void *mem, l0_int s_len) {
00829     l0_string s;
00830     _l0_h_string *hs = (_l0_h_string *)mem;
00831     hs->refcount = 1;       /* reference counted */
00832     hs->len = (l0_int)s_len;
00833     hs->bytes[s_len] = '\0';   /* null-terminate */
00834
00835     s.kind = L0_STRING_K_HEAP;
00836     s.data.h_str = hs;
00837     return s;
00838 }
00839
00852 #ifdef L0_TRACE_MEMORY
00853 static l0_string _rt_alloc_string_impl(l0_int len, const char *_loc_file, int _loc_line) {
00854     if (len < 0) {
00855         _rt_panic("_rt_alloc_string: negative length");
00856     }
00857     void *mem = malloc(sizeof(_l0_h_string) + len + 1);
00858     if (mem == NULL) {
00859         _rt_panic("_rt_alloc_string: out of memory");
00860     }
00861     l0_string s = _rt_init_heap_string(mem, len);
00862     _RT_TRACE_MEM_LOC(_loc_file, _loc_line, "op=alloc_string len=%d ptr=%p", (int)len,
      (void*)s.data.h_str);
00863     return s;
00864 }
00865 #define _rt_alloc_string(len) _rt_alloc_string_impl((len), __FILE__, __LINE__)
00866 #else
00867 static l0_string _rt_alloc_string(l0_int len) {
00868     if (len < 0) {
00869         _rt_panic("_rt_alloc_string: negative length");
00870     }
00871     void *mem = malloc(sizeof(_l0_h_string) + len + 1);
00872     if (mem == NULL) {
00873         _rt_panic("_rt_alloc_string: out of memory");
00874     }
00875     l0_string s = _rt_init_heap_string(mem, len);
00876     _RT_TRACE_MEM("op=alloc_string len=%d ptr=%p", (int)len, (void*)s.data.h_str);
00877     return s;
00878 }
00879 #endif
00880
00887 #if defined(L0_TRACE_ARC) || defined(L0_TRACE_MEMORY)
00888 static void _rt_free_string_impl(l0_string str, const char *_loc_file, int _loc_line) {
00889     if (str.kind == L0_STRING_K_STATIC) {
00890         /* Static string: do nothing */
00891         _RT_TRACE_ARC_LOC(_loc_file, _loc_line, "op=release kind=static ptr=%p rc_before=-1
      rc_after=-1 action=noop", (void*)str.data.s_str.bytes);
00892         return;
00893     }
00894     _l0_h_string *hs = str.data.h_str;
00895     if (hs == NULL) {
00896         _RT_TRACE_ARC_LOC(_loc_file, _loc_line, "op=release kind=heap ptr=%p rc_before=-1 rc_after=-1
      action=panic-null-ptr", (void*)hs);
00897         _RT_TRACE_MEM_LOC(_loc_file, _loc_line, "op=free_string ptr=%p action=panic-null-ptr",
      (void*)hs);
00898         _rt_panic("_rt_free_string: null heap string pointer");
00899     }
00900     l0_int rc_before = hs->refcount;
00901     if (rc_before > 0 && rc_before < INT32_MAX) {
00902         /* Reference counted string */
00903         hs->refcount--;
00904         if (hs->refcount == 0) {
00905             _RT_TRACE_ARC_LOC(
00906                 _loc_file, _loc_line,
00907                 "op=release kind=heap ptr=%p rc_before=%d rc_after=0 action=free",
00908                 (void*)hs, (int)rc_before
00909             );
00910             _RT_TRACE_MEM_LOC(_loc_file, _loc_line, "op=free_string ptr=%p action=free", (void*)hs);
00911             hs->refcount = _RT_MEM_SENTINEL; /* prevent double free */
00912             _rt_untrack_arc_alloc((void*)hs->bytes);
00913             free((void*)hs);
00914         } else {
```

```
00915             _RT_TRACE_ARC_LOC(
00916                 _loc_file, _loc_line,
00917                 "op=release kind=heap ptr=%p rc_before=%d rc_after=%d action=keep",
00918                 (void*)hs, (int)rc_before, (int)hs->refcount
00919             );
00920             _RT_TRACE_MEM_LOC(_loc_file, _loc_line, "op=free_string ptr=%p action=decrement-only",
    (void*)hs);
00921         }
00922         return;
00923     }
00924     if (rc_before == INT32_MAX) {
00925         /* Non-reference counted string: do nothing */
00926         _RT_TRACE_ARC_LOC(
00927             _loc_file, _loc_line,
00928             "op=release kind=heap ptr=%p rc_before=%d rc_after=%d action=noop-nonref",
00929             (void*)hs, (int)rc_before, (int)rc_before
00930         );
00931         _RT_TRACE_MEM_LOC(_loc_file, _loc_line, "op=free_string ptr=%p action=noop-nonref",
    (void*)hs);
00932         return;
00933     }
00934     if (rc_before == _RT_MEM_SENTINEL) {
00935         _RT_TRACE_ARC_LOC(
00936             _loc_file, _loc_line,
00937             "op=release kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-double-free",
00938             (void*)hs, (int)rc_before, (int)rc_before
00939         );
00940         _RT_TRACE_MEM_LOC(_loc_file, _loc_line, "op=free_string ptr=%p action=panic-double-free",
    (void*)hs);
00941         _rt_panic("_rt_free_string: double free detected");
00942     }
00943     _RT_TRACE_ARC_LOC(
00944         _loc_file, _loc_line,
00945         "op=release kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-invalid-state",
00946         (void*)hs, (int)rc_before, (int)rc_before
00947     );
00948     _RT_TRACE_MEM_LOC(_loc_file, _loc_line, "op=free_string ptr=%p action=panic-invalid-state",
    (void*)hs);
00949     _rt_panic_fmt("_rt_free_string: invalid string refcount state: %d", (int)hs->refcount);
00950 }
00951 #define _rt_free_string(str) _rt_free_string_impl((str), __FILE__, __LINE__)
00952 #else
00953 static void _rt_free_string(l0_string str) {
00954     if (str.kind == L0_STRING_K_STATIC) {
00955         /* Static string: do nothing */
00956         _RT_TRACE_ARC("op=release kind=static ptr=%p rc_before=-1 rc_after=-1 action=noop",
    (void*)str.data.s_str.bytes);
00957         return;
00958     }
00959     _l0_h_string *hs = str.data.h_str;
00960     if (hs == NULL) {
00961         _RT_TRACE_ARC("op=release kind=heap ptr=%p rc_before=-1 rc_after=-1 action=panic-null-ptr",
    (void*)hs);
00962         _RT_TRACE_MEM("op=free_string ptr=%p action=panic-null-ptr", (void*)hs);
00963         _rt_panic("_rt_free_string: null heap string pointer");
00964     }
00965     l0_int rc_before = hs->refcount;
00966     if (rc_before > 0 && rc_before < INT32_MAX) {
00967         /* Reference counted string */
00968         hs->refcount--;
00969         if (hs->refcount == 0) {
00970             _RT_TRACE_ARC(
00971                 "op=release kind=heap ptr=%p rc_before=%d rc_after=0 action=free",
00972                 (void*)hs, (int)rc_before
00973             );
00974             _RT_TRACE_MEM("op=free_string ptr=%p action=free", (void*)hs);
00975             hs->refcount = _RT_MEM_SENTINEL; /* prevent double free */
00976             _rt_untrack_arc_alloc((void*)hs->bytes);
00977             free((void*)hs);
00978         } else {
00979             _RT_TRACE_ARC(
00980                 "op=release kind=heap ptr=%p rc_before=%d rc_after=%d action=keep",
00981                 (void*)hs, (int)rc_before, (int)hs->refcount
00982             );
00983             _RT_TRACE_MEM("op=free_string ptr=%p action=decrement-only", (void*)hs);
00984         }
00985         return;
00986     }
00987     if (rc_before == INT32_MAX) {
00988         /* Non-reference counted string: do nothing */
00989         _RT_TRACE_ARC(
00990             "op=release kind=heap ptr=%p rc_before=%d rc_after=%d action=noop-nonref",
00991             (void*)hs, (int)rc_before, (int)rc_before
00992         );
00993         _RT_TRACE_MEM("op=free_string ptr=%p action=noop-nonref", (void*)hs);
00994         return;
00995     }
```

```
00996     if (rc_before == _RT_MEM_SENTINEL) {
00997         _RT_TRACE_ARC(
00998             "op=release kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-double-free",
00999             (void*)hs, (int)rc_before, (int)rc_before
01000         );
01001         _RT_TRACE_MEM("op=free_string ptr=%p action=panic-double-free", (void*)hs);
01002         _rt_panic("_rt_free_string: double free detected");
01003     }
01004     _RT_TRACE_ARC(
01005         "op=release kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-invalid-state",
01006         (void*)hs, (int)rc_before, (int)rc_before
01007     );
01008     _RT_TRACE_MEM("op=free_string ptr=%p action=panic-invalid-state", (void*)hs);
01009     _rt_panic_fmt("_rt_free_string: invalid string refcount state: %d", (int)hs->refcount);
01010 }
01011 #endif
01012
01020 static l0_string _rt_realloc_string(l0_string s, l0_int new_len) {
01021     if (new_len < 0) {
01022         _rt_panic("_rt_realloc_string: negative length");
01023     }
01024     if (new_len == 0) {
01025         _rt_free_string(s);
01026         return L0_STRING_EMPTY;
01027     }
01028     if (s.kind == L0_STRING_K_STATIC && s.data.s_str.len == 0) {
01029         /* Reallocating empty static string: allocate new heap string */
01030         return _rt_alloc_string(new_len);
01031     }
01032     if (s.kind != L0_STRING_K_HEAP || s.data.h_str == NULL) {
01033         _RT_TRACE_MEM("op=realloc_string old_ptr=%p new_len=%d action=panic-invalid-string",
    (void*)s.data.h_str, (int)new_len);
01034         _rt_panic("_rt_realloc_string: string is not heap-allocated");
01035     }
01036
01037     /* Use volatile to prevent the compiler from tracking the pointer across realloc
01038        and complaining about use-after-free when tracing the old pointer value. */
01039     volatile uintptr_t old_ptr_addr = (uintptr_t)s.data.h_str;
01040     size_t new_size = sizeof(_l0_h_string) + new_len + 1;
01041     _rt_untrack_arc_alloc(s.data.h_str->bytes);
01042     void *new_mem = realloc((void*)old_ptr_addr, new_size);
01043     if (new_mem == NULL) {
01044         _RT_TRACE_MEM("op=realloc_string old_ptr=%p new_len=%d action=panic-oom", (void*)old_ptr_addr,
    (int)new_len);
01045         _rt_panic("_rt_realloc_string: out of memory");
01046     }
01047     _l0_h_string *new_hs = (_l0_h_string *)new_mem;
01048     new_hs->len = new_len;
01049     new_hs->bytes[new_len] = '\0'; /* null-terminate */
01050     s.data.h_str = new_hs;
01051     _RT_TRACE_MEM(
01052         "op=realloc_string old_ptr=%p new_ptr=%p new_len=%d action=ok",
01053         (void*)old_ptr_addr, (void*)new_hs, (int)new_len
01054     );
01055     return s;
01056 }
01057
01065 static l0_string _rt_new_l0_string(const char *c_str) {
01066     if (c_str == NULL) {
01067         return L0_STRING_EMPTY;
01068     }
01069     size_t len = strlen(c_str);
01070     if ((uint64_t)len > INT32_MAX) {
01071         _rt_panic("_rt_new_l0_string: string too long for l0_int");
01072     }
01073     l0_string s = _rt_alloc_string((l0_int)len);
01074     _l0_h_string *hs = s.data.h_str;
01075     memcpy(hs->bytes, c_str, len + 1);
01076
01077     return s;
01078 }
01079
01090 static char *_rt_string_bytes(l0_string s) {
01091     switch (s.kind) {
01092         case L0_STRING_K_STATIC:
01093             return (char*)s.data.s_str.bytes;
01094         case L0_STRING_K_HEAP:
01095             if (s.data.h_str != NULL) {
01096                 return s.data.h_str->bytes;
01097             }
01098             /* fallthrough */
01099         default:
01100             _rt_panic_fmt("_rt_string_bytes: invalid string kind: %d or null data", (int)s.kind);
01101             return NULL; /* Unreachable */
01102     }
01103 }
01104
```

```
01105 /* ============================================================================
01106  * User string operations
01107  * ============================================================================ */
01108
01117 static l0_int rt_strlen(l0_string str) {
01118     switch(str.kind) {
01119     case L0_STRING_K_STATIC:
01120         return str.data.s_str.len;
01121     case L0_STRING_K_HEAP:
01122         if (str.data.h_str == NULL) {
01123             _rt_panic("rt_strlen: string data is null");
01124             return 0; /* Unreachable */
01125         }
01126         return str.data.h_str->len;
01127     default:
01128         _rt_panic_fmt("rt_strlen: invalid string kind: %d", (int)str.kind);
01129         return 0; /* Unreachable */
01130     }
01131 }
01132
01143 static l0_byte rt_string_get(l0_string a, l0_int index) {
01144     l0_int a_len = rt_strlen(a);
01145     if (index < 0 || index >= a_len) {
01146         _rt_panic_fmt("rt_string_get: index %d out of bounds for string of length %d",
01147                       (int)index, (int)a_len);
01148     }
01149     char *a_data = _rt_string_bytes(a);
01150     if (a_data == NULL) {
01151         _rt_panic("rt_string_get: string data is null");
01152     }
01153     return (l0_byte)a_data[index];
01154 }
01155
01170 static l0_byte *rt_string_bytes_ptr(l0_string s) {
01171     char *bytes = _rt_string_bytes(s);
01172     if (s.kind == L0_STRING_K_STATIC && bytes != NULL) {
01173         _rt_track_static_bytes(bytes, (size_t)s.data.s_str.len + 1);
01174     } else if (s.kind == L0_STRING_K_HEAP && s.data.h_str != NULL) {
01175         _l0_h_string *hs = s.data.h_str;
01176         _rt_track_arc_bytes((void*)hs->bytes, (size_t)hs->len + 1);
01177     }
01178     return (l0_byte*)bytes;
01179 }
01180
01190 static l0_bool rt_string_equals(l0_string a, l0_string b) {
01191     l0_int a_len = rt_strlen(a);
01192     l0_int b_len = rt_strlen(b);
01193     if (a_len != b_len) {
01194         return 0;
01195     }
01196     if (a_len == 0) {
01197         return 1;  /* Both empty */
01198     }
01199     char *a_data = _rt_string_bytes(a);
01200     char *b_data = _rt_string_bytes(b);
01201     if (a_data == NULL || b_data == NULL) {
01202         _rt_panic("rt_string_equals: invalid state - string data is null");
01203     }
01204     return memcmp(a_data, b_data, (size_t)a_len) == 0 ? 1 : 0;
01205 }
01206
01217 static l0_int rt_string_compare(l0_string a, l0_string b) {
01218     l0_int a_len = rt_strlen(a);
01219     l0_int b_len = rt_strlen(b);
01220
01221     l0_int min_len = a_len;
01222     if (b_len < min_len) {
01223         min_len = b_len;
01224     }
01225
01226     if (min_len > 0) {
01227         char *a_data = _rt_string_bytes(a);
01228         char *b_data = _rt_string_bytes(b);
01229         if (a_data == NULL || b_data == NULL) {
01230             _rt_panic("rt_string_compare: string data is null");
01231         }
01232
01233         int result = memcmp(a_data, b_data, (size_t)min_len);
01234         if (result < 0) {
01235             return -1;
01236         }
01237         if (result > 0) {
01238             return 1;
01239         }
01240     }
01241
01242     if (a_len < b_len) {
```

```
01243         return -1;
01244     }
01245     if (a_len > b_len) {
01246         return 1;
01247     }
01248     return 0;
01249 }
01250
01261 #ifdef L0_TRACE_MEMORY
01262 static l0_string _rt_string_concat_impl(l0_string a, l0_string b, const char *_loc_file, int
      _loc_line) {
01263     l0_int a_len = rt_strlen(a);
01264     l0_int b_len = rt_strlen(b);
01265
01266     /* Check for overflow in total length */
01267     if (a_len > INT32_MAX - b_len) {
01268         _rt_panic("rt_string_concat: combined length too large for l0_int");
01269     }
01270
01271     l0_int total_len = a_len + b_len;
01272
01273     if (total_len == 0) {
01274         return L0_STRING_EMPTY;
01275     }
01276
01277     l0_string s = _rt_alloc_string_impl(total_len, _loc_file, _loc_line); /* result string */
01278     char *s_data = _rt_string_bytes(s);
01279     char *a_data = _rt_string_bytes(a);
01280     char *b_data = _rt_string_bytes(b);
01281     if (s_data == NULL) {
01282         _rt_panic("rt_string_concat: result string data is null");
01283     }
01284     if (a_data != NULL && a_len > 0) {
01285         memcpy(s_data, a_data, (size_t)a_len);
01286     }
01287     if (b_data != NULL && b_len > 0) {
01288         memcpy(s_data + a_len, b_data, (size_t)b_len);
01289     }
01290     s_data[total_len] = '\0'; /* null-terminate */
01291     return s;
01292 }
01293 #define rt_string_concat(a, b) _rt_string_concat_impl((a), (b), __FILE__, __LINE__)
01294 #else
01295 static l0_string rt_string_concat(l0_string a, l0_string b) {
01296     l0_int a_len = rt_strlen(a);
01297     l0_int b_len = rt_strlen(b);
01298
01299     /* Check for overflow in total length */
01300     if (a_len > INT32_MAX - b_len) {
01301         _rt_panic("rt_string_concat: combined length too large for l0_int");
01302     }
01303
01304     l0_int total_len = a_len + b_len;
01305
01306     if (total_len == 0) {
01307         return L0_STRING_EMPTY;
01308     }
01309
01310     l0_string s = _rt_alloc_string(total_len); /* result string */
01311     char *s_data = _rt_string_bytes(s);
01312     char *a_data = _rt_string_bytes(a);
01313     char *b_data = _rt_string_bytes(b);
01314     if (s_data == NULL) {
01315         _rt_panic("rt_string_concat: result string data is null");
01316     }
01317     if (a_data != NULL && a_len > 0) {
01318         memcpy(s_data, a_data, (size_t)a_len);
01319     }
01320     if (b_data != NULL && b_len > 0) {
01321         memcpy(s_data + a_len, b_data, (size_t)b_len);
01322     }
01323     s_data[total_len] = '\0'; /* null-terminate */
01324     return s;
01325 }
01326 #endif
01327
01339 static l0_string rt_string_slice(l0_string s, l0_int start, l0_int end) {
01340     l0_int s_len = rt_strlen(s);
01341     if (start < 0 || start > s_len) {
01342         _rt_panic_fmt("rt_string_slice: start %d out of bounds for string of length %d",
01343                       (int)start, (int)s_len);
01344     }
01345     if (end < start || end > s_len) {
01346         _rt_panic_fmt("rt_string_slice: end %d invalid for start %d, string length %d",
01347                       (int)end, (int)start, (int)s_len);
01348     }
01349
```

```
01350     l0_int slice_len = end - start;
01351
01352     if (slice_len == 0) {
01353         return L0_STRING_EMPTY;
01354     }
01355
01356     l0_string result = _rt_alloc_string(slice_len);
01357     char *s_data = _rt_string_bytes(s);
01358     char *d_data = _rt_string_bytes(result);
01359     memcpy(d_data, s_data + start, (size_t)slice_len);
01360     d_data[slice_len] = '\0';
01361
01362     return result;
01363 }
01364
01375 static l0_string rt_string_from_byte(l0_byte b) {
01376     l0_string s = _rt_alloc_string(1);
01377     char *s_data = _rt_string_bytes(s);
01378     s_data[0] = (char)b;
01379     s_data[1] = '\0'; /* null-terminate */
01380     return s;
01381 }
01382
01396 static l0_string rt_string_from_byte_array(l0_byte* bytes, l0_int len) {
01397     if (len < 0) {
01398         _rt_panic("rt_string_from_byte_array: negative length");
01399     }
01400     l0_string s = _rt_alloc_string(len);
01401     char *s_data = _rt_string_bytes(s);
01402     memcpy(s_data, bytes, (size_t)len);
01403     return s;
01404 }
01405
01414 #ifdef L0_TRACE_ARC
01415 static void _rt_string_retain_impl(l0_string s, const char *_loc_file, int _loc_line) {
01416     if (s.kind == L0_STRING_K_STATIC) {
01417         _RT_TRACE_ARC("op=retain kind=static ptr=%p rc_before=-1 rc_after=-1 action=noop
      loc=\"%s\":%d", (void*)s.data.s_str.bytes, _loc_file, _loc_line);
01418         return; /* Static strings are not reference counted */
01419     }
01420     _l0_h_string *hs = s.data.h_str;
01421     if (hs == NULL) {
01422         _RT_TRACE_ARC("op=retain kind=heap ptr=%p rc_before=-1 rc_after=-1 action=panic-null-ptr
      loc=\"%s\":%d", (void*)hs, _loc_file, _loc_line);
01423         _rt_panic("rt_string_retain: null heap string pointer");
01424     }
01425     l0_int rc_before = hs->refcount;
01426     if (rc_before == _RT_MEM_SENTINEL) {
01427         _RT_TRACE_ARC(
01428             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-use-after-free
      loc=\"%s\":%d",
01429             (void*)hs, (int)rc_before, (int)rc_before, _loc_file, _loc_line
01430         );
01431         _rt_panic("rt_string_retain: use after free");
01432     }
01433     if (rc_before > 0 && rc_before < INT32_MAX - 1) {
01434         hs->refcount++;
01435         _RT_TRACE_ARC(
01436             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=retain loc=\"%s\":%d",
01437             (void*)hs, (int)rc_before, (int)hs->refcount, _loc_file, _loc_line
01438         );
01439     } else if (rc_before == INT32_MAX - 1) {
01440         _RT_TRACE_ARC(
01441             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-overflow loc=\"%s\":%d",
01442             (void*)hs, (int)rc_before, (int)rc_before, _loc_file, _loc_line
01443         );
01444         _rt_panic_fmt("rt_string_retain: invalid refcount state: %d", (int)hs->refcount);
01445     } else if (hs->refcount == INT32_MAX) {
01446         /* Non-refcounted heap string: no-op */
01447         _RT_TRACE_ARC(
01448             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=noop-nonref loc=\"%s\":%d",
01449             (void*)hs, (int)rc_before, (int)rc_before, _loc_file, _loc_line
01450         );
01451     } else {
01452         _RT_TRACE_ARC(
01453             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-invalid-state
      loc=\"%s\":%d",
01454             (void*)hs, (int)rc_before, (int)rc_before, _loc_file, _loc_line
01455         );
01456         _rt_panic_fmt("rt_string_retain: invalid refcount state: %d", (int)hs->refcount);
01457     }
01458 }
01459 #define rt_string_retain(s) _rt_string_retain_impl((s), __FILE__, __LINE__)
01460 #else
01461 static void rt_string_retain(l0_string s) {
01462     if (s.kind == L0_STRING_K_STATIC) {
01463         _RT_TRACE_ARC("op=retain kind=static ptr=%p rc_before=-1 rc_after=-1 action=noop",
```

```
    (void*)s.data.s_str.bytes);
01464         return; /* Static strings are not reference counted */
01465     }
01466     _l0_h_string *hs = s.data.h_str;
01467     if (hs == NULL) {
01468         _RT_TRACE_ARC("op=retain kind=heap ptr=%p rc_before=-1 rc_after=-1 action=panic-null-ptr",
    (void*)hs);
01469         _rt_panic("rt_string_retain: null heap string pointer");
01470     }
01471     l0_int rc_before = hs->refcount;
01472     if (rc_before == _RT_MEM_SENTINEL) {
01473         _RT_TRACE_ARC(
01474             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-use-after-free",
01475             (void*)hs, (int)rc_before, (int)rc_before
01476         );
01477         _rt_panic("rt_string_retain: use after free");
01478     }
01479     if (rc_before > 0 && rc_before < INT32_MAX - 1) {
01480         hs->refcount++;
01481         _RT_TRACE_ARC(
01482             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=retain",
01483             (void*)hs, (int)rc_before, (int)hs->refcount
01484         );
01485     } else if (rc_before == INT32_MAX - 1) {
01486         _RT_TRACE_ARC(
01487             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-overflow",
01488             (void*)hs, (int)rc_before, (int)rc_before
01489         );
01490         _rt_panic_fmt("rt_string_retain: invalid refcount state: %d", (int)hs->refcount);
01491     } else if (hs->refcount == INT32_MAX) {
01492         /* Non-refcounted heap string: no-op */
01493         _RT_TRACE_ARC(
01494             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=noop-nonref",
01495             (void*)hs, (int)rc_before, (int)rc_before
01496         );
01497     } else {
01498         _RT_TRACE_ARC(
01499             "op=retain kind=heap ptr=%p rc_before=%d rc_after=%d action=panic-invalid-state",
01500             (void*)hs, (int)rc_before, (int)rc_before
01501         );
01502         _rt_panic_fmt("rt_string_retain: invalid refcount state: %d", (int)hs->refcount);
01503     }
01504 }
01505 #endif
01506
01514 #ifdef L0_TRACE_ARC
01515 static void _rt_string_release_impl(l0_string s, const char *_loc_file, int _loc_line) {
01516     _rt_free_string_impl(s, _loc_file, _loc_line);
01517 }
01518 #define rt_string_release(s) _rt_string_release_impl((s), __FILE__, __LINE__)
01519 #else
01520 static void rt_string_release(l0_string s) {
01521     _rt_free_string(s);
01522 }
01523 #endif
01524
01525 /* ============================================================================
01526  * System interaction and environment
01527  * ============================================================================ */
01528
01539 static l0_int rt_system(l0_string cmd) {
01540     char *c = _rt_string_bytes(cmd);
01541 #if defined(L0_TRACE_ARC) || defined(L0_TRACE_MEMORY)
01542     fflush(stderr);
01543 #endif
01544     int status = system(c);
01545 #if defined(_WIN32)
01546     return (l0_int)status;
01547 #else
01548     if (status < 0) {
01549         return (l0_int)status;
01550     }
01551     if (WIFEXITED(status)) {
01552         return (l0_int)WEXITSTATUS(status);
01553     }
01554     if (WIFSIGNALED(status)) {
01555         return (l0_int)(128 + WTERMSIG(status));
01556     }
01557     return (l0_int)status;
01558 #endif
01559 }
01560
01570 static l0_opt_string rt_get_env_var(l0_string name) {
01571     if (rt_strlen(name) == 0) {
01572         return L0_OPT_STRING_NULL;
01573     }
01574
```

```
01575     /* Get the underlying null-terminated char[] */
01576     char *c_name = _rt_string_bytes(name);
01577     if (c_name == NULL) {
01578         return L0_OPT_STRING_NULL;
01579     }
01580
01581     /* Get environment variable */
01582     char *c_value = getenv(c_name);
01583
01584     if (c_value == NULL) {
01585         return L0_OPT_STRING_NULL;
01586     }
01587
01588     /* Convert value to L0 string*? */
01589     l0_string result = _rt_new_l0_string(c_value);
01590     return (l0_opt_string){ .has_value = 1, .value = result };
01591 }
01592
01600 static l0_int rt_get_argc(void) {
01601     return (l0_int)_rt_argc;
01602 }
01603
01611 static l0_bool _rt_pid_to_l0_int(intmax_t value, l0_int *out) {
01612     if (value < 0 || value > INT32_MAX) {
01613         return 0;
01614     }
01615     *out = (l0_int)value;
01616     return 1;
01617 }
01618
01626 static l0_int rt_get_pid(void) {
01627     l0_int out = 0;
01628 #if defined(_WIN32)
01629     if (!_rt_pid_to_l0_int((intmax_t)_getpid(), &out)) {
01630         _rt_panic("rt_get_pid: process identifier does not fit in l0_int");
01631     }
01632 #else
01633     if (!_rt_pid_to_l0_int((intmax_t)getpid(), &out)) {
01634         _rt_panic("rt_get_pid: process identifier does not fit in l0_int");
01635     }
01636 #endif
01637     return out;
01638 }
01639
01649 static l0_string rt_get_argv(l0_int i) {
01650     if (i < 0 || i >= _rt_argc) {
01651         _rt_panic_fmt("rt_get_argv: index %d out of bounds (argc=%d)", (int)i, _rt_argc);
01652     }
01653     return _rt_l0_string_from_const_literal(_rt_argv[i]);
01654 }
01655
01656 /* ============================================================================
01657  * Time APIs
01658  * ============================================================================ */
01659
01663 static l0_bool _rt_time_to_l0_int_sec(time_t value, l0_int *out) {
01664     long long sec = (long long)value;
01665     if (sec < INT32_MIN || sec > INT32_MAX) {
01666         return 0;
01667     }
01668     *out = (l0_int)sec;
01669     return 1;
01670 }
01671
01675 static l0_bool _rt_time_to_l0_int_nsec(long value, l0_int *out) {
01676     long long nsec = (long long)value;
01677     if (nsec < 0 || nsec > 999999999LL) {
01678         return 0;
01679     }
01680     *out = (l0_int)nsec;
01681     return 1;
01682 }
01683
01687 static l0_bool _rt_time_write_parts(struct l0_sys_rt_RtTimeParts *out, l0_int sec, l0_int nsec) {
01688     if (out == NULL) {
01689         _rt_panic("_rt_time_write_parts: out-parameter is null");
01690     }
01691     out->sec = sec;
01692     out->nsec = nsec;
01693     return 1;
01694 }
01695
01704 static l0_bool rt_time_unix(struct l0_sys_rt_RtTimeParts *out) {
01705     if (out == NULL) {
01706         _rt_panic("rt_time_unix: out-parameter is null");
01707     }
01708
```

```
01709 #if defined(CLOCK_REALTIME)
01710     struct timespec ts;
01711     if (clock_gettime(CLOCK_REALTIME, &ts) == 0) {
01712         l0_int sec = 0;
01713         l0_int nsec = 0;
01714         if (!_rt_time_to_l0_int_sec(ts.tv_sec, &sec)) {
01715             return 0;
01716         }
01717         if (!_rt_time_to_l0_int_nsec(ts.tv_nsec, &nsec)) {
01718             return 0;
01719         }
01720         return _rt_time_write_parts(out, sec, nsec);
01721     }
01722 #endif
01723
01724     time_t now = time(NULL);
01725     if (now == (time_t)-1) {
01726         return 0;
01727     }
01728
01729     l0_int sec = 0;
01730     if (!_rt_time_to_l0_int_sec(now, &sec)) {
01731         return 0;
01732     }
01733     return _rt_time_write_parts(out, sec, 0);
01734 }
01735
01744 static l0_bool rt_time_monotonic(struct l0_sys_rt_RtTimeParts *out) {
01745     if (out == NULL) {
01746         _rt_panic("rt_time_monotonic: out-parameter is null");
01747     }
01748
01749 #if defined(CLOCK_MONOTONIC)
01750     struct timespec ts;
01751     if (clock_gettime(CLOCK_MONOTONIC, &ts) != 0) {
01752         return 0;
01753     }
01754
01755     l0_int sec = 0;
01756     l0_int nsec = 0;
01757     if (!_rt_time_to_l0_int_sec(ts.tv_sec, &sec)) {
01758         return 0;
01759     }
01760     if (!_rt_time_to_l0_int_nsec(ts.tv_nsec, &nsec)) {
01761         return 0;
01762     }
01763     return _rt_time_write_parts(out, sec, nsec);
01764 #else
01765     (void)out;
01766     return 0;
01767 #endif
01768 }
01769
01777 static l0_bool rt_time_monotonic_supported(void) {
01778 #if defined(CLOCK_MONOTONIC)
01779     return 1;
01780 #else
01781     return 0;
01782 #endif
01783 }
01784
01796 static l0_opt_int rt_time_local_offset_sec(l0_int unix_sec) {
01797     time_t t = (time_t)unix_sec;
01798     if ((l0_int)t != unix_sec) {
01799         return (l0_opt_int){ .has_value = 0 };
01800     }
01801
01802     struct tm *utc_ptr = gmtime(&t);
01803     if (utc_ptr == NULL) {
01804         return (l0_opt_int){ .has_value = 0 };
01805     }
01806     struct tm utc_tm = *utc_ptr;
01807
01808     struct tm *local_ptr = localtime(&t);
01809     if (local_ptr == NULL) {
01810         return (l0_opt_int){ .has_value = 0 };
01811     }
01812     struct tm local_tm = *local_ptr;
01813
01814     /* Day difference: can only be -1, 0, or +1 for timezone offsets. */
01815     int day_diff;
01816     if (local_tm.tm_year > utc_tm.tm_year) {
01817         day_diff = 1;
01818     } else if (local_tm.tm_year < utc_tm.tm_year) {
01819         day_diff = -1;
01820     } else {
01821         day_diff = local_tm.tm_yday - utc_tm.tm_yday;
```

```
01822     }
01823
01824     long long offset = (long long)day_diff * 86400
01825                      + (long long)(local_tm.tm_hour - utc_tm.tm_hour) * 3600
01826                      + (long long)(local_tm.tm_min - utc_tm.tm_min) * 60
01827                      + (long long)(local_tm.tm_sec - utc_tm.tm_sec);
01828     if (offset < INT32_MIN || offset > INT32_MAX) {
01829         return (l0_opt_int){ .has_value = 0 };
01830     }
01831
01832     return (l0_opt_int){ .has_value = 1, .value = (l0_int)offset };
01833 }
01834
01843 static l0_opt_bool rt_time_local_is_dst(l0_int unix_sec) {
01844     time_t t = (time_t)unix_sec;
01845     if ((l0_int)t != unix_sec) {
01846         return (l0_opt_bool){ .has_value = 0 };
01847     }
01848
01849     struct tm *local_ptr = localtime(&t);
01850     if (local_ptr == NULL) {
01851         return (l0_opt_bool){ .has_value = 0 };
01852     }
01853
01854     if (local_ptr->tm_isdst < 0) {
01855         return (l0_opt_bool){ .has_value = 0 };
01856     }
01857     return (l0_opt_bool){ .has_value = 1, .value = local_ptr->tm_isdst > 0 ? 1 : 0 };
01858 }
01859
01860 /* ============================================================================
01861  * I/O operations (whole-file)
01862  * ============================================================================ */
01863
01873 static l0_opt_string rt_read_file_all(l0_string path) {
01874
01875     l0_int path_len = rt_strlen(path);
01876
01877     if (path_len == 0) {
01878         return L0_OPT_STRING_NULL;
01879     }
01880
01881     char *path_cstr = _rt_string_bytes(path);
01882     struct stat st;
01883
01884     if (stat(path_cstr, &st) != 0) {
01885         return L0_OPT_STRING_NULL;
01886     }
01887     if (!S_ISREG(st.st_mode)) {
01888         return L0_OPT_STRING_NULL;
01889     }
01890     if (st.st_size < 0 || (uint64_t)st.st_size > INT32_MAX) {
01891         return L0_OPT_STRING_NULL;
01892     }
01893
01894     FILE *file = fopen(path_cstr, "rb");
01895     if (file == NULL) {
01896         return L0_OPT_STRING_NULL;
01897     }
01898
01899     size_t size = (size_t)st.st_size;
01900
01901     l0_string result = _rt_alloc_string((l0_int)size);
01902     char *buffer = _rt_string_bytes(result);
01903
01904     /* Read file contents */
01905     size_t bytes_read = fread(buffer, 1, size, file);
01906     fclose(file);
01907
01908     if (bytes_read != size) {
01909         _rt_free_string(result);
01910         return L0_OPT_STRING_NULL;
01911     }
01912
01913     return (l0_opt_string){ .has_value = 1, .value = result };
01914 }
01915
01926 static l0_bool rt_write_file_all(l0_string path, l0_string data) {
01927     l0_int path_len = rt_strlen(path);
01928     if (path_len == 0) {
01929         return 0;
01930     }
01931
01932     /* Ensure path is null-terminated for fopen */
01933     char *path_cstr = _rt_string_bytes(path);
01934     FILE *file = fopen(path_cstr, "wb");
01935     if (file == NULL) {
```

```
01936         return 0;
01937     }
01938
01939     l0_int data_len = rt_strlen(data);
01940     size_t written = 0;
01941     if (data_len > 0) {
01942         char *data_b = _rt_string_bytes(data);
01943         written = fwrite(data_b, 1, (size_t)data_len, file);
01944     }
01945     int close_result = DEA_RUNTIME_FCLOSE(file);
01946
01947     return written == (size_t)data_len && close_result == 0;
01948 }
01949
01958 static struct l0_sys_rt_RtFileInfo rt_file_info(l0_string path) {
01959     struct l0_sys_rt_RtFileInfo out = {
01960         .exists = 0,
01961         .is_file = 0,
01962         .is_dir = 0,
01963         .size = { .has_value = 0 },
01964         .mtime_sec = { .has_value = 0 },
01965         .mtime_nsec = { .has_value = 0 },
01966     };
01967     if (rt_strlen(path) == 0) {
01968         return out;
01969     }
01970     char *c = _rt_string_bytes(path);
01971 #if defined(_WIN32)
01972     struct _stat64 st;
01973     if (DEA_RUNTIME_FILE_INFO_STAT(c, &st) != 0) {
01974         return out;
01975     }
01976
01977     out.exists = 1;
01978     out.is_file = (st.st_mode & _S_IFREG) ? 1 : 0;
01979     out.is_dir = (st.st_mode & _S_IFDIR) ? 1 : 0;
01980
01981     if (st.st_size >= 0 && (__int64)(l0_int)st.st_size == st.st_size) {
01982         out.size = (l0_opt_int){ .has_value = 1, .value = (l0_int)st.st_size };
01983     }
01984     if ((time_t)(l0_int)st.st_mtime == st.st_mtime) {
01985         out.mtime_sec = (l0_opt_int){ .has_value = 1, .value = (l0_int)st.st_mtime };
01986     }
01987     return out;
01988 #else
01989     struct stat st;
01990     if (DEA_RUNTIME_FILE_INFO_STAT(c, &st) != 0) {
01991         return out;
01992     }
01993
01994     out.exists = 1;
01995     out.is_file = S_ISREG(st.st_mode) ? 1 : 0;
01996     out.is_dir = S_ISDIR(st.st_mode) ? 1 : 0;
01997
01998     if (st.st_size >= 0 && (off_t)(l0_int)st.st_size == st.st_size) {
01999         out.size = (l0_opt_int){ .has_value = 1, .value = (l0_int)st.st_size };
02000     }
02001     if ((time_t)(l0_int)st.st_mtime == st.st_mtime) {
02002         out.mtime_sec = (l0_opt_int){ .has_value = 1, .value = (l0_int)st.st_mtime };
02003 #if defined(__APPLE__)
02004         if ((long)(l0_int)st.st_mtimespec.tv_nsec == st.st_mtimespec.tv_nsec) {
02005             out.mtime_nsec = (l0_opt_int){ .has_value = 1, .value = (l0_int)st.st_mtimespec.tv_nsec };
02006         }
02007 #elif defined(_POSIX_C_SOURCE) && _POSIX_C_SOURCE >= 200809L
02008         if ((long)(l0_int)st.st_mtim.tv_nsec == st.st_mtim.tv_nsec) {
02009             out.mtime_nsec = (l0_opt_int){ .has_value = 1, .value = (l0_int)st.st_mtim.tv_nsec };
02010         }
02011 #endif
02012     }
02013     return out;
02014 #endif
02015 }
02016
02026 static l0_bool rt_delete_file(l0_string path) {
02027     if (rt_strlen(path) == 0) {
02028         return 0;
02029     }
02030     char *c = _rt_string_bytes(path);
02031     int result = DEA_RUNTIME_REMOVE(c);
02032     return result == 0;
02033 }
02034
02043 static l0_int _rt_stream_write_some(FILE *stream, const l0_byte *buf, l0_int len) {
02044     if (len < 0) {
02045         return -1;
02046     }
02047     if (len == 0) {
```

```
02048         return 0;
02049     }
02050     if (buf == NULL) {
02051         return -1;
02052     }
02053
02054     clearerr(stream);
02055     size_t written = fwrite(buf, 1, (size_t)len, stream);
02056     if (written == 0 && ferror(stream)) {
02057         return -1;
02058     }
02059     return (l0_int)written;
02060 }
02061
02072 static l0_int rt_stdin_read(l0_byte *buf, l0_int capacity) {
02073     if (capacity < 0) {
02074         return -1;
02075     }
02076     if (capacity == 0) {
02077         return 0;
02078     }
02079     if (buf == NULL) {
02080         return -1;
02081     }
02082
02083     clearerr(stdin);
02084     size_t nread = fread(buf, 1, (size_t)capacity, stdin);
02085     if (nread == 0 && ferror(stdin)) {
02086         return -1;
02087     }
02088     return (l0_int)nread;
02089 }
02090
02101 static l0_int rt_stdout_write(l0_byte *buf, l0_int len) {
02102     return _rt_stream_write_some(stdout, buf, len);
02103 }
02104
02115 static l0_int rt_stderr_write(l0_byte *buf, l0_int len) {
02116     return _rt_stream_write_some(stderr, buf, len);
02117 }
02118
02119 /* ============================================================================
02120  * Printing to stdout/stderr
02121  * ============================================================================ */
02122
02125 static void rt_flush_stdout(void) {
02126     fflush(stdout);
02127 }
02128
02137 static void rt_flush_stderr(void) {
02138     fflush(stderr);
02139 }
02140
02147 void _rt_print(l0_string s, FILE *stream){
02148     l0_int s_len = rt_strlen(s);
02149     char *s_data = _rt_string_bytes(s);
02150     if (s_len > 0 && s_data != NULL) {
02151         fwrite(s_data, 1, (size_t)s_len, stream);
02152     }
02153 }
02154
02162 static void rt_print(l0_string s) {
02163     _rt_print(s, stdout);
02164 }
02165
02173 static void rt_print_stderr(l0_string s) {
02174     _rt_print(s, stderr);
02175 }
02176
02179 static void rt_println(void) {
02180     fputc('\n', stdout);
02181 }
02182
02191 static void rt_println_stderr(void) {
02192     fputc('\n', stderr);
02193 }
02194
02202 static void rt_print_int(l0_int x) {
02203     printf("%d", (int)x);
02204 }
02205
02213 static void rt_print_int_stderr(l0_int x) {
02214     fprintf(stderr, "%d", (int)x);
02215 }
02216
02224 static void rt_print_bool(l0_bool x) {
02225     printf("%s", x ? "true" : "false");
```

```
02226 }
02227
02235 static void rt_print_bool_stderr(l0_bool x) {
02236     fprintf(stderr, "%s", x ? "true" : "false");
02237 }
02238
02239 /* ============================================================================
02240  * Reading from stdin
02241  * ============================================================================ */
02242
02254 static l0_opt_string rt_read_line(void) {
02255     size_t capacity = 128;
02256     size_t length = 0;
02257
02258     l0_string s = _rt_alloc_string(capacity);
02259     char *s_data = _rt_string_bytes(s);
02260
02261     int c;
02262     while ((c = fgetc(stdin)) != EOF && c != '\n') {
02263         if (length + 1 >= capacity) {
02264             capacity = capacity * 2;
02265             s = _rt_realloc_string(s, (l0_int)capacity);
02266             s_data = _rt_string_bytes(s);
02267         }
02268         s_data[length++] = (char)c;
02269     }
02270
02271     /* EOF with no data ⇒ None */
02272     if (c == EOF && length == 0) {
02273         _rt_free_string(s);
02274         return L0_OPT_STRING_NULL;
02275     }
02276
02277     if (length > INT32_MAX) {
02278         _rt_free_string(s);
02279         _rt_panic("rt_read_line: line too long for l0_int");
02280     }
02281
02282     /* Empty line ⇒ Some(empty string) without allocating owned storage. */
02283     if (length == 0) {
02284         _rt_free_string(s);
02285         return L0_OPT_STRING_EMPTY;
02286     }
02287
02288     /* Trim string to actual length */
02289     if ((size_t)length < capacity) {
02290         s = _rt_realloc_string(s, (l0_int)length);
02291     }
02292
02293     return (l0_opt_string){ .has_value = 1, .value = s };
02294 }
02295
02296
02305 static l0_int rt_read_char(void) {
02306     int c = fgetc(stdin);
02307     if (c == EOF) {
02308         return -1;
02309     }
02310     return (l0_int)c;
02311 }
02312
02313 /* ============================================================================
02314  * Other runtime utilities
02315  * ============================================================================ */
02316
02324 static void rt_abort(l0_string message) {
02325     if (rt_strlen(message) == 0) {
02326         _rt_panic(NULL);
02327     } else {
02328         _rt_panic_fmt("%s", _rt_string_bytes(message));
02329     }
02330     abort();
02331 }
02332
02340 static void rt_exit(l0_int code) {
02341     exit((int)code);
02342 }
02343
02344 /* ============================================================================
02345  * Random number generation
02346  * ============================================================================ */
02347
02356 static void rt_srand(l0_int seed) {
02357     if (seed == 0) {
02358         srand((unsigned int)time(NULL));
02359     } else {
02360         srand((unsigned int)seed);
```

```
02361     }
02362 }
02363
02373 static l0_int rt_rand(l0_int max) {
02374     if (max <= 0) {
02375         return 0;
02376     }
02377     return (l0_int)(rand() % max);
02378 }
02379
02387 static l0_int rt_errno(void) {
02388     return (l0_int)errno;
02389 }
02390
02391 /* ============================================================================
02392  * UNSAFE ZONE: HERE BE DRAGONS
02393  * ----------------------------------------------------------------------------
02394  * This section contains functions that directly manipulate memory.
02395  * Use with caution - these functions do not perform safety checks beyond basic
02396  * validation of input parameters.
02397  * They are intended for low-level operations where performance is critical.
02398  * Misuse can lead to undefined behavior, memory corruption, or security
02399  * vulnerabilities.
02400  * ============================================================================ */
02401
02402 /* ============================================================================
02403  * Memory allocation and manipulation functions.
02404  * ============================================================================ */
02405
02416 #ifdef L0_TRACE_MEMORY
02417 static void *_rt_alloc_impl(l0_int bytes, const char *_loc_file, int _loc_line) {
02418     /* zero-size allocations are not allowed */
02419     if (bytes <= 0) {
02420         _rt_panic("rt_alloc: invalid allocation size");
02421     }
02422
02423     /* Check for overflow when converting to size_t */
02424     if ((uint64_t)bytes > SIZE_MAX) {
02425         _rt_panic_fmt("rt_alloc: allocation size overflow (%d bytes requested)", (int)bytes);
02426     }
02427
02428     size_t size = (size_t)bytes;
02429     void *ptr = malloc(size);
02430
02431     if (ptr == NULL) {
02432         /* Allocation failed - return NULL and let caller handle it */
02433         _RT_TRACE_MEM("op=alloc bytes=%d ptr=%p action=fail loc=\"%s\":%d", (int)bytes, (void*)ptr,
      _loc_file, _loc_line);
02434         return NULL;
02435     }
02436
02437     _rt_track_alloc_record(ptr, size, 0, 0, _loc_file, _loc_line);
02438     _RT_TRACE_MEM("op=alloc bytes=%d ptr=%p action=ok loc=\"%s\":%d", (int)bytes, ptr, _loc_file,
      _loc_line);
02439     return ptr;
02440 }
02441 #define rt_alloc(bytes) _rt_alloc_impl((bytes), __FILE__, __LINE__)
02442 #else
02443 static void *rt_alloc(l0_int bytes) {
02444     /* zero-size allocations are not allowed */
02445     if (bytes <= 0) {
02446         _rt_panic("rt_alloc: invalid allocation size");
02447     }
02448
02449     /* Check for overflow when converting to size_t */
02450     if ((uint64_t)bytes > SIZE_MAX) {
02451         _rt_panic_fmt("rt_alloc: allocation size overflow (%d bytes requested)", (int)bytes);
02452     }
02453
02454     size_t size = (size_t)bytes;
02455     void *ptr = malloc(size);
02456
02457     if (ptr == NULL) {
02458         /* Allocation failed - return NULL and let caller handle it */
02459         _RT_TRACE_MEM("op=alloc bytes=%d ptr=%p action=fail", (int)bytes, (void*)ptr);
02460         return NULL;
02461     }
02462
02463     _rt_track_alloc_record(ptr, size, 0, 0, "<runtime>", 0);
02464     _RT_TRACE_MEM("op=alloc bytes=%d ptr=%p action=ok", (int)bytes, ptr);
02465     return ptr;
02466 }
02467 #endif
02468
02481 #ifdef L0_TRACE_MEMORY
02482 static void *_rt_realloc_impl(void *ptr, l0_int new_bytes, const char *_loc_file, int _loc_line) {
02483     /* zero-size allocations are not allowed */
```

```
02484     if (new_bytes <= 0) {
02485         _rt_panic("rt_realloc: invalid allocation size");
02486     }
02487
02488     if ((uint64_t)new_bytes > SIZE_MAX) {
02489         _rt_panic_fmt("rt_realloc: allocation size overflow (%d bytes requested)", (int)new_bytes);
02490     }
02491
02492     volatile uintptr_t old_ptr_addr = (uintptr_t)ptr;
02493     size_t new_size = (size_t)new_bytes;
02494     void *new_ptr = _rt_realloc_tracked_alloc(ptr, new_size, _loc_file, _loc_line);
02495
02496     if (new_ptr == NULL) {
02497         /* Real failure! original pointer is still valid */
02498         _RT_TRACE_MEM("op=realloc old_ptr=%p bytes=%d new_ptr=%p action=fail loc=\"%s\":%d",
      (void*)old_ptr_addr, (int)new_bytes, (void*)new_ptr, _loc_file, _loc_line);
02499         return NULL;
02500     }
02501
02502     _RT_TRACE_MEM("op=realloc old_ptr=%p bytes=%d new_ptr=%p action=ok loc=\"%s\":%d",
      (void*)old_ptr_addr, (int)new_bytes, new_ptr, _loc_file, _loc_line);
02503     return new_ptr;
02504 }
02505 #define rt_realloc(ptr, new_bytes) _rt_realloc_impl((ptr), (new_bytes), __FILE__, __LINE__)
02506 #else
02507 static void *rt_realloc(void *ptr, l0_int new_bytes) {
02508     /* zero-size allocations are not allowed */
02509     if (new_bytes <= 0) {
02510         _rt_panic("rt_realloc: invalid allocation size");
02511     }
02512
02513     if ((uint64_t)new_bytes > SIZE_MAX) {
02514         _rt_panic_fmt("rt_realloc: allocation size overflow (%d bytes requested)", (int)new_bytes);
02515     }
02516
02517     size_t new_size = (size_t)new_bytes;
02518     void *new_ptr = _rt_realloc_tracked_alloc(ptr, new_size, "<runtime>", 0);
02519
02520     if (new_ptr == NULL) {
02521         /* Real failure! original pointer is still valid */
02522         _RT_TRACE_MEM("op=realloc old_ptr=%p bytes=%d new_ptr=%p action=fail", ptr, (int)new_bytes,
      (void*)new_ptr);
02523         return NULL;
02524     }
02525
02526     _RT_TRACE_MEM("op=realloc old_ptr=%p bytes=%d new_ptr=%p action=ok", ptr, (int)new_bytes,
      new_ptr);
02527     return new_ptr;
02528 }
02529 #endif
02530
02538 #ifdef L0_TRACE_MEMORY
02539 static void _rt_free_impl(void *ptr, const char *_loc_file, int _loc_line) {
02540     /* free(NULL) is a no-op in C */
02541     _RT_TRACE_MEM("op=free ptr=%p action=call loc=\"%s\":%d", ptr, _loc_file, _loc_line);
02542     _rt_release_tracked_alloc(ptr, _loc_file, _loc_line, "rt_free");
02543 }
02544 #define rt_free(ptr) _rt_free_impl((ptr), __FILE__, __LINE__)
02545 #else
02546 static void rt_free(void *ptr) {
02547     /* free(NULL) is a no-op in C */
02548     _RT_TRACE_MEM("op=free ptr=%p action=call", ptr);
02549     _rt_release_tracked_alloc(ptr, "<runtime>", 0, "rt_free");
02550 }
02551 #endif
02552
02563 #ifdef L0_TRACE_MEMORY
02564 static void *_rt_calloc_impl(l0_int count, l0_int elem_size, const char *_loc_file, int _loc_line) {
02565     if (count <= 0 || elem_size <= 0) {
02566         _rt_panic("rt_calloc: invalid count or element size");
02567     }
02568
02569     /* Check for overflow in multiplication*/
02570     if ((uint64_t)count * (uint64_t)elem_size > SIZE_MAX) {
02571         _rt_panic_fmt("rt_calloc: allocation size overflow (%d elements of size %d requested)",
02572                       (int)count, (int)elem_size);
02573     }
02574
02575     size_t n = (size_t)count;
02576     size_t size = (size_t)elem_size;
02577
02578     void *ptr = calloc(n, size);
02579     if (ptr != NULL) {
02580         _rt_track_alloc_record(ptr, n * size, 0, 0, _loc_file, _loc_line);
02581     }
02582     _RT_TRACE_MEM(
02583         "op=calloc count=%d elem_size=%d ptr=%p action=%s loc=\"%s\":%d",
```

```
02584         (int)count, (int)elem_size, ptr, ptr == NULL ? "fail" : "ok", _loc_file, _loc_line
02585     );
02586     return ptr;
02587 }
02588 #define rt_calloc(count, elem_size) _rt_calloc_impl((count), (elem_size), __FILE__, __LINE__)
02589 #else
02590 static void *rt_calloc(l0_int count, l0_int elem_size) {
02591     if (count <= 0 || elem_size <= 0) {
02592         _rt_panic("rt_calloc: invalid count or element size");
02593     }
02594
02595     /* Check for overflow in multiplication*/
02596     if ((uint64_t)count * (uint64_t)elem_size > SIZE_MAX) {
02597         _rt_panic_fmt("rt_calloc: allocation size overflow (%d elements of size %d requested)",
02598                       (int)count, (int)elem_size);
02599     }
02600
02601     size_t n = (size_t)count;
02602     size_t size = (size_t)elem_size;
02603
02604     void *ptr = calloc(n, size);
02605     if (ptr != NULL) {
02606         _rt_track_alloc_record(ptr, n * size, 0, 0, "<runtime>", 0);
02607     }
02608     _RT_TRACE_MEM(
02609         "op=calloc count=%d elem_size=%d ptr=%p action=%s",
02610         (int)count, (int)elem_size, ptr, ptr == NULL ? "fail" : "ok"
02611     );
02612     return ptr;
02613 }
02614 #endif
02615
02627 static void *rt_memset(void *dest, l0_int value, l0_int bytes) {
02628     if (bytes < 0) {
02629         _rt_panic("rt_memset: negative byte count");
02630     }
02631
02632     if (bytes == 0 || dest == NULL) {
02633         return dest;
02634     }
02635
02636     size_t n = (size_t)bytes;
02637     int c = (int)value;
02638     return memset(dest, c, n);
02639 }
02640
02652 static void *rt_memcpy(void *dest, void *src, l0_int bytes) {
02653     if (bytes < 0) {
02654         _rt_panic("rt_memcpy: negative byte count");
02655     }
02656
02657     if (bytes == 0 || dest == NULL || src == NULL) {
02658         return dest;
02659     }
02660
02661     size_t n = (size_t)bytes;
02662     return memcpy(dest, src, n);
02663 }
02664
02676 static l0_int rt_memcmp(void *a, void *b, l0_int bytes) {
02677     if (bytes < 0) {
02678         _rt_panic("rt_memcmp: negative byte count");
02679     }
02680
02681     if (bytes == 0 || a == NULL || b == NULL) {
02682         return 0;
02683     }
02684
02685     size_t n = (size_t)bytes;
02686     int result = memcmp(a, b, n);
02687     if (result < 0) {
02688         return -1;
02689     } else if (result > 0) {
02690         return 1;
02691     } else {
02692         return 0;
02693     }
02694 }
02695
02707 static void *rt_array_element(void *array_data, l0_int element_size, l0_int index) {
02708     if (array_data == NULL) {
02709         _rt_panic("rt_array_element: null array data pointer");
02710     }
02711     if (element_size <= 0) {
02712         _rt_panic("rt_array_element: invalid element size");
02713     }
02714     if (index < 0) {
```

```
02715         _rt_panic("rt_array_element: negative index");
02716     }
02717
02718     /* Check for overflow in multiplication */
02719     if ((uint64_t)index * (uint64_t)element_size > SIZE_MAX) {
02720         _rt_panic_fmt("rt_array_element: index * element_size overflow (%d * %d)",
02721                       (int)index, (int)element_size);
02722     }
02723
02724     /* One shared entry point serving every container call site: a single
02725      * static site slot would thrash across unrelated allocations, so skip
02726      * the per-call-site cache and always take the full lookup. Read mode is
02727      * used even for element stores: callers only pass container storage,
02728      * which is runtime-owned writable raw memory, so read-only enforcement
02729      * never applies here. */
02730     return _rt_check_index_ptr_site(
02731         NULL,
02732         array_data,
02733         index,
02734         element_size,
02735         0,
02736         _RT_ACCESS_READ,
02737         __FILE__,
02738         __LINE__
02739     );
02740 }
02741
02742 /* ============================================================================
02743  * End of UNSAFE ZONE
02744  * ============================================================================ */
02745
02746 /* ============================================================================
02747  * Runtime support for `new` & `drop`: allocation tracker implementation
02748  * ----------------------------------------------------------------------------
02749  * Base-pointer lookup uses an open-addressing hash table (O(1) amortized).
02750  * Interior pointers resolve through an address-ordered treap keyed by
02751  * allocation base (O(log n) insert/remove/lookup, no bulk moves). Records
02752  * come from a never-freed pool so call-site caches may keep record pointers
02753  * across frees; the generation counter invalidates recycled records.
02754  * ============================================================================ */
02755
02756 #ifndef L0_RT_UNCHECKED
02757
02758 #define _RT_ALLOC_TOMBSTONE ((_rt_alloc_record*)(uintptr_t)1)
02759
02760 static _rt_alloc_record **_rt_alloc_table = NULL;
02761 static size_t _rt_alloc_table_cap = 0;
02762 static size_t _rt_alloc_table_cnt = 0;
02763 static size_t _rt_alloc_table_tombstones = 0;
02764 static uint64_t _rt_alloc_next_generation = 1;
02765
02766 #ifndef L0_RT_CHECK_BASIC
02767 static _rt_alloc_record *_rt_alloc_tree_root = NULL;
02768 #endif
02769 static _rt_alloc_record *_rt_rec_free_list = NULL;
02770 static _rt_alloc_record_cold **_rt_cold_chunks = NULL;
02771 static size_t _rt_rec_pool_chunks = 0;
02772
02773 static _rt_alloc_record *_rt_quarantine_head = NULL;
02774 static _rt_alloc_record *_rt_quarantine_tail = NULL;
02775 static size_t _rt_quarantine_bytes = 0;
02776 static size_t _rt_quarantine_count = 0;
02777
02778 #if _RT_REC_POOL_CHUNK == 256
02779 #define _rt_rec_cold(rec) (&_rt_cold_chunks[(rec)->cold_index » 8][(rec)->cold_index & 255u])
02780 #else
02781 #define _rt_rec_cold(rec) (&_rt_cold_chunks[(size_t)(rec)->cold_index /
      (size_t)_RT_REC_POOL_CHUNK][(size_t)(rec)->cold_index % (size_t)_RT_REC_POOL_CHUNK])
02782 #endif
02783
02789 static _rt_alloc_record *_rt_rec_new(_rt_alloc_record_cold **cold_out) {
02790     if (_rt_rec_free_list == NULL) {
02791         size_t chunk_count = (size_t)_RT_REC_POOL_CHUNK;
02792         size_t chunk_no = _rt_rec_pool_chunks;
02793         if (chunk_count == 0) {
02794             _rt_panic("runtime allocation tracker: invalid record pool chunk size");
02795         }
02796         if (chunk_count > (SIZE_MAX - 63u) / sizeof(_rt_alloc_record)) {
02797             _rt_panic("runtime allocation tracker: record pool size overflow");
02798         }
02799         if (chunk_count > SIZE_MAX / sizeof(_rt_alloc_record_cold)) {
02800             _rt_panic("runtime allocation tracker: cold record pool size overflow");
02801         }
02802         if (chunk_no > ((size_t)UINT32_MAX - (chunk_count - 1u)) / chunk_count) {
02803             _rt_panic("runtime allocation tracker: record pool index overflow");
02804         }
02805         if (chunk_no > (SIZE_MAX / sizeof(_rt_alloc_record_cold*)) - 1u) {
```

```
02806             _rt_panic("runtime allocation tracker: cold chunk directory overflow");
02807         }
02808
02809         size_t hot_bytes = chunk_count * sizeof(_rt_alloc_record);
02810         unsigned char *raw = (unsigned char*)malloc(hot_bytes + 63u);
02811         if (raw == NULL) {
02812             _rt_panic("runtime allocation tracker: out of memory (hot record pool)");
02813         }
02814         size_t off = (size_t)(-(uintptr_t)raw & 63u);
02815         _rt_alloc_record *chunk = (_rt_alloc_record*)(void*)(raw + off);
02816
02817         _rt_alloc_record_cold **new_cold_chunks = (_rt_alloc_record_cold**)realloc(
02818             _rt_cold_chunks,
02819             (chunk_no + 1u) * sizeof(_rt_alloc_record_cold*)
02820         );
02821         if (new_cold_chunks == NULL) {
02822             _rt_panic("runtime allocation tracker: out of memory (cold chunk directory)");
02823         }
02824         _rt_cold_chunks = new_cold_chunks;
02825
02826         _rt_alloc_record_cold *cold_chunk = (_rt_alloc_record_cold*)malloc(chunk_count *
      sizeof(_rt_alloc_record_cold));
02827         if (cold_chunk == NULL) {
02828             _rt_panic("runtime allocation tracker: out of memory (cold record pool)");
02829         }
02830         _rt_cold_chunks[chunk_no] = cold_chunk;
02831         _rt_rec_pool_chunks++;
02832
02833         size_t base_index = chunk_no * chunk_count;
02834         for (size_t i = 0; i < chunk_count; i++) {
02835             chunk[i].base = NULL;
02836             chunk[i].size = 0;
02837             chunk[i].state = _RT_ALLOC_POOLED;
02838             chunk[i].read_only = 0;
02839             chunk[i].mem_kind = 0;
02840             chunk[i].generation = 0;
02841             chunk[i].cold_index = (uint32_t)(base_index + i);
02842             chunk[i].q_next = _rt_rec_free_list;
02843             _rt_rec_free_list = &chunk[i];
02844         }
02845     }
02846     _rt_alloc_record *rec = _rt_rec_free_list;
02847     _rt_alloc_record_cold *cold = _rt_rec_cold(rec);
02848     _rt_rec_free_list = rec->q_next;
02849     if (cold_out != NULL) {
02850         *cold_out = cold;
02851     }
02852     return rec;
02853 }
02854
02855 static void _rt_rec_recycle(_rt_alloc_record *rec) {
02856     rec->state = _RT_ALLOC_POOLED;
02857     rec->generation = 0;
02858     rec->q_next = _rt_rec_free_list;
02859     _rt_rec_free_list = rec;
02860 }
02861
02862 static inline size_t _rt_alloc_hash(void *ptr, size_t cap) {
02863     uint64_t v = (uint64_t)(uintptr_t)ptr;
02864     uint32_t x = (uint32_t)(v ^ (v » 32));
02865     x ^= x » 16; x *= 0x85ebca6bu;
02866     x ^= x » 13; x *= 0xc2b2ae35u;
02867     x ^= x » 16;
02868     return (size_t)(x & (uint32_t)(cap - 1));
02869 }
02870
02878 static void _rt_alloc_table_rehash(void) {
02879     size_t old_cap = _rt_alloc_table_cap;
02880     _rt_alloc_record **old_tbl = _rt_alloc_table;
02881     size_t new_cap = _RT_ALLOC_INIT_CAP;
02882     while (new_cap < _rt_alloc_table_cnt * 2) {
02883         new_cap *= 2;
02884     }
02885
02886     _rt_alloc_record **new_tbl = (_rt_alloc_record**)calloc(new_cap, sizeof(_rt_alloc_record*));
02887     if (new_tbl == NULL) {
02888         _rt_panic("new: out of memory (alloc tracker rehash)");
02889     }
02890
02891     for (size_t i = 0; i < old_cap; i++) {
02892         _rt_alloc_record *rec = old_tbl[i];
02893         if (rec != NULL && rec != _RT_ALLOC_TOMBSTONE) {
02894             size_t idx = _rt_alloc_hash(rec->base, new_cap);
02895             while (new_tbl[idx] != NULL) {
02896                 idx = (idx + 1) & (new_cap - 1);
02897             }
02898             new_tbl[idx] = rec;
```

```
02899         }
02900     }
02901
02902     free(old_tbl);
02903     _rt_alloc_table     = new_tbl;
02904     _rt_alloc_table_cap = new_cap;
02905     _rt_alloc_table_tombstones = 0;
02906 }
02907
02908 static _rt_alloc_record *_rt_alloc_table_lookup(void *ptr) {
02909     if (_rt_alloc_table_cap == 0 || ptr == NULL) return NULL;
02910
02911     size_t idx = _rt_alloc_hash(ptr, _rt_alloc_table_cap);
02912     for (size_t probed = 0; probed < _rt_alloc_table_cap && _rt_alloc_table[idx] != NULL; probed++) {
02913         _rt_alloc_record *rec = _rt_alloc_table[idx];
02914         if (rec != _RT_ALLOC_TOMBSTONE && rec->base == ptr) {
02915             return rec;
02916         }
02917         idx = (idx + 1) & (_rt_alloc_table_cap - 1);
02918     }
02919     return NULL;
02920 }
02921
02922 static void _rt_alloc_table_insert(_rt_alloc_record *rec) {
02923     if (_rt_alloc_table_cap == 0 ||
02924         (_rt_alloc_table_cnt + _rt_alloc_table_tombstones + 1) * 10 > _rt_alloc_table_cap * 7) {
02925         _rt_alloc_table_rehash();
02926     }
02927
02928     size_t idx = _rt_alloc_hash(rec->base, _rt_alloc_table_cap);
02929     while (_rt_alloc_table[idx] != NULL &&
02930            _rt_alloc_table[idx] != _RT_ALLOC_TOMBSTONE) {
02931         idx = (idx + 1) & (_rt_alloc_table_cap - 1);
02932     }
02933     if (_rt_alloc_table[idx] == _RT_ALLOC_TOMBSTONE) {
02934         _rt_alloc_table_tombstones--;
02935     }
02936     _rt_alloc_table[idx] = rec;
02937     _rt_alloc_table_cnt++;
02938 }
02939
02940 static void _rt_alloc_table_remove(_rt_alloc_record *target) {
02941     if (_rt_alloc_table_cap == 0) return;
02942
02943     size_t idx = _rt_alloc_hash(target->base, _rt_alloc_table_cap);
02944     for (size_t probed = 0; probed < _rt_alloc_table_cap && _rt_alloc_table[idx] != NULL; probed++) {
02945         if (_rt_alloc_table[idx] == target) {
02946             _rt_alloc_table[idx] = _RT_ALLOC_TOMBSTONE;
02947             _rt_alloc_table_cnt--;
02948             _rt_alloc_table_tombstones++;
02949             if (_rt_alloc_table_tombstones > _rt_alloc_table_cap / 2 ||
02950                 (_rt_alloc_table_cap > _RT_ALLOC_INIT_CAP &&
02951                  _rt_alloc_table_cnt < _rt_alloc_table_cap / 4)) {
02952                 _rt_alloc_table_rehash();
02953             }
02954             return;
02955         }
02956         idx = (idx + 1) & (_rt_alloc_table_cap - 1);
02957     }
02958 }
02959
02960 static void _rt_panic_invalid_access(const char *reason, void *ptr, const char *loc_file, int
      loc_line) {
02961     _rt_panic_fmt(
02962         "runtime error: %s\n  pointer: %p\n  accessed at: %s:%d",
02963         reason,
02964         ptr,
02965         loc_file ? loc_file : "<unknown>",
02966         loc_line
02967     );
02968 }
02969
02970 static void _rt_panic_invalid_drop(const char *reason, void *ptr, const char *loc_file, int loc_line)
      {
02971     _rt_panic_fmt(
02972         "runtime error: invalid drop\n  reason: %s\n  pointer: %p\n  dropped at: %s:%d",
02973         reason,
02974         ptr,
02975         loc_file ? loc_file : "<unknown>",
02976         loc_line
02977     );
02978 }
02979
02980 static void _rt_panic_invalid_release(
02981     const char *op_name,
02982     const char *reason,
02983     void *ptr,
```

```
02984     const char *loc_file,
02985     int loc_line
02986 ) {
02987     _rt_panic_fmt(
02988         "runtime error: invalid %s\n  reason: %s\n  pointer: %p\n  released at: %s:%d",
02989         op_name ? op_name : "release",
02990         reason,
02991         ptr,
02992         loc_file ? loc_file : "<unknown>",
02993         loc_line
02994     );
02995 }
02996
02997 static void _rt_trace_invalid_drop(void *ptr, const char *loc_file, int loc_line) {
02998     _RT_TRACE_MEM(
02999         "op=drop ptr=%p action=panic-not-found loc=\"%s\":%d",
03000         ptr,
03001         loc_file ? loc_file : "<unknown>",
03002         loc_line
03003     );
03004 }
03005
03006 static int _rt_ptr_is_aligned(void *ptr, size_t align) {
03007     if (align <= 1) return 1;
03008     return ((uintptr_t)ptr % align) == 0;
03009 }
03010
03011 static int _rt_range_contains(void *base, size_t total_size, void *ptr, size_t need_size, size_t
      *offset_out) {
03012     uintptr_t b = (uintptr_t)base;
03013     uintptr_t p = (uintptr_t)ptr;
03014     if (p < b) return 0;
03015
03016     size_t offset = (size_t)(p - b);
03017     if (offset > total_size) return 0;
03018     if (need_size > total_size - offset) return 0;
03019     if (offset_out != NULL) {
03020         *offset_out = offset;
03021     }
03022     return 1;
03023 }
03024
03025 #ifndef L0_RT_CHECK_BASIC
03026 static inline uint32_t _rt_tree_prio_for(void *base) {
03027     uint64_t v = (uint64_t)(uintptr_t)base;
03028     uint32_t x = (uint32_t)(v ^ (v » 32));
03029     x ^= x » 16; x *= 0x85ebca6bu;
03030     x ^= x » 13; x *= 0xc2b2ae35u;
03031     x ^= x » 16;
03032     return x | 1u;
03033 }
03034
03035 static _rt_alloc_record *_rt_tree_insert_at(_rt_alloc_record *node, _rt_alloc_record *rec) {
03036     if (node == NULL) {
03037         rec->tree_left = NULL;
03038         rec->tree_right = NULL;
03039         return rec;
03040     }
03041     if ((uintptr_t)rec->base < (uintptr_t)node->base) {
03042         node->tree_left = _rt_tree_insert_at(node->tree_left, rec);
03043         if (node->tree_left->tree_prio > node->tree_prio) {
03044             _rt_alloc_record *pivot = node->tree_left;
03045             node->tree_left = pivot->tree_right;
03046             pivot->tree_right = node;
03047             node = pivot;
03048         }
03049     } else {
03050         node->tree_right = _rt_tree_insert_at(node->tree_right, rec);
03051         if (node->tree_right->tree_prio > node->tree_prio) {
03052             _rt_alloc_record *pivot = node->tree_right;
03053             node->tree_right = pivot->tree_left;
03054             pivot->tree_left = node;
03055             node = pivot;
03056         }
03057     }
03058     return node;
03059 }
03060
03062 static _rt_alloc_record *_rt_tree_merge(_rt_alloc_record *a, _rt_alloc_record *b) {
03063     if (a == NULL) return b;
03064     if (b == NULL) return a;
03065     if (a->tree_prio > b->tree_prio) {
03066         a->tree_right = _rt_tree_merge(a->tree_right, b);
03067         return a;
03068     }
03069     b->tree_left = _rt_tree_merge(a, b->tree_left);
03070     return b;
```

```
03071 }
03072
03073 static _rt_alloc_record *_rt_tree_remove_at(_rt_alloc_record *node, void *base) {
03074     if (node == NULL) return NULL;
03075     if ((uintptr_t)base < (uintptr_t)node->base) {
03076         node->tree_left = _rt_tree_remove_at(node->tree_left, base);
03077     } else if ((uintptr_t)base > (uintptr_t)node->base) {
03078         node->tree_right = _rt_tree_remove_at(node->tree_right, base);
03079     } else {
03080         node = _rt_tree_merge(node->tree_left, node->tree_right);
03081     }
03082     return node;
03083 }
03084
03085 static void _rt_alloc_tree_insert(_rt_alloc_record *rec) {
03086     rec->tree_prio = _rt_tree_prio_for(rec->base);
03087     _rt_alloc_tree_root = _rt_tree_insert_at(_rt_alloc_tree_root, rec);
03088 }
03089
03090 static void _rt_alloc_tree_remove(_rt_alloc_record *rec) {
03091     _rt_alloc_tree_root = _rt_tree_remove_at(_rt_alloc_tree_root, rec->base);
03092 }
03093
03095 static _rt_alloc_record *_rt_alloc_tree_glb(void *ptr) {
03096     _rt_alloc_record *node = _rt_alloc_tree_root;
03097     _rt_alloc_record *best = NULL;
03098     uintptr_t target = (uintptr_t)ptr;
03099     while (node != NULL) {
03100         if ((uintptr_t)node->base <= target) {
03101             best = node;
03102             node = node->tree_right;
03103         } else {
03104             node = node->tree_left;
03105         }
03106     }
03107     return best;
03108 }
03109
03111 static _rt_alloc_record *_rt_alloc_tree_lub(void *ptr) {
03112     _rt_alloc_record *node = _rt_alloc_tree_root;
03113     _rt_alloc_record *best = NULL;
03114     uintptr_t target = (uintptr_t)ptr;
03115     while (node != NULL) {
03116         if ((uintptr_t)node->base >= target) {
03117             best = node;
03118             node = node->tree_left;
03119         } else {
03120             node = node->tree_right;
03121         }
03122     }
03123     return best;
03124 }
03125
03126 static _rt_alloc_record *_rt_alloc_tree_find_containing(void *ptr, size_t need_size) {
03127     if (ptr == NULL) return NULL;
03128
03129     _rt_alloc_record *candidate = _rt_alloc_tree_glb(ptr);
03130     while (candidate != NULL) {
03131         if (_rt_range_contains(candidate->base, candidate->size, ptr, need_size, NULL)) {
03132             return candidate;
03133         }
03134         /* Static spans may overlap when the C compiler shares literal
03135          * suffixes; heap spans never overlap, so only static candidates
03136          * warrant walking further down the address order. */
03137         if (candidate->mem_kind != _RT_MEM_STATIC || (uintptr_t)candidate->base == 0) {
03138             return NULL;
03139         }
03140         candidate = _rt_alloc_tree_glb((void*)((uintptr_t)candidate->base - 1));
03141     }
03142     return NULL;
03143 }
03144 #endif /* L0_RT_CHECK_BASIC */
03145
03146 static void _rt_ptr_site_store(_rt_ptr_site *site, _rt_alloc_record *owner) {
03147     if (site == NULL || owner == NULL) return;
03148     site->owner = owner;
03149     site->generation = owner->generation;
03150 }
03151
03152 static size_t _rt_required_size(l0_int required_size) {
03153     if (required_size < 0) {
03154         _rt_panic("runtime pointer check: negative required size");
03155     }
03156     return required_size == 0 ? 1u : (size_t)required_size;
03157 }
03158
03159 static size_t _rt_required_align(l0_int required_align) {
```

```
03160     if (required_align < 0) {
03161         _rt_panic("runtime pointer check: negative required alignment");
03162     }
03163     return (size_t)required_align;
03164 }
03165
03166 static size_t _rt_index_element_size(l0_int element_size) {
03167     if (element_size <= 0) {
03168         _rt_panic("runtime pointer index: invalid element size");
03169     }
03170     return (size_t)element_size;
03171 }
03172
03173 static size_t _rt_index_offset(l0_int index, size_t element_size) {
03174     if (index < 0) {
03175         _rt_panic("runtime pointer index: negative pointer index");
03176     }
03177     /* `l0_int` is 32-bit, so the 64-bit product of two non-negative
03178      * operands cannot wrap and the SIZE_MAX comparison is exact. */
03179     if ((uint64_t)index * (uint64_t)element_size > SIZE_MAX) {
03180         _rt_panic_fmt(
03181             "runtime pointer index: index * element_size overflow (%d * %d)",
03182             (int)index,
03183             (int)element_size
03184         );
03185     }
03186     return (size_t)index * element_size;
03187 }
03188
03189 static int _rt_required_access_mode(int access_mode) {
03190     if (access_mode != _RT_ACCESS_READ && access_mode != _RT_ACCESS_WRITE) {
03191         _rt_panic("runtime pointer check: invalid access mode");
03192     }
03193     return access_mode;
03194 }
03195
03196 static int _rt_index_access_mode(int access_mode) {
03197     int mode = access_mode & _RT_ACCESS_WRITE;
03198     int flags = access_mode & ~_RT_ACCESS_WRITE;
03199     if ((flags != 0 && flags != _RT_ACCESS_UNTRACKED_OK) ||
03200         (mode != _RT_ACCESS_READ && mode != _RT_ACCESS_WRITE)) {
03201         _rt_panic("runtime pointer index: invalid access mode");
03202     }
03203     return mode;
03204 }
03205
03206 static int _rt_index_allows_untracked(int access_mode) {
03207     return (access_mode & _RT_ACCESS_UNTRACKED_OK) != 0;
03208 }
03209
03218 static void _rt_check_ptr_align(void *ptr, size_t need_align, const char *loc_file, int loc_line) {
03219     if (!_rt_ptr_is_aligned(ptr, need_align)) {
03220         _rt_panic_invalid_access("misaligned pointer access", ptr, loc_file, loc_line);
03221     }
03222 }
03223
03224 static void _rt_check_record_writeable(_rt_alloc_record *rec, void *ptr, int access_mode, const char
      *loc_file, int loc_line) {
03225     if (access_mode == _RT_ACCESS_WRITE && rec->read_only) {
03226         _rt_panic_invalid_access("read-only pointer write", ptr, loc_file, loc_line);
03227     }
03228 }
03229
03230 static void _rt_track_alloc_record_kind(
03231     void *ptr,
03232     size_t size,
03233     size_t align,
03234     uint32_t type_id,
03235     int mem_kind,
03236     int read_only,
03237     const char *loc_file,
03238     int loc_line
03239 ) {
03240     if (ptr == NULL) return;
03241     if (_rt_alloc_table_lookup(ptr) != NULL) {
03242         _rt_panic_fmt("runtime allocation tracker: duplicate allocation address %p", ptr);
03243     }
03244
03245     _rt_alloc_record_cold *cold = NULL;
03246     _rt_alloc_record *rec = _rt_rec_new(&cold);
03247
03248     rec->base = ptr;
03249     rec->size = size == 0 ? 1u : size;
03250     rec->generation = _rt_alloc_next_generation++;
03251     if (_rt_alloc_next_generation == 0) _rt_alloc_next_generation = 1;
03252     rec->state = _RT_ALLOC_LIVE;
03253     rec->read_only = read_only ? 1u : 0u;
```

```
03254     rec->mem_kind = (uint8_t)mem_kind;
03255     cold->align = align;
03256     cold->type_id = type_id;
03257     cold->alloc_file = loc_file;
03258     cold->alloc_line = loc_line;
03259     cold->drop_file = NULL;
03260     cold->drop_line = 0;
03261     cold->reserved = 0;
03262     rec->q_next = NULL;
03263 
03264     _rt_alloc_table_insert(rec);
03265 #ifndef L0_RT_CHECK_BASIC
03266     _rt_alloc_tree_insert(rec);
03267 #endif
03268 }
03269 
03270 static void _rt_track_alloc_record(
03271     void *ptr,
03272     size_t size,
03273     size_t align,
03274     uint32_t type_id,
03275     const char *loc_file,
03276     int loc_line
03277 ) {
03278     _rt_track_alloc_record_kind(ptr, size, align, type_id, _RT_MEM_RAW, 0, loc_file, loc_line);
03279 }
03280 
03281 static void _rt_promote_new_alloc(void *ptr) {
03282     if (ptr == NULL) return;
03283 
03284     _rt_alloc_record *rec = _rt_alloc_table_lookup(ptr);
03285     if (rec == NULL || rec->state != _RT_ALLOC_LIVE || rec->mem_kind != _RT_MEM_RAW) {
03286         _rt_panic_fmt("runtime allocation tracker: cannot promote non-raw allocation %p to new", ptr);
03287     }
03288     rec->mem_kind = _RT_MEM_NEW;
03289 }
03290 
03296 static void _rt_track_arc_bytes(void *ptr, size_t size) {
03297     if (ptr == NULL) return;
03298     if (_rt_alloc_table_lookup(ptr) != NULL) return;
03299 
03300     _rt_track_alloc_record_kind(ptr, size, 0, 0, _RT_MEM_ARC, 1, "<runtime>", 0);
03301 }
03302 
03303 static void _rt_untrack_arc_alloc(void *ptr) {
03304     if (ptr == NULL) return;
03305 
03306     _rt_alloc_record *rec = _rt_alloc_table_lookup(ptr);
03307     if (rec == NULL) return;
03308 
03309     _rt_alloc_table_remove(rec);
03310 #ifndef L0_RT_CHECK_BASIC
03311     _rt_alloc_tree_remove(rec);
03312 #endif
03313     _rt_rec_recycle(rec);
03314 }
03315 
03316 static void _rt_track_static_bytes(const void *ptr, size_t size) {
03317     if (ptr == NULL) return;
03318     if (_rt_alloc_table_lookup((void*)ptr) != NULL) return;
03319 #ifndef L0_RT_CHECK_BASIC
03320     if (_rt_alloc_tree_find_containing((void*)ptr, size == 0 ? 1u : size) != NULL) return;
03321 #else
03322     (void)size;
03323 #endif
03324 
03325     _rt_track_alloc_record_kind((void*)ptr, size, 0, 0, _RT_MEM_STATIC, 1, "<static>", 0);
03326 }
03327 
03329 static void rt_register_foreign(void *ptr, l0_int bytes, l0_bool read_only) {
03330     if (ptr == NULL) {
03331         _rt_panic("rt_register_foreign: null pointer");
03332     }
03333     if (bytes <= 0) {
03334         _rt_panic("rt_register_foreign: invalid byte extent");
03335     }
03336     if (read_only != 0 && read_only != 1) {
03337         _rt_panic("rt_register_foreign: invalid read-only flag");
03338     }
03339 
03340     size_t size = (size_t)bytes;
03341     uintptr_t base_addr = (uintptr_t)ptr;
03342     if (size > UINTPTR_MAX - base_addr) {
03343         _rt_panic("rt_register_foreign: address range overflow");
03344     }
03345 
03346     _rt_alloc_record *existing = _rt_alloc_table_lookup(ptr);
```

```
03347     if (existing != NULL) {
03348         if (existing->state == _RT_ALLOC_LIVE && existing->mem_kind == _RT_MEM_FOREIGN &&
03349             existing->size == size && existing->read_only == (read_only ? 1u : 0u)) {
03350             return;
03351         }
03352         _rt_panic_fmt("rt_register_foreign: conflicting tracked base %p", ptr);
03353     }
03354
03355 #ifndef L0_RT_CHECK_BASIC
03356     if (_rt_alloc_tree_find_containing(ptr, 1) != NULL) {
03357         _rt_panic_fmt("rt_register_foreign: range overlaps tracked storage at %p", ptr);
03358     }
03359     _rt_alloc_record *next = _rt_alloc_tree_lub(ptr);
03360     if (next != NULL && (uintptr_t)next->base < base_addr + size) {
03361         _rt_panic_fmt("rt_register_foreign: range overlaps tracked storage at %p", next->base);
03362     }
03363 #endif
03364
03365     _rt_track_alloc_record_kind(ptr, size, 0, 0, _RT_MEM_FOREIGN, read_only, "<foreign>", 0);
03366 }
03367
03369 static void rt_unregister_foreign(void *ptr) {
03370     if (ptr == NULL) {
03371         _rt_panic("rt_unregister_foreign: null pointer");
03372     }
03373
03374     _rt_alloc_record *rec = _rt_alloc_table_lookup(ptr);
03375     if (rec == NULL || rec->state != _RT_ALLOC_LIVE || rec->mem_kind != _RT_MEM_FOREIGN) {
03376         _rt_panic_fmt("rt_unregister_foreign: pointer is not a live foreign base %p", ptr);
03377     }
03378
03379     _rt_alloc_table_remove(rec);
03380 #ifndef L0_RT_CHECK_BASIC
03381     _rt_alloc_tree_remove(rec);
03382 #endif
03383     _rt_rec_recycle(rec);
03384 }
03385
03386 static void _rt_evict_quarantine(void) {
03387     while (_rt_quarantine_head != NULL &&
03388            (_rt_quarantine_bytes > _RT_QUARANTINE_MAX_BYTES ||
03389             _rt_quarantine_count > _RT_QUARANTINE_MAX_COUNT)) {
03390         _rt_alloc_record *rec = _rt_quarantine_head;
03391         _rt_quarantine_head = rec->q_next;
03392         if (_rt_quarantine_head == NULL) {
03393             _rt_quarantine_tail = NULL;
03394         }
03395
03396         _rt_quarantine_bytes -= rec->size;
03397         _rt_quarantine_count--;
03398         _rt_alloc_table_remove(rec);
03399 #ifndef L0_RT_CHECK_BASIC
03400         _rt_alloc_tree_remove(rec);
03401 #endif
03402         free(rec->base);
03403         _rt_rec_recycle(rec);
03404     }
03405 }
03406
03407 static void _rt_quarantine_alloc_record(_rt_alloc_record *rec, _rt_alloc_record_cold *cold, const char
      *loc_file, int loc_line) {
03408     rec->state = _RT_ALLOC_QUARANTINED;
03409     cold->drop_file = loc_file;
03410     cold->drop_line = loc_line;
03411     rec->q_next = NULL;
03412
03413     if (_rt_quarantine_tail == NULL) {
03414         _rt_quarantine_head = rec;
03415         _rt_quarantine_tail = rec;
03416     } else {
03417         _rt_quarantine_tail->q_next = rec;
03418         _rt_quarantine_tail = rec;
03419     }
03420
03421     _rt_quarantine_bytes += rec->size;
03422     _rt_quarantine_count++;
03423     _rt_evict_quarantine();
03424 }
03425
03426 static void _rt_release_tracked_alloc(void *ptr, const char *loc_file, int loc_line, const char
      *op_name) {
03427     if (ptr == NULL) return;
03428
03429     _rt_alloc_record *rec = _rt_alloc_table_lookup(ptr);
03430     if (rec == NULL) {
03431 #ifndef L0_RT_CHECK_BASIC
03432         if (_rt_alloc_tree_find_containing(ptr, 1) != NULL) {
```

```
03433             _rt_panic_invalid_release(op_name, "pointer is not an allocation base", ptr, loc_file,
      loc_line);
03434         }
03435 #endif
03436         _rt_panic_invalid_release(op_name, "unregistered pointer", ptr, loc_file, loc_line);
03437     }
03438     _rt_alloc_record_cold *cold = _rt_rec_cold(rec);
03439     if (rec->mem_kind != _RT_MEM_RAW) {
03440         const char *reason = "pointer is not a raw allocation";
03441         if (rec->mem_kind == _RT_MEM_NEW) reason = "new allocation must be released with drop";
03442         else if (rec->mem_kind == _RT_MEM_ARC) reason = "ARC-managed memory is not raw-owned";
03443         else if (rec->mem_kind == _RT_MEM_STATIC) reason = "static memory is not raw-owned";
03444         else if (rec->mem_kind == _RT_MEM_FOREIGN) reason = "foreign memory is not runtime-owned";
03445         _rt_panic_invalid_release(op_name, reason, ptr, loc_file, loc_line);
03446     }
03447     if (rec->state != _RT_ALLOC_LIVE) {
03448         _rt_panic_fmt(
03449             "runtime error: double %s\n  pointer: %p\n  first released at: %s:%d\n  second released
      at: %s:%d",
03450             op_name ? op_name : "release",
03451             ptr,
03452             cold->drop_file ? cold->drop_file : "<unknown>",
03453             cold->drop_line,
03454             loc_file ? loc_file : "<unknown>",
03455             loc_line
03456         );
03457     }
03458
03459     _rt_quarantine_alloc_record(rec, cold, loc_file, loc_line);
03460 }
03461
03462 static void *_rt_realloc_tracked_alloc(void *ptr, size_t new_size, const char *loc_file, int loc_line)
      {
03463     if (ptr == NULL) {
03464         void *new_ptr = malloc(new_size);
03465         if (new_ptr != NULL) {
03466             _rt_track_alloc_record(new_ptr, new_size, 0, 0, loc_file, loc_line);
03467         }
03468         return new_ptr;
03469     }
03470
03471     _rt_alloc_record *old_rec = _rt_alloc_table_lookup(ptr);
03472     _rt_alloc_record_cold *old_cold = old_rec != NULL ? _rt_rec_cold(old_rec) : NULL;
03473     if (old_rec == NULL || old_rec->state != _RT_ALLOC_LIVE ||
03474         old_rec->mem_kind != _RT_MEM_RAW) {
03475         const char *reason = "invalid realloc pointer";
03476         if (old_rec != NULL && old_rec->mem_kind == _RT_MEM_NEW) {
03477             reason = "new allocation cannot be reallocated";
03478         } else if (old_rec != NULL && old_rec->mem_kind == _RT_MEM_FOREIGN) {
03479             reason = "foreign memory is not runtime-owned";
03480         }
03481         _rt_panic_invalid_release("rt_realloc", reason, ptr, loc_file, loc_line);
03482     }
03483
03484     void *new_ptr = malloc(new_size);
03485     if (new_ptr == NULL) {
03486         return NULL;
03487     }
03488
03489     size_t copy_size = old_rec->size < new_size ? old_rec->size : new_size;
03490     if (copy_size > 0) {
03491         memcpy(new_ptr, ptr, copy_size);
03492     }
03493     _rt_track_alloc_record(new_ptr, new_size, old_cold->align, old_cold->type_id, loc_file, loc_line);
03494     _rt_quarantine_alloc_record(old_rec, old_cold, loc_file, loc_line);
03495     return new_ptr;
03496 }
03497
03498 static void *_rt_check_ptr_site_slow(_rt_ptr_site *site, void *ptr, l0_int required_size, l0_int
      required_align, int access_mode, const char *loc_file, int loc_line) {
03499     if (ptr == NULL) {
03500         _rt_panic_invalid_access("null pointer access", ptr, loc_file, loc_line);
03501     }
03502
03503     size_t need_size = _rt_required_size(required_size);
03504     size_t need_align = _rt_required_align(required_align);
03505     int mode = _rt_required_access_mode(access_mode);
03506
03507     _rt_alloc_record *base = _rt_alloc_table_lookup(ptr);
03508     if (base != NULL) {
03509         if (base->state != _RT_ALLOC_LIVE) {
03510             _rt_alloc_record_cold *base_cold = _rt_rec_cold(base);
03511             _rt_panic_fmt(
03512                 "runtime error: use after drop/free\n  pointer: %p\n  allocated at: %s:%d\n  released
      at: %s:%d\n  accessed at: %s:%d",
03513                 ptr,
03514                 base_cold->alloc_file ? base_cold->alloc_file : "<unknown>",
```

```
03515                 base_cold->alloc_line,
03516                 base_cold->drop_file ? base_cold->drop_file : "<unknown>",
03517                 base_cold->drop_line,
03518                 loc_file ? loc_file : "<unknown>",
03519                 loc_line
03520             );
03521         }
03522         if (need_size > base->size) {
03523             _rt_panic_invalid_access("pointer access exceeds allocation size", ptr, loc_file,
      loc_line);
03524         }
03525         _rt_check_ptr_align(ptr, need_align, loc_file, loc_line);
03526         _rt_check_record_writeable(base, ptr, mode, loc_file, loc_line);
03527         _rt_ptr_site_store(site, base);
03528         return ptr;
03529     }
03530
03531 #ifdef L0_RT_CHECK_BASIC
03532     _rt_check_ptr_align(ptr, need_align, loc_file, loc_line);
03533     return ptr;
03534 #else
03535     _rt_alloc_record *owner = _rt_alloc_tree_find_containing(ptr, need_size);
03536     if (owner != NULL) {
03537         if (owner->state != _RT_ALLOC_LIVE) {
03538             _rt_panic_invalid_access("stale derived pointer access", ptr, loc_file, loc_line);
03539         }
03540         _rt_check_ptr_align(ptr, need_align, loc_file, loc_line);
03541         _rt_check_record_writeable(owner, ptr, mode, loc_file, loc_line);
03542         _rt_ptr_site_store(site, owner);
03543         return ptr;
03544     }
03545
03546     if (_rt_alloc_tree_find_containing(ptr, 1) != NULL) {
03547         _rt_panic_invalid_access("pointer access exceeds allocation size", ptr, loc_file, loc_line);
03548     }
03549
03550     _rt_panic_invalid_access("unregistered pointer access", ptr, loc_file, loc_line);
03551     return ptr;
03552 #endif
03553 }
03554
03555 static void *_rt_check_index_ptr_site_slow(_rt_ptr_site *site, void *base_ptr, l0_int index, l0_int
      element_size, l0_int required_align, int access_mode, const char *loc_file, int loc_line) {
03556     if (base_ptr == NULL) {
03557         _rt_panic_invalid_access("null pointer access", base_ptr, loc_file, loc_line);
03558     }
03559
03560     size_t elem_size = _rt_index_element_size(element_size);
03561     size_t index_offset = _rt_index_offset(index, elem_size);
03562     size_t need_align = _rt_required_align(required_align);
03563     int mode = _rt_index_access_mode(access_mode);
03564
03565     _rt_alloc_record *owner = _rt_alloc_table_lookup(base_ptr);
03566 #ifndef L0_RT_CHECK_BASIC
03567     if (owner == NULL) {
03568         owner = _rt_alloc_tree_find_containing(base_ptr, 1);
03569     }
03570 #endif
03571     if (owner == NULL) {
03572 #ifdef L0_RT_CHECK_BASIC
03573         {
03574 #else
03575         if (_rt_index_allows_untracked(access_mode)) {
03576 #endif
03577             uintptr_t base_addr = (uintptr_t)base_ptr;
03578             if (index_offset > UINTPTR_MAX - base_addr) {
03579                 _rt_panic_invalid_access("pointer index outside allocation", base_ptr, loc_file,
      loc_line);
03580             }
03581             void *target = (void *)(base_addr + index_offset);
03582             _rt_check_ptr_align(target, need_align, loc_file, loc_line);
03583             return target;
03584         }
03585         _rt_panic_invalid_access("unregistered pointer index base", base_ptr, loc_file, loc_line);
03586     }
03587     if (owner->state != _RT_ALLOC_LIVE) {
03588         _rt_alloc_record_cold *owner_cold = _rt_rec_cold(owner);
03589         _rt_panic_fmt(
03590             "runtime error: use after drop/free\n  pointer: %p\n  allocated at: %s:%d\n  released at:
      %s:%d\n  accessed at: %s:%d",
03591             base_ptr,
03592             owner_cold->alloc_file ? owner_cold->alloc_file : "<unknown>",
03593             owner_cold->alloc_line,
03594             owner_cold->drop_file ? owner_cold->drop_file : "<unknown>",
03595             owner_cold->drop_line,
03596             loc_file ? loc_file : "<unknown>",
03597             loc_line
```

```
03598         );
03599     }
03600
03601     size_t base_offset = 0;
03602     if (!_rt_range_contains(owner->base, owner->size, base_ptr, 1, &base_offset)) {
03603         _rt_panic_invalid_access("pointer index base outside allocation", base_ptr, loc_file,
      loc_line);
03604     }
03605
03606     /* Two-step subtraction form: folding this into one containment check on
03607      * index_offset + elem_size could wrap size_t, so bound each term against
03608      * the remaining capacity instead. The first comparison guarantees the
03609      * second subtraction cannot underflow. */
03610     if (index_offset > owner->size - base_offset ||
03611         elem_size > owner->size - base_offset - index_offset) {
03612         _rt_panic_invalid_access("pointer index outside allocation", base_ptr, loc_file, loc_line);
03613     }
03614
03615     uintptr_t owner_addr = (uintptr_t)owner->base;
03616     size_t target_offset = base_offset + index_offset;
03617     if (target_offset > UINTPTR_MAX - owner_addr) {
03618         _rt_panic_invalid_access("pointer index outside allocation", base_ptr, loc_file, loc_line);
03619     }
03620
03621     void *target = (void *)(owner_addr + target_offset);
03622     _rt_check_ptr_align(target, need_align, loc_file, loc_line);
03623     _rt_check_record_writeable(owner, target, mode, loc_file, loc_line);
03624     _rt_ptr_site_store(site, owner);
03625     return target;
03626 }
03627
03628 static void *_rt_drop_begin_impl(
03629     void *ptr,
03630     l0_int required_size,
03631     l0_int required_align,
03632     const char *loc_file,
03633     int loc_line
03634 ) {
03635     if (ptr == NULL) {
03636         return NULL;
03637     }
03638
03639     size_t need_size = _rt_required_size(required_size);
03640     size_t need_align = _rt_required_align(required_align);
03641
03642     _rt_alloc_record *rec = _rt_alloc_table_lookup(ptr);
03643     if (rec == NULL) {
03644 #ifndef L0_RT_CHECK_BASIC
03645         if (_rt_alloc_tree_find_containing(ptr, 1) != NULL) {
03646             _rt_trace_invalid_drop(ptr, loc_file, loc_line);
03647             _rt_panic_invalid_drop("pointer is not an allocation base", ptr, loc_file, loc_line);
03648         }
03649 #endif
03650         _rt_trace_invalid_drop(ptr, loc_file, loc_line);
03651         _rt_panic_invalid_drop("unregistered pointer", ptr, loc_file, loc_line);
03652     }
03653     _rt_alloc_record_cold *cold = _rt_rec_cold(rec);
03654     if (rec->mem_kind != _RT_MEM_NEW) {
03655         const char *reason = "pointer was not allocated by new";
03656         if (rec->mem_kind == _RT_MEM_ARC) reason = "ARC-managed memory is not droppable";
03657         else if (rec->mem_kind == _RT_MEM_STATIC) reason = "static memory is not droppable";
03658         else if (rec->mem_kind == _RT_MEM_FOREIGN) reason = "foreign memory is not runtime-owned";
03659         _rt_trace_invalid_drop(ptr, loc_file, loc_line);
03660         _rt_panic_invalid_drop(reason, ptr, loc_file, loc_line);
03661     }
03662     if (rec->state != _RT_ALLOC_LIVE) {
03663         _rt_trace_invalid_drop(ptr, loc_file, loc_line);
03664         _rt_panic_fmt(
03665             "runtime error: double drop\n  pointer: %p\n  first dropped at: %s:%d\n  second dropped
      at: %s:%d",
03666             ptr,
03667             cold->drop_file ? cold->drop_file : "<unknown>",
03668             cold->drop_line,
03669             loc_file ? loc_file : "<unknown>",
03670             loc_line
03671         );
03672     }
03673     if (need_size > rec->size) {
03674         _rt_trace_invalid_drop(ptr, loc_file, loc_line);
03675         _rt_panic_invalid_drop("drop pointee exceeds allocation size", ptr, loc_file, loc_line);
03676     }
03677     if (!_rt_ptr_is_aligned(ptr, need_align)) {
03678         _rt_trace_invalid_drop(ptr, loc_file, loc_line);
03679         _rt_panic_invalid_drop("misaligned drop pointer", ptr, loc_file, loc_line);
03680     }
03681     return ptr;
03682 }
```

```
03683
03684 static void *_rt_validate_derived_ptr(void *derived, void *parent_base, l0_int size, l0_int align,
      const char *loc_file, int loc_line) {
03685     if (derived == NULL || parent_base == NULL) {
03686         return derived;
03687     }
03688
03689     size_t required_size = _rt_required_size(size);
03690     size_t required_align = _rt_required_align(align);
03691
03692     _rt_alloc_record *parent = _rt_alloc_table_lookup(parent_base);
03693 #ifndef L0_RT_CHECK_BASIC
03694     if (parent == NULL) {
03695         parent = _rt_alloc_tree_find_containing(parent_base, 1);
03696     }
03697 #endif
03698     if (parent == NULL) {
03699         /* Unregistered parent storage: leave validation to the access site. */
03700         return derived;
03701     }
03702     if (parent->state != _RT_ALLOC_LIVE) {
03703         _rt_panic_invalid_access("stale derived pointer access", parent_base, loc_file, loc_line);
03704     }
03705     if (!_rt_range_contains(parent->base, parent->size, derived, required_size, NULL)) {
03706         _rt_panic_invalid_access("derived pointer outside parent allocation", derived, loc_file,
      loc_line);
03707     }
03708     if (!_rt_ptr_is_aligned(derived, required_align)) {
03709         _rt_panic_invalid_access("misaligned derived pointer", derived, loc_file, loc_line);
03710     }
03711     return derived;
03712 }
03713
03714 #else /* L0_RT_UNCHECKED */
03715
03716 static void _rt_track_alloc_record(void *ptr, size_t size, size_t align, uint32_t type_id, const char
      *loc_file, int loc_line) {
03717     (void)ptr; (void)size; (void)align; (void)type_id; (void)loc_file; (void)loc_line;
03718 }
03719
03720 static void _rt_promote_new_alloc(void *ptr) {
03721     (void)ptr;
03722 }
03723
03724 static void _rt_track_arc_bytes(void *ptr, size_t size) {
03725     (void)ptr; (void)size;
03726 }
03727
03728 static void _rt_untrack_arc_alloc(void *ptr) {
03729     (void)ptr;
03730 }
03731
03732 static void _rt_track_static_bytes(const void *ptr, size_t size) {
03733     (void)ptr; (void)size;
03734 }
03735
03736 static void rt_register_foreign(void *ptr, l0_int bytes, l0_bool read_only) {
03737     if (ptr == NULL) {
03738         _rt_panic("rt_register_foreign: null pointer");
03739     }
03740     if (bytes <= 0) {
03741         _rt_panic("rt_register_foreign: invalid byte extent");
03742     }
03743     if (read_only != 0 && read_only != 1) {
03744         _rt_panic("rt_register_foreign: invalid read-only flag");
03745     }
03746 }
03747
03748 static void rt_unregister_foreign(void *ptr) {
03749     if (ptr == NULL) {
03750         _rt_panic("rt_unregister_foreign: null pointer");
03751     }
03752 }
03753
03754 static void _rt_release_tracked_alloc(void *ptr, const char *loc_file, int loc_line, const char
      *op_name) {
03755     (void)loc_file; (void)loc_line; (void)op_name;
03756     free(ptr);
03757 }
03758
03759 static void *_rt_realloc_tracked_alloc(void *ptr, size_t new_size, const char *loc_file, int loc_line)
      {
03760     (void)loc_file; (void)loc_line;
03761     return realloc(ptr, new_size);
03762 }
03763
03764 static void *_rt_drop_begin_impl(
```

```
03765     void *ptr,
03766     l0_int required_size,
03767     l0_int required_align,
03768     const char *loc_file,
03769     int loc_line
03770 ) {
03771     (void)required_size; (void)required_align; (void)loc_file; (void)loc_line;
03772     return ptr;
03773 }
03774
03775 static void *_rt_validate_derived_ptr(void *derived, void *parent_base, l0_int size, l0_int align,
      const char *loc_file, int loc_line) {
03776     (void)parent_base; (void)size; (void)align; (void)loc_file; (void)loc_line;
03777     return derived;
03778 }
03779
03780 #endif /* L0_RT_UNCHECKED */
03781
03782 static void _rt_drop_finish_impl(void *ptr, const char *loc_file, int loc_line) {
03783     if (ptr == NULL) {
03784         _RT_TRACE_MEM("op=drop ptr=%p action=noop-null loc=\"%s\":%d", ptr, loc_file, loc_line);
03785         return;
03786     }
03787
03788 #ifndef L0_RT_UNCHECKED
03789     _rt_alloc_record *rec = _rt_alloc_table_lookup(ptr);
03790     if (rec == NULL || rec->state != _RT_ALLOC_LIVE || rec->mem_kind != _RT_MEM_NEW) {
03791         _rt_trace_invalid_drop(ptr, loc_file, loc_line);
03792         _rt_panic_invalid_drop("drop finish without live new allocation", ptr, loc_file, loc_line);
03793     }
03794     _RT_TRACE_MEM("op=drop ptr=%p action=free loc=\"%s\":%d", ptr, loc_file, loc_line);
03795     _rt_quarantine_alloc_record(rec, _rt_rec_cold(rec), loc_file, loc_line);
03796 #else
03797     _RT_TRACE_MEM("op=drop ptr=%p action=free loc=\"%s\":%d", ptr, loc_file, loc_line);
03798     free(ptr);
03799 #endif
03800 }
03801 #ifdef L0_TRACE_MEMORY
03802 static void *_rt_alloc_obj_impl(l0_int bytes, const char *_loc_file, int _loc_line) {
03803     if (bytes <= 0) {
03804         _rt_panic("new: invalid allocation size");
03805     }
03806
03807     void *ptr = _rt_calloc_impl(1, bytes, _loc_file, _loc_line);
03808     if (ptr == NULL) {
03809         _rt_free_impl(ptr, _loc_file, _loc_line);
03810         _RT_TRACE_MEM("op=new_alloc bytes=%d ptr=%p action=panic-oom loc=\"%s\":%d", (int)bytes, ptr,
      _loc_file, _loc_line);
03811         _rt_panic("new: out of memory");
03812     }
03813
03814     _rt_promote_new_alloc(ptr);
03815     _RT_TRACE_MEM("op=new_alloc bytes=%d ptr=%p action=ok loc=\"%s\":%d", (int)bytes, ptr, _loc_file,
      _loc_line);
03816     return ptr;
03817 }
03818 #define _rt_alloc_obj(bytes) _rt_alloc_obj_impl((bytes), __FILE__, __LINE__)
03819 #else
03820 static void *_rt_alloc_obj(l0_int bytes) {
03821     if (bytes <= 0) {
03822         _rt_panic("new: invalid allocation size");
03823     }
03824
03825     void *ptr = rt_calloc(1, bytes);
03826     if (ptr == NULL) {
03827         rt_free(ptr);
03828         _RT_TRACE_MEM("op=new_alloc bytes=%d ptr=%p action=panic-oom", (int)bytes, ptr);
03829         _rt_panic("new: out of memory");
03830     }
03831
03832     _rt_promote_new_alloc(ptr);
03833     _RT_TRACE_MEM("op=new_alloc bytes=%d ptr=%p action=ok", (int)bytes, ptr);
03834     return ptr;
03835 }
03836 #endif
03837
03838 /* ============================================================================
03839  * Runtime support for hashing (using SipHash-1-3)
03840  * ============================================================================ */
03841
03848 static inline uint32_t _rt_fmix32(uint32_t x) {
03849     x ^= x >> 16;
03850     x *= 0x85ebca6bu;
03851     x ^= x >> 13;
03852     x *= 0xc2b2ae35u;
03853     x ^= x >> 16;
03854     return x;
```

```
03855 }
03856
03863 static inline uint32_t _rt_fold_u64_to_u32_fmix(uint64_t h) {
03864     uint32_t x = (uint32_t)(h ^ (h » 32));
03865     return _rt_fmix32(x);
03866 }
03867
03868 typedef uint8_t _rt_siphash_key_t[16];
03869 typedef uint8_t _rt_siphash_tag8_t[8];
03870
03871 /* Type tags for L0 runtime type identification */
03872 static const _rt_siphash_tag8_t _l0_sh_tag_bool   = { 0, 'b', 'o', 'o', 'l' };
03873 static const _rt_siphash_tag8_t _l0_sh_tag_byte   = { 0, 'i', 'n', 't', 8 };
03874 static const _rt_siphash_tag8_t _l0_sh_tag_int    = { 0, 'i', 'n', 't', 32 };
03875 static const _rt_siphash_tag8_t _l0_sh_tag_string = { 0, 's', 't', 'r', 'i', 'n', 'g' };
03876 static const _rt_siphash_tag8_t _l0_sh_tag_data   = { 0, 'd', 'a', 't', 'a' };
03877
03878 /* Flag bits for hash functions */
03879 #define _L0_TAG_OPT 0x80    /* option */
03880 #define _L0_TAG_PTR 0x40    /* pointer */
03881 #define _L0_TAG_ENUM 0x20   /* enum */
03882 #define _L0_TAG_STRUCT 0x10 /* struct */
03883
03888 static _rt_siphash_key_t _rt_sh_key = {
03889     0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
03890     0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
03891 };
03892
03903 static l0_int _rt_hash_tag8(const _rt_siphash_tag8_t tag8,
03904                             const uint8_t flags,
03905                             const void *data, size_t len,
03906                             const _rt_siphash_key_t key)
03907 {
03908     uint64_t hash = siphash13_tag8_bf(tag8, flags, data, len, key); /* compute SipHash-1-3 */
03909     return _rt_fold_u64_to_u32_fmix(hash);
03910 }
03911
03912 /* Hash functions for basic types */
03913
03921 static l0_int _rt_hash_bool(l0_bool value, const uint8_t flags) {
03922     return _rt_hash_tag8(_l0_sh_tag_bool, flags, &value, sizeof(l0_bool), _rt_sh_key);
03923 }
03924
03932 static l0_int _rt_hash_byte(l0_byte value, const uint8_t flags) {
03933     return _rt_hash_tag8(_l0_sh_tag_byte, flags, &value, sizeof(l0_byte), _rt_sh_key);
03934 }
03935
03943 static l0_int _rt_hash_int(l0_int value, const uint8_t flags) {
03944     return _rt_hash_tag8(_l0_sh_tag_int, flags, &value, sizeof(l0_int), _rt_sh_key);
03945 }
03946
03954 static l0_int _rt_hash_string(l0_string str, const uint8_t flags) {
03955     const char *str_data = _rt_string_bytes(str);
03956     l0_int str_len = rt_strlen(str);
03957     return _rt_hash_tag8(_l0_sh_tag_string, flags, str_data, (size_t)str_len, _rt_sh_key);
03958 }
03959
03968 static l0_int _rt_hash_data(void *data, l0_int size, const uint8_t flags) {
03969     return _rt_hash_tag8(_l0_sh_tag_data, flags, data, (size_t)size, _rt_sh_key);
03970 }
03971
03972 /* ============================================================================
03973  * User-exposed hash functions
03974  * ============================================================================ */
03975
03984 static l0_int rt_hash_bool(l0_bool value) {
03985     return _rt_hash_bool(value, 0);
03986 }
03987
03996 static l0_int rt_hash_byte(l0_byte value) {
03997     return _rt_hash_byte(value, 0);
03998 }
03999
04008 static l0_int rt_hash_int(l0_int value) {
04009     return _rt_hash_int(value, 0);
04010 }
04011
04020 static l0_int rt_hash_string(l0_string value) {
04021     return _rt_hash_string(value, 0);
04022 }
04023
04034 static l0_int rt_hash_data(void *data, l0_int size) {
04035     if (size < 0) {
04036         _rt_panic("rt_hash_data: negative size");
04037     }
04038     if (data == NULL) {
04039         _rt_panic("rt_hash_data: null data pointer");
```

```
04040     }
04041     return _rt_hash_data(data, size, 0);
04042 }
04043
04052 static l0_int rt_hash_opt_bool(l0_opt_bool opt) {
04053     uint8_t flags = _L0_TAG_OPT;
04054     return _rt_hash_data(&opt, sizeof(l0_opt_bool), flags);
04055 }
04056
04065 static l0_int rt_hash_opt_byte(l0_opt_byte opt) {
04066     uint8_t flags = _L0_TAG_OPT;
04067     return _rt_hash_data(&opt, sizeof(l0_opt_byte), flags);
04068 }
04069
04078 static l0_int rt_hash_opt_int(l0_opt_int opt) {
04079     uint8_t flags = _L0_TAG_OPT;
04080     return _rt_hash_data(&opt, sizeof(l0_opt_int), flags);
04081 }
04082
04092 static l0_int rt_hash_opt_string(l0_opt_string opt) {
04093     uint8_t flags = _L0_TAG_OPT;
04094     if (opt.has_value) {
04095         return _rt_hash_string(opt.value, flags);
04096     } else {
04097         return _rt_hash_string(L0_STRING_EMPTY, flags);
04098     }
04099 }
04100
04111 static l0_int rt_hash_ptr(void *ptr) {
04112     if (ptr == NULL) {
04113         _rt_panic("rt_hash_ptr: null pointer");
04114     }
04115     uint8_t flags = _L0_TAG_PTR;
04116     return _rt_hash_data(&ptr, sizeof(void*), flags);
04117 }
04118
04129 static l0_int rt_hash_opt_ptr(void *opt) {
04130     if (opt == NULL) {
04131         _rt_panic("rt_hash_opt_ptr: unwrap of empty optional");
04132     }
04133     uint8_t flags = _L0_TAG_OPT | _L0_TAG_PTR;
04134     return _rt_hash_data(&opt, sizeof(void*), flags);
04135 }
04136
04137 /* ============================================================================
04138  * End of L0 Runtime
04139  * ============================================================================ */
04140
04141 #endif /* L0_RUNTIME_H */
```

8.24 compiler/stage1_py/l0_analysis.py File Reference

Classes

- class l0_analysis.VarRefResolution

  Enumeration of variable reference resolution targets.
- class l0_analysis.AnalysisResult

  Full front-end analysis result for an entry module.

Namespaces

- namespace l0_analysis

8.25 compiler/stage1_py/l0_ast.py File Reference

Classes

- class l0_ast.Span

Represents a range of text in a source file.

- class l0_ast.Node

  Base class for all AST nodes.
- class l0_ast.TypeRef

  Reference to a type in the source code.
- class l0_ast.Import

  An import declaration.
- class l0_ast.TopLevelDecl

  Base class for top-level declarations (functions, structs, etc.).
- class l0_ast.Param

  A function parameter declaration.
- class l0_ast.FuncDecl

  A function declaration.
- class l0_ast.FieldDecl

  A field declaration within a struct.
- class l0_ast.StructDecl

  A struct declaration.
- class l0_ast.EnumVariant

  A variant declaration within an enum.
- class l0_ast.EnumDecl

  An enum declaration.
- class l0_ast.TypeAliasDecl

  A type alias declaration.
- class l0_ast.LetDecl

  A top-level 'let' declaration.
- class l0_ast.Module

  A complete L0 module.
- class l0_ast.Stmt

  Base class for all statements.
- class l0_ast.Block

  A block containing a sequence of statements.
- class l0_ast.LetStmt

  A local variable declaration statement.
- class l0_ast.AssignStmt

  An assignment statement.
- class l0_ast.ExprStmt

  A statement consisting of a single expression.
- class l0_ast.IfStmt

  An if-else statement.
- class l0_ast.WhileStmt

  A while loop statement.
- class l0_ast.ForStmt

  A for loop statement.
- class l0_ast.ReturnStmt

  A return statement.
- class l0_ast.DropStmt

  An explicit drop statement for manual memory/resource release.
- class l0_ast.MatchArm

  A single arm in a match statement.
- class l0_ast.MatchStmt

  A match statement for pattern matching.

- class l0_ast.WithItem
  An item in a with statement.
- class l0_ast.WithStmt
  A with statement for RAII-style resource management.
- class l0_ast.CaseArm
  A single arm in a case statement.
- class l0_ast.CaseElse
  The else arm of a case statement.
- class l0_ast.CaseStmt
  A case statement (switch-like).
- class l0_ast.BreakStmt
  A break statement.
- class l0_ast.ContinueStmt
  A continue statement.
- class l0_ast.Pattern
  Base class for match patterns.
- class l0_ast.WildcardPattern
  A wildcard pattern ('_') that matches anything.
- class l0_ast.VariantPattern
  A pattern that matches an enum variant.
- class l0_ast.Expr
  Base class for all expressions.
- class l0_ast.IntLiteral
  An integer literal expression.
- class l0_ast.ByteLiteral
  A byte literal expression.
- class l0_ast.StringLiteral
  A string literal expression.
- class l0_ast.BoolLiteral
  A boolean literal expression.
- class l0_ast.NullLiteral
  A null literal expression.
- class l0_ast.VarRef
  A variable reference expression.
- class l0_ast.NewExpr
  A heap allocation or constructor expression.
- class l0_ast.UnaryOp
  A unary operator expression.
- class l0_ast.BinaryOp
  A binary operator expression.
- class l0_ast.CallExpr
  A function or method call expression.
- class l0_ast.IndexExpr
  An array indexing expression.
- class l0_ast.FieldAccessExpr
  A field access expression.
- class l0_ast.ParenExpr
  A parenthesized expression.
- class l0_ast.CastExpr
  A type cast expression.
- class l0_ast.TryExpr
  A try expression for error handling.
- class l0_ast.TypeExpr
  A type used in an expression position.

Namespaces

- namespace l0_ast

8.26 compiler/stage1_py/l0_ast_printer.py File Reference

Namespaces

- namespace l0_ast_printer

Functions

- str l0_ast_printer._format_span (Span|None span)

  Format a source span for printing.
- List[str] l0_ast_printer.format_node (Any node, int indent=0)

  Generic, reflection-based AST pretty-printer.
- str l0_ast_printer.format_module (Module mod)

  Convenience function to pretty-print a single Module as a string.

8.27 compiler/stage1_py/l0_backend.py File Reference

Classes

- class l0_backend.Backend

  Language-agnostic code generation backend.

Namespaces

- namespace l0_backend

8.28 compiler/stage1_py/l0_c_emitter.py File Reference

Classes

- class l0_c_emitter.CCodeBuilder

  Helper for building C code with indentation tracking.
- class l0_c_emitter.CEmitter

  C-specific code emitter.

Namespaces

- namespace l0_c_emitter

8.29 compiler/stage1_py/l0_compilation.py File Reference

Classes

- class l0_compilation.CompilationUnit

  A closed set of modules starting from an entry module.

Namespaces

- namespace l0_compilation

8.30 compiler/stage1_py/l0_context.py File Reference

Classes

- class l0_context.LogLevel

  Hierarchical logging levels for the L0 compiler.
- class l0_context.CompilationContext

  Cross-cutting compiler options affecting multiple compilation stages.

Namespaces

- namespace l0_context

8.31 compiler/stage1_py/l0_diagnostics.py File Reference

Classes

- class l0_diagnostics.Diagnostic

  Represents a compiler diagnostic (error or warning).

Namespaces

- namespace l0_diagnostics

Functions

- Diagnostic l0_diagnostics.diag_from_node (str kind, str message, ∗Optional[str] module_name, Optional[str] filename, Optional[Node] node)

  Create a Diagnostic from an AST node.
- Diagnostic l0_diagnostics.diag_from_token (str kind, str message, ∗Optional[str] module_name, Optional[str] filename, Optional["Token"] token)

  Create a Diagnostic from a lexer token.

Variables

- dict l0_diagnostics.DIAGNOSTIC_CODE_FAMILIES

8.32 compiler/stage1_py/l0_driver.py File Reference

Classes

- class l0_driver.ImportCycleError

  Raised when a cyclic import is detected.
- class l0_driver.SourceEncodingError

  Raised when a source file cannot be decoded as UTF-8.
- class l0_driver.L0Driver

  Stage-1 driver for the L0 compiler.

Namespaces

- namespace l0_driver

Functions

- str l0_driver.load_source_utf8 (str|Path path)

  Read source bytes and decode as UTF-8.

8.33 compiler/stage1_py/l0_expr_types.py File Reference

Classes

- class l0_expr_types.StmtFlow

  Reachability after checking a statement or block.
- class l0_expr_types.LoopFlowCapture

  Liveness states for loop-control exits from one loop body.
- class l0_expr_types.ExpressionTypeChecker

  Expression-level type checker for L0.

Namespaces

- namespace l0_expr_types

8.34 compiler/stage1_py/l0_internal_error.py File Reference

Classes

- class l0_internal_error.ICELocation

  Represents the location where an internal compiler error occurred.
- class l0_internal_error.InternalCompilerError

  Exception raised when an internal compiler invariant is violated.

Namespaces

- namespace l0_internal_error

8.35 compiler/stage1_py/l0_lexer.py File Reference

Classes

- class l0_lexer.TokenKind

  Enumeration of all supported token types in L0.
- class l0_lexer.Token

  A single token produced by the lexer.
- class l0_lexer.Lexer

  Tokenizes L0 source code.

Namespaces

- namespace l0_lexer

Functions

- bool l0_lexer._is_ascii_ident_start (str c)

  Return whether c can start an ASCII Dea identifier.
- bool l0_lexer._is_ascii_ident_part (str c)

  Return whether c can continue an ASCII Dea identifier.
- bool l0_lexer._logical_predecessor_ends_expression (Token token, bool previous)

  Return whether the parser-visible predecessor ends an expression.
- bool l0_lexer.is_reserved_keyword (str word)

  Check if a word is a reserved L0 keyword.

Variables

- dict l0_lexer.KEYWORDS
- l0_lexer._EXPR_ENDING_TOKENS
- str l0_lexer.OCT_CHARS = "01234567"
- str l0_lexer.HEX_CHARS = "0123456789abcdefABCDEF"

8.36 compiler/stage1_py/l0_locals.py File Reference

Classes

- class l0_locals.LocalKind

  Enumeration of the different kinds of local bindings.
- class l0_locals.LocalSymbol

  Represents a single local binding.
- class l0_locals.Scope

  A lexical scope for local bindings.
- class l0_locals.FunctionEnv

  Environment for a single function.
- class l0_locals.LocalScopeResolver

  Builds lexical scopes for functions within a compilation unit.

Namespaces

- namespace l0_locals

8.37 compiler/stage1_py/l0_logger.py File Reference

Namespaces

- namespace l0_logger

Functions

- None l0_logger.log (CompilationContext context, LogLevel log_level, str message)

  Log a message according to the current context and level.
- None l0_logger.log_error (CompilationContext context, str message)

  Log an error-level message.
- None l0_logger.log_warning (CompilationContext context, str message)

  Log a warning-level message.
- None l0_logger.log_info (CompilationContext context, str message)

  Log an info-level message.
- None l0_logger.log_debug (CompilationContext context, str message)

  Log a debug-level message.
- None l0_logger.log_stage (CompilationContext context, str stage, Optional[str] module=None)

  Log the start or transition of a compilation stage.

8.38 compiler/stage1_py/l0_name_resolver.py File Reference

Classes

- class l0_name_resolver.NameResolver

  Stage-1 name resolver for module-level symbols.

Namespaces

- namespace l0_name_resolver

8.39 compiler/stage1_py/l0_parser.py File Reference

Classes

- class l0_parser._ParseSyncException

  Internal exception used for error recovery during parsing.
- class l0_parser.Parser

  Recursive descent parser for L0.

Namespaces

- namespace l0_parser

Functions

- int l0_parser.token_len (Token tok)

  Calculate the display length of a token.

8.40 compiler/stage1_py/l0_paths.py File Reference

Classes

- class l0_paths.SourceSearchPaths

  Search configuration for L0 modules.

Namespaces

- namespace l0_paths

8.41 compiler/stage1_py/l0_resolve.py File Reference

Classes

- class l0_resolve.ResolveErrorKind

  Kinds of errors that can occur during symbol resolution.
- class l0_resolve.TypeResolveErrorKind

  Kinds of errors that can occur during type resolution.
- class l0_resolve.SymbolResolution

  The result of a symbol resolution attempt.
- class l0_resolve.TypeResolution

  The result of a type reference resolution attempt.

Namespaces

- namespace l0_resolve

Functions

- bool l0_resolve._is_imported (ModuleEnv current_env, str module_name)

  Check if a module is explicitly imported by the current environment.
- SymbolResolution l0_resolve.resolve_symbol (Dict[str, ModuleEnv] module_envs, str current_module, str name, Optional[List[str]] module_path=None, *bool require_import=True)

  Resolve a symbol by name, optionally qualified by module path.
- TypeResolution l0_resolve.resolve_type_ref (Dict[str, ModuleEnv] module_envs, str current_module, TypeRef tref, Optional[List[str]] module_path=None, *Optional[Callable[[Symbol], Optional[Type]]] resolve_↩ alias=None, bool require_import=True)

  Resolve an AST TypeRef into a semantic Type.

8.42 compiler/stage1_py/l0_scope_context.py File Reference

Classes

- class l0_scope_context.ScopeContext

  Tracks variables and cleanup requirements for a single lexical scope.

Namespaces

- namespace l0_scope_context

8.43 compiler/stage1_py/l0_signatures.py File Reference

Classes

- class l0_signatures.StructFieldInfo

  Information about a single struct field.
- class l0_signatures.StructInfo

  Resolved type information for a struct.
- class l0_signatures.EnumVariantInfo

  Information about a single enum variant.
- class l0_signatures.EnumInfo

  Resolved type information for an enum.
- class l0_signatures.SignatureResolver

  Resolves top-level type signatures across all modules.

Namespaces

- namespace l0_signatures

8.44 compiler/stage1_py/l0_string_escape.py File Reference

Classes

- class l0_string_escape.EscapeDecodeError

  Raised when an invalid escape sequence is encountered during decoding.

Namespaces

- namespace l0_string_escape

Functions

- bytes l0_string_escape.decode_l0_string_token (str text)

  Decode an L0 string literal payload to raw bytes.
- str l0_string_escape.encode_c_string_bytes (bytes data)

  Encode raw bytes into a C-safe string literal body.

Variables

- str l0_string_escape._HEX_CHARS = "0123456789abcdefABCDEF"
- str l0_string_escape._OCT_CHARS = "01234567"
- l0_string_escape._TRIGRAPH_THIRD_BYTES = frozenset(map(ord, "=/'()!<>-"))
- dict l0_string_escape._SIMPLE_ESCAPES

8.45 compiler/stage1_py/l0_symbols.py File Reference

Classes

- class l0_symbols.SymbolKind

  Enumeration of all top-level symbol categories.
- class l0_symbols.Symbol

  A resolved module-level symbol.
- class l0_symbols.ModuleEnv

  Per-module symbol environment.

Namespaces

- namespace l0_symbols

8.46 compiler/stage1_py/l0_types.py File Reference

Classes

- class l0_types.Type

  Base class for all semantic types.
- class l0_types.BuiltinType

  A built-in primitive type.
- class l0_types.StructType

  A user-defined struct type.
- class l0_types.EnumType

  A user-defined enum type.
- class l0_types.PointerType

  A pointer to another type (T).*
- class l0_types.NullableType

  A nullable (optional) wrapper for another type (T?).
- class l0_types.FuncType

  A function signature type.
- class l0_types.NullType

  The type of the 'null' literal.

Namespaces

- namespace l0_types

Functions

- BuiltinType l0_types.get_builtin_type (str name)

  Get or create a canonical BuiltinType for a given name.
- NullType l0_types.get_null_type ()

  Get the canonical NullType instance.
- str l0_types.format_type (Optional[Type] t)

  Format a type into a human-readable string.

Variables

- tuple l0_types.L0_PRIMITIVE_TYPES = ("bool", "byte", "int", "string", "void")
- l0_types._NULL_TYPE = NullType()

8.47 compiler/stage1_py/l0c.py File Reference

Classes

- class l0c._TemporarySourceWriteError

  Report a generated-C write failure that also retained its path.

Namespaces

- namespace l0c

Functions

- str l0c.compiler_identity_text ()

  L0 Compiler (l0c) - Stage 1: Analysis and C Code Generation This is the main entry point for the L0 compiler's stage 1, which includes:
- Tuple[int, bool] l0c._scan_cli_presentation_options (Sequence[str] argv)

  Return fallback verbosity and rich-log settings before --.
- None l0c._emit_verbose_compiler_identity (argparse.Namespace args)

  Emit the compiler identity through the normal verbose logging path.
- None l0c._init_env_defaults ()

  Initialize default environment variables based on L0_HOME.
- List[str] l0c._load_file_lines (str path, Dict[str, List[str]] cache)

  Load lines of a source file, using a cache to avoid redundant reads.
- None l0c._emit_diagnostic (str message)

  Write one normative compiler diagnostic line to standard error.
- None l0c.print_diagnostics (AnalysisResult result)

  Print diagnostics from the analysis result.
- None l0c.print_diagnostic_list (List[Diagnostic] diagnostics)

  Print a list of diagnostics, using a file cache for source snippets.
- None l0c.print_diagnostic_with_snippet (Diagnostic diag, Dict[str, List[str]] file_cache)

  Print a single diagnostic, including the source line and a caret.
- bool l0c._is_valid_module_name (str module_name)

  Check if a module name is valid (dot-separated identifiers).

- Optional[SourceSearchPaths] l0c.build_search_paths (CompilationContext context, argparse.Namespace args)

  Build source search paths from command-line arguments.
- CompilationContext l0c.build_compilation_context (argparse.Namespace args)

  Build a CompilationContext from command-line arguments.
- l0c._run_analysis (argparse.Namespace args)

  Run the analysis pipeline.
- List[str] l0c._get_module_names (argparse.Namespace args, CompilationUnit cu)

  Get the list of module names based on the `--all-modules` flag.
- Optional[str] l0c._find_cc ()

  Find the best available C compiler.
- l0c._compiler_flag_family (str compiler)

  Determine the compiler family for flag selection using a simple heuristic.
- bool l0c._is_windows_host ()

  Return whether the current Python host is Windows.
- str l0c._default_executable_name ()

  Return the default build output name for the current platform.
- list[str] l0c._runtime_include_flags (str flag_family, str runtime_include)

  Return compiler-family-specific runtime include flags.
- list[str] l0c._runtime_library_flags (str flag_family, str runtime_lib)

  Return compiler-family-specific runtime library search-path flags.
- list[str] l0c._output_flags (str flag_family, Path exe_path)

  Return compiler-family-specific output flags.
- Path l0c._validated_temporary_directory ()

  Resolve and validate the host temporary directory.
- None l0c._emit_temporary_source_cleanup_failure (Path c_path)

  Report a compiler-temporary source retained after failed cleanup.
- bool l0c._remove_temporary_c_source (Path c_path)

  Remove a generated-C temporary and report retained recovery state.
- Path l0c._write_temporary_c_source (str c_code)

  Atomically create and write one anonymous generated-C source.
- bool l0c._check_entry_main_for_build (AnalysisResult result, str entry_name)

  Check that the entry module defines a valid 'main' function for build/run.
- bool l0c._validate_runtime_library_path (str runtime_lib_path)

  Validate that the provided runtime library path exists and is a directory.
- List[str] l0c._split_c_options (Optional[str] raw_options)

  Split a raw C options string into individual compiler arguments.
- Optional[str] l0c._get_optimize_flag (str flag_family, List[str] extra_opts)

  Determine the appropriate optimization flag for the compiler family.
- int l0c._compile_generated_c (argparse.Namespace args, CompilationContext context, Path c_path, Path exe_path)

  Invoke the selected host compiler for one generated-C source.
- int l0c.cmd_build (argparse.Namespace args)

  Build an executable from an L0 module.
- int l0c.cmd_run (argparse.Namespace args)

  Build and run an L0 module.
- int l0c.cmd_codegen (argparse.Namespace args)

  Generate C code for a module.
- int l0c.cmd_check (argparse.Namespace args)

  Run analysis and type checking without code generation.
- int l0c.cmd_compile (argparse.Namespace args)

Report that separate compilation is not implemented in Stage 1 yet.

- int l0c.cmd_ast (argparse.Namespace args)

  Pretty-print the parsed AST.
- str l0c._format_token_dump_text (tok)

  Return token payload text for `--tok` output.
- int l0c._dump_tokens_for_file (Path path, bool include_eof)

  Dump lexer tokens for a single file.
- int l0c.cmd_tok (argparse.Namespace args)

  Dump lexer tokens.
- int l0c.cmd_sym (argparse.Namespace args)

  Dump module-level symbol tables.
- int l0c.cmd_type (argparse.Namespace args)

  Dump resolved type information.
- None l0c._add_target_args (argparse.ArgumentParser parser)

  Add target module/file arguments to the parser.
- None l0c._add_all_modules_arg (argparse.ArgumentParser parser)

  Add the –all-modules flag to the parser.
- None l0c._add_runtime_args (argparse.ArgumentParser parser)

  Add runtime-related arguments to the parser.
- None l0c._add_codegen_arg (argparse.ArgumentParser parser)

  Add codegen-related arguments to the parser.
- Tuple[List[str], List[str], bool] l0c._split_cli_and_program_args (Sequence[str] argv)

  Split compiler CLI arguments from program arguments.
- List[str] l0c._normalize_cli_argv (argparse.ArgumentParser parser, Sequence[str] argv)

  Normalize exact short-option spellings before passing them to argparse.
- None l0c._validate_mode_scoped_flags (argparse.ArgumentParser parser, argparse.Namespace args)

  Reject flags that were provided for a mode where they are not valid.
- None l0c._validate_reserved_canonical_flags (argparse.ArgumentParser parser, argparse.Namespace args)

  Reject canonical compiler options whose behavior is not implemented.
- None l0c.main (Optional[List[str]] argv=None)

  Parse CLI arguments, dispatch the selected mode, and exit.

Variables

- dict l0c._CLI_LONG_VALUE_OPTIONS
- dict l0c._CLI_NAMESPACED_SHORT_VALUE_OPTIONS
- dict l0c._CLI_CANONICAL_SHORT_VALUE_OPTIONS
- dict l0c._CLI_OTHER_SHORT_VALUE_OPTIONS
- tuple l0c._CLI_VALUE_OPTIONS

8.48 compiler/stage2_l0/scripts/check_trace_log.py File Reference

Classes

- class check_trace_log._MessageCollector

  Count messages while retaining only a bounded leading sample.

Namespaces

- namespace check_trace_log

Functions

- tuple[Iterable[dict[str, str]], _MessageCollector, dict[str, int]] check_trace_log._iter_events (Iterable[str] lines, int max_details=20)

  Yield structured trace events from one raw line iterator.
- tuple[_MessageCollector, _MessageCollector, dict[str, int], dict] check_trace_log._validate_events (Iterable[dict[str, str]] events, int max_details=20)

  Validate trace event sequences for definite runtime misuse patterns.
- None check_trace_log._print_report (dict[str, int] counts, _MessageCollector parse_warnings, _MessageCollector errors, _MessageCollector warnings, dict[str, int] op_counts, dict triage, int max_details, bool show_triage)

  Print a validation summary and optional leak triage details.
- argparse.ArgumentParser check_trace_log._build_arg_parser ()

  Build the command-line parser for the trace log checker.
- int check_trace_log.main (list[str] argv)

  Run the trace log checker command-line entry point.

Variables

- check_trace_log.TRACE_RE = re.compile(r"^\[l0\]\[(mem|arc)\]\s+(.*)$")
- check_trace_log.KV_RE = re.compile(r"(\w+)=([^\s]+)")

8.49 compiler/stage2_l0/src/analysis.l0 File Reference

Module: `analysis`

```
import name_resolver;
import std.string;
import std.hashmap;
import std.vector;
import util.diag;
import driver;
import types;
import expr_types;
import locals;
import sem_context;
import util.log;
import std.text;
import signatures;
import source_paths;
```

Functions

- func analysis_copy_driver_diags(dst: DiagCollector*, src: DriverState*)

  Copies diagnostic messages from the driver state to a diagnostic collector.
- func analysis_signature_summary(sigs: SignatureTables*) → string

  Format the Stage 1-style signature-resolution summary text.
- func analysis_analyze_entry(search_paths: SourceSearchPaths*, entry_module: string, cfg: LogConfig*) → AnalysisResult*

  Analyzes the entry module and its dependencies.

8.49.1 Function Documentation

8.49.1.1 analysis_analyze_entry()

```
func analysis_analyze_entry(search_paths: SourceSearchPaths*, entry_module: string, cfg:
LogConfig*) → AnalysisResult*
```

Analyzes the entry module and its dependencies.

Parameters

search_paths	The source search paths configuration.
entry_module	The name of the entry module.
cfg	The logging configuration.

Returns

The complete analysis result including semantic context.

8.49.1.2 analysis_copy_driver_diags()

```
func analysis_copy_driver_diags(dst: DiagCollector*, src: DriverState*)
```

Copies diagnostic messages from the driver state to a diagnostic collector.

Parameters

dst	The destination diagnostic collector.
src	The source driver state containing diagnostics.

8.49.1.3 analysis_signature_summary()

```
func analysis_signature_summary(sigs: SignatureTables*) → string
```

Format the Stage 1-style signature-resolution summary text.

Parameters

sigs	Resolved signature tables.

Returns

Summary string for verbose logging.

8.50 compiler/stage2_l0/src/ast.l0 File Reference

Module: `ast`

```
import std.vector;
import std.string;
import std.assert;
```

Classes

- struct Span

  Represents the Span structure.
- struct TypeRef

  Represents the TypeRef structure.
- struct Import

  Represents the Import structure.
- struct Param

  Represents the Param structure.
- struct FieldDecl

  Represents the FieldDecl structure.
- struct EnumVariant

  Represents the EnumVariant structure.
- struct FuncDecl

  Represents the FuncDecl structure.
- struct StructDecl

  Represents the StructDecl structure.
- struct EnumDecl

  Represents the EnumDecl structure.
- struct TypeAliasDecl

  Represents the TypeAliasDecl structure.
- struct LetDecl

  Represents the LetDecl structure.
- struct TopLevelDecl

  Represents the TopLevelDecl structure.
- struct Module

  Represents the Module structure.
- struct PatternNode

  Represents the PatternNode structure.
- struct PatternArena

  Represents the PatternArena structure.
- struct ExprNode

  Represents the ExprNode structure.
- struct ExprArena

  Represents the ExprArena structure.
- struct MatchArm

  Represents the MatchArm structure.
- struct CaseArm

  Represents the CaseArm structure.
- struct CaseElse

  Represents the CaseElse structure.
- struct WithItem

  Represents the WithItem structure.
- struct StmtNode

  Represents the StmtNode structure.
- struct StmtArena

  Represents the StmtArena structure.

Enumerations

- enum TopLevelDeclKind

  Defines the TopLevelDeclKind enumeration.
- enum PatternKind { PT_WILDCARD , PT_VARIANT }

  Defines the PatternKind enumeration.
- enum ExprKind {
  EX_INT , EX_BYTE , EX_STRING , EX_BOOL ,
  EX_NULL , EX_VAR_REF , EX_NEW , EX_UNARY ,
  EX_BINARY , EX_CALL , EX_INDEX , EX_FIELD ,
  EX_PAREN , EX_CAST , EX_TRY , EX_TYPE }

  Defines the ExprKind enumeration.
- enum StmtKind {
  ST_BLOCK , ST_LET , ST_ASSIGN , ST_EXPR ,
  ST_IF , ST_WHILE , ST_FOR , ST_RETURN ,
  ST_DROP , ST_MATCH , ST_CASE , ST_BREAK ,
  ST_CONTINUE , ST_WITH }

  Defines the StmtKind enumeration.

Functions

- type ExprId = int

  Returns an invalid ID for a expr to indicate absence.
- type StmtId = int

  Returns an invalid ID for a stmt to indicate absence.
- type PatternId = int

  Returns an invalid ID for a pattern to indicate absence.
- func ptr_vec_create(initial_capacity: int) → VectorBase∗

  Create a vector of pointer slots.
- func ptr_vec_push(self: VectorBase∗, ptr: void∗)

  Append a pointer value to a pointer vector.
- func ptr_vec_get(self: VectorBase∗, index: int) → void∗

  Return a pointer value from a pointer vector.
- func id_vec_create(initial_capacity: int) → VectorBase∗

  Create a vector of integer ids.
- func id_vec_push(self: VectorBase∗, id: int)

  Append an id to an id vector.
- func id_vec_get(self: VectorBase∗, index: int) → int

  Return an id from an id vector.
- func span_here(line: int, column: int) → Span

  Build a zero-width span at a source location.
- func pattern_node_create(kind: PatternKind, span: Span) → PatternNode∗

  Create a pattern node.
- func pattern_arena_create(initial_capacity: int) → PatternArena∗

  Create a pattern arena.
- func pattern_arena_add(self: PatternArena∗, node: PatternNode∗) → PatternId

  Add a pattern node to the arena.
- func pattern_arena_get(self: PatternArena∗, id: PatternId) → PatternNode∗

  Return a pattern node from the arena.
- func sv_free_opt(v: StringVector∗?)

  Free an optional string vector.

- func pattern_node_free(node: PatternNode∗)

  Free a pattern node and any owned vectors.
- func pattern_arena_free(self: PatternArena∗)

  Free a pattern arena and every stored node.
- func expr_node_create(kind: ExprKind, span: Span) → ExprNode∗

  Create an expression node.
- func expr_arena_create(initial_capacity: int) → ExprArena∗

  Create an expression arena.
- func expr_arena_add(self: ExprArena∗, node: ExprNode∗) → ExprId

  Add an expression node to the arena.
- func expr_arena_get(self: ExprArena∗, id: ExprId) → ExprNode∗

  Return an expression node from the arena.
- func stmt_node_create(kind: StmtKind, span: Span) → StmtNode∗

  Create a statement node.
- func stmt_arena_create(initial_capacity: int) → StmtArena∗

  Create a statement arena.
- func stmt_arena_add(self: StmtArena∗, node: StmtNode∗) → StmtId

  Add a statement node to the arena.
- func stmt_arena_get(self: StmtArena∗, id: StmtId) → StmtNode∗

  Return a statement node from the arena.
- func type_ref_free(self: TypeRef∗?)

  Free an optional type reference.
- func param_free(self: Param∗)

  Free a function parameter node.
- func field_decl_free(self: FieldDecl∗)

  Free a field declaration node.
- func enum_variant_free(self: EnumVariant∗)

  Free an enum variant declaration.
- func func_decl_free(self: FuncDecl∗)

  Free a function declaration.
- func struct_decl_free(self: StructDecl∗)

  Free a struct declaration.
- func enum_decl_free(self: EnumDecl∗)

  Free an enum declaration.
- func type_alias_decl_free(self: TypeAliasDecl∗)

  Free a type-alias declaration.
- func let_decl_free(self: LetDecl∗)

  Free a top-level `let` declaration.
- func top_level_decl_free(self: TopLevelDecl∗)

  Free a top-level declaration wrapper and its payload.
- func module_free(self: Module∗)

  Free a parsed module and its owned declarations.
- func id_vec_free(v: VectorBase∗?)

  Free an optional id vector.
- func import_vec_free(v: VectorBase∗?)

  Free an optional import vector.
- func top_level_decl_vec_free(v: VectorBase∗?)

  Free an optional top-level declaration vector.
- func param_vec_free(v: VectorBase∗?)

  Free an optional parameter vector.
- func field_decl_vec_free(v: VectorBase∗?)

Free an optional field-declaration vector.

- func enum_variant_vec_free(v: VectorBase∗?)

  Free an optional enum-variant vector.

- func match_arm_vec_free(v: VectorBase∗?)

  Free an optional match-arm vector.

- func case_arm_vec_free(v: VectorBase∗?)

  Free an optional case-arm vector.

- func with_item_vec_free(v: VectorBase∗?)

  Free an optional `with`-item vector.

- func expr_node_free(self: ExprNode∗)

  Free an expression node and its owned payload.

- func stmt_node_free(self: StmtNode∗)

  Free a statement node and its owned payload.

- func expr_arena_free(self: ExprArena∗)

  Free an expression arena and every stored node.

- func stmt_arena_free(self: StmtArena∗)

  Free a statement arena and every stored node.

Variables

- type ExprId = int
- type StmtId = int
- type PatternId = int

8.50.1 Enumeration Type Documentation

8.50.1.1 ExprKind

```
enum ExprKind
```

Defines the ExprKind enumeration.

Enumerator

EX_INT	
EX_BYTE	
EX_STRING	
EX_BOOL	
EX_NULL	
EX_VAR_REF	
EX_NEW	
EX_UNARY	
EX_BINARY	
EX_CALL	
EX_INDEX	
EX_FIELD	
EX_PAREN	
EX_CAST	
EX_TRY	
EX_TYPE	

8.50.1.2 PatternKind

```
enum PatternKind
```

Defines the PatternKind enumeration.

Enumerator

PT_WILDCARD	
PT_VARIANT	

8.50.1.3 StmtKind

```
enum StmtKind
```

Defines the StmtKind enumeration.

Enumerator

ST_BLOCK	
ST_LET	
ST_ASSIGN	
ST_EXPR	
ST_IF	
ST_WHILE	
ST_FOR	
ST_RETURN	
ST_DROP	
ST_MATCH	
ST_CASE	
ST_BREAK	
ST_CONTINUE	
ST_WITH	

8.50.1.4 TopLevelDeclKind

```
enum TopLevelDeclKind
```

Defines the TopLevelDeclKind enumeration.

Enumerator

TD_FUNC	
TD_STRUCT	
TD_ENUM	
TD_TYPE_ALIAS	
TD_LET	

8.50.2 Function Documentation

8.50.2.1 case_arm_vec_free()

```
func case_arm_vec_free(v:  VectorBase*?)
```

Free an optional case-arm vector.

Parameters

v	Optional case-arm vector to free.

8.50.2.2 enum_decl_free()

```
func enum_decl_free(self:  EnumDecl*)
```

Free an enum declaration.

Parameters

self	Enum declaration to free.

8.50.2.3 enum_variant_free()

```
func enum_variant_free(self:  EnumVariant*)
```

Free an enum variant declaration.

Parameters

self	Enum variant to free.

8.50.2.4 enum_variant_vec_free()

```
func enum_variant_vec_free(v:  VectorBase*?)
```

Free an optional enum-variant vector.

Parameters

v	Optional enum-variant vector to free.

8.50.2.5 expr_arena_add()

```
func expr_arena_add(self:  ExprArena*, node:  ExprNode*) → ExprId
```

Add an expression node to the arena.

Parameters

self	Expression arena to extend.
node	Expression node to store.

Returns

Expression id assigned to the stored node.

8.50.2.6 expr_arena_create()

```
func expr_arena_create(initial_capacity:  int) → ExprArena*
```

Create an expression arena.

Parameters

initial_capacity	Initial arena capacity.

Returns

New expression arena.

8.50.2.7 expr_arena_free()

```
func expr_arena_free(self:  ExprArena*)
```

Free an expression arena and every stored node.

Parameters

self	Expression arena to free.

8.50.2.8 expr_arena_get()

```
func expr_arena_get(self:  ExprArena*, id:  ExprId) → ExprNode*
```

Return an expression node from the arena.

Parameters

self	Expression arena to inspect.
id	Expression id to look up.

Returns

Expression node for `id`.

8.50.2.9 expr_node_create()

```
func expr_node_create(kind:  ExprKind, span:  Span) → ExprNode*
```

Create an expression node.

Parameters

kind	Expression kind.
span	Source span for the node.

Returns

New expression node.

8.50.2.10 expr_node_free()

```
func expr_node_free(self:  ExprNode*)
```

Free an expression node and its owned payload.

Parameters

self	Expression node to free.

8.50.2.11 field_decl_free()

```
func field_decl_free(self:  FieldDecl*)
```

Free a field declaration node.

Parameters

self	Field declaration to free.

8.50.2.12 field_decl_vec_free()

```
func field_decl_vec_free(v:  VectorBase*?)
```

Free an optional field-declaration vector.

Parameters

v	Optional field-declaration vector to free.

8.50.2.13 func_decl_free()

```
func func_decl_free(self:  FuncDecl*)
```

Free a function declaration.

Parameters

self	Function declaration to free.

8.50.2.14 id_vec_create()

```
func id_vec_create(initial_capacity:  int) → VectorBase*
```

Create a vector of integer ids.

Parameters

initial_capacity	Initial vector capacity.

Returns

New id vector.

8.50.2.15 id_vec_free()

```
func id_vec_free(v:  VectorBase*?)
```

Free an optional id vector.

Parameters

v	Optional vector to free.

8.50.2.16 id_vec_get()

```
func id_vec_get(self:  VectorBase*, index:  int) → int
```

Return an id from an id vector.

Parameters

self	Id vector to inspect.
index	Zero-based element index.

Returns

Id value at `index`.

8.50.2.17 id_vec_push()

```
func id_vec_push(self:  VectorBase*, id:  int)
```

Append an id to an id vector.

Parameters

self	Id vector to extend.
id	Id value to append.

8.50.2.18 import_vec_free()

```
func import_vec_free(v:  VectorBase*?)
```

Free an optional import vector.

Parameters

v	Optional import vector to free.

8.50.2.19 invalid_expr_id()

```
func invalid_expr_id() → ExprId
```

Returns an invalid ID for a expr to indicate absence.

Returns

The sentinel value indicating an invalid expression ID (-1).

8.50.2.20 invalid_pattern_id()

```
func invalid_pattern_id() → PatternId
```

Returns an invalid ID for a pattern to indicate absence.

Returns

The sentinel value indicating an invalid pattern ID (-1).

8.50.2.21 invalid_stmt_id()

```
func invalid_stmt_id() → StmtId
```

Returns an invalid ID for a stmt to indicate absence.

Returns

The sentinel value indicating an invalid statement ID (-1).

8.50.2.22 let_decl_free()

```
func let_decl_free(self:  LetDecl*)
```

Free a top-level `let` declaration.

Parameters

self	Let declaration to free.

8.50.2.23 match_arm_vec_free()

```
func match_arm_vec_free(v:  VectorBase*?)
```

Free an optional match-arm vector.

Parameters

v	Optional match-arm vector to free.

8.50.2.24 module_free()

```
func module_free(self:  Module*)
```

Free a parsed module and its owned declarations.

Parameters

self	Module to free.

8.50.2.25 param_free()

```
func param_free(self:  Param*)
```

Free a function parameter node.

Parameters

self	Parameter to free.

8.50.2.26 param_vec_free()

```
func param_vec_free(v:  VectorBase*?)
```

Free an optional parameter vector.

Parameters

v	Optional parameter vector to free.

8.50.2.27 pattern_arena_add()

```
func pattern_arena_add(self:  PatternArena*, node:  PatternNode*) → PatternId
```

Add a pattern node to the arena.

Parameters

self	Pattern arena to extend.
node	Pattern node to store.

Returns

Pattern id assigned to the stored node.

8.50.2.28 pattern_arena_create()

```
func pattern_arena_create(initial_capacity:  int) → PatternArena*
```

Create a pattern arena.

Parameters

initial_capacity	Initial arena capacity.

Returns

New pattern arena.

8.50.2.29 pattern_arena_free()

```
func pattern_arena_free(self:  PatternArena*)
```

Free a pattern arena and every stored node.

Parameters

self	Pattern arena to free.

8.50.2.30 pattern_arena_get()

```
func pattern_arena_get(self:  PatternArena*, id:  PatternId) → PatternNode*
```

Return a pattern node from the arena.

Parameters

self	Pattern arena to inspect.
id	Pattern id to look up.

Returns

Pattern node for `id`.

8.50.2.31 pattern_node_create()

```
func pattern_node_create(kind:  PatternKind, span:  Span) → PatternNode*
```

Create a pattern node.

Parameters

kind	Pattern kind.
span	Source span for the node.

Returns

New pattern node.

8.50.2.32 pattern_node_free()

```
func pattern_node_free(node:  PatternNode*)
```

Free a pattern node and any owned vectors.

Parameters

node	Pattern node to free.

8.50.2.33 ptr_vec_create()

```
func ptr_vec_create(initial_capacity:  int) → VectorBase*
```

Create a vector of pointer slots.

Parameters

initial_capacity	Initial vector capacity.

Returns

New pointer vector.

8.50.2.34 ptr_vec_get()

```
func ptr_vec_get(self:  VectorBase*, index:  int) → void*
```

Return a pointer value from a pointer vector.

Parameters

self	Pointer vector to inspect.
index	Zero-based element index.

Returns

Pointer value at `index`.

8.50.2.35 ptr_vec_push()

```
func ptr_vec_push(self:  VectorBase*, ptr:  void*)
```

Append a pointer value to a pointer vector.

Parameters

self	Pointer vector to extend.
ptr	Pointer value to append.

8.50.2.36 span_here()

```
func span_here(line:  int, column:  int) → Span
```

Build a zero-width span at a source location.

Parameters

line	One-based source line.
column	One-based source column.

Returns

Span whose start and end are the same location.

8.50.2.37 stmt_arena_add()

```
func stmt_arena_add(self:  StmtArena*, node:  StmtNode*) → StmtId
```

Add a statement node to the arena.

Parameters

self	Statement arena to extend.
node	Statement node to store.

Returns

Statement id assigned to the stored node.

8.50.2.38 stmt_arena_create()

```
func stmt_arena_create(initial_capacity:  int) → StmtArena*
```

Create a statement arena.

Parameters

initial_capacity	Initial arena capacity.

Returns

New statement arena.

8.50.2.39 stmt_arena_free()

```
func stmt_arena_free(self:  StmtArena*)
```

Free a statement arena and every stored node.

Parameters

self	Statement arena to free.

8.50.2.40 stmt_arena_get()

```
func stmt_arena_get(self:  StmtArena*, id:  StmtId) → StmtNode*
```

Return a statement node from the arena.

Parameters

self	Statement arena to inspect.
id	Statement id to look up.

Returns

Statement node for `id`.

8.50.2.41 stmt_node_create()

```
func stmt_node_create(kind:  StmtKind, span:  Span) → StmtNode*
```

Create a statement node.

Parameters

kind	Statement kind.
span	Source span for the node.

Returns

New statement node.

8.50.2.42 stmt_node_free()

```
func stmt_node_free(self:  StmtNode*)
```

Free a statement node and its owned payload.

Parameters

self	Statement node to free.

8.50.2.43 struct_decl_free()

```
func struct_decl_free(self:  StructDecl*)
```

Free a struct declaration.

Parameters

self	Struct declaration to free.

8.50.2.44 sv_free_opt()

```
func sv_free_opt(v:  StringVector*?)
```

Free an optional string vector.

Parameters

v	Optional vector to free.

8.50.2.45 top_level_decl_free()

```
func top_level_decl_free(self:  TopLevelDecl*)
```

Free a top-level declaration wrapper and its payload.

Parameters

self	Top-level declaration to free.

8.50.2.46 top_level_decl_vec_free()

```
func top_level_decl_vec_free(v:  VectorBase*?)
```

Free an optional top-level declaration vector.

Parameters

v	Optional declaration vector to free.

8.50.2.47 type_alias_decl_free()

```
func type_alias_decl_free(self:  TypeAliasDecl*)
```

Free a type-alias declaration.

Parameters

self	Type-alias declaration to free.

8.50.2.48 type_ref_free()

```
func type_ref_free(self:  TypeRef*?)
```

Free an optional type reference.

Parameters

self	Optional type reference to free.

8.50.2.49 with_item_vec_free()

```
func with_item_vec_free(v:  VectorBase*?)
```

Free an optional `with`-item vector.

Parameters

v	Optional `with`-item vector to free.

8.50.3 Variable Documentation

8.50.3.1 ExprId

```
type ExprId = int
```

8.50.3.2 PatternId

```
type PatternId = int
```

8.50.3.3 StmtId

```
type StmtId = int
```

8.51 compiler/stage2_l0/src/ast_printer.l0 File Reference

Module: `ast_printer`

```
import parser;
import ast;
import std.vector;
import std.text;
import std.string;
import std.io;
```

Functions

- func ap_bool_str(b: bool) → string

  Return "True" or "False" for a boolean value.
- func ap_span_suffix(span: Span) → string

  Build a span-suffix annotation string @sl:sc-el:ec.
- func ap_indent_str(n: int) → string

  Build an indentation prefix string (2 spaces per level).
- func ap_print_line(indent: int, text: string)

  Print a line at the given indent level.
- func ap_print_label(indent: int, name: string)

  Print a child-field label at indent + 1.
- func ap_quote_string(s: string) → string

  Return a single-quoted escaped representation of a string.
- func ap_module_prefix(module_path: StringVector∗?) → string

  Build the module qualifier prefix from an optional path vector.
- func ap_full_name(text_value: string, module_path: StringVector∗?, name_qualifier: StringVector∗?) → string

  Build the full qualified name for a variable reference or similar node.
- func ap_print_type_ref(tr: TypeRef∗, indent: int)

  Print a type reference node.
- func ap_print_import(imp: Import∗, indent: int)

  Print an import node.
- func ap_print_param(p: Param∗, indent: int)

  Print a parameter node, including its type annotation.
- func ap_print_field_decl(fd: FieldDecl∗, indent: int)

  Print a field declaration node.
- func ap_print_enum_variant(ev: EnumVariant∗, indent: int)

  Print an enum variant node, including any payload fields.
- func ap_print_pattern(id: PatternId, pr: ParseResult∗, indent: int)

  Print a pattern node by arena id.
- func ap_print_expr(id: ExprId, pr: ParseResult∗, indent: int)

  Print an expression node by arena id.
- func ap_print_stmt(id: StmtId, pr: ParseResult∗, indent: int)

  Print a statement node by arena id.
- func ap_print_func_decl(fd: FuncDecl∗, pr: ParseResult∗, indent: int)

  Print a function declaration including params, return type, and body.
- func ap_print_struct_decl(sd: StructDecl∗, indent: int)

  Print a struct declaration including its fields.
- func ap_print_enum_decl(ed: EnumDecl∗, indent: int)

  Print an enum declaration including its variants.
- func ap_print_type_alias_decl(ta: TypeAliasDecl∗, indent: int)

  Print a type-alias declaration.
- func ap_print_let_decl(ld: LetDecl∗, pr: ParseResult∗, indent: int)

  Print a top-level let declaration.
- func ap_print_top_level_decl(tld: TopLevelDecl∗, pr: ParseResult∗, indent: int)

  Dispatch to the correct per-kind declaration printer.
- func ap_print_module(mod: Module∗, pr: ParseResult∗)

  Print the AST of a parsed module.

8.51.1 Function Documentation

8.51.1.1 ap_bool_str()

```
func ap_bool_str(b: bool) → string
```

Return "True" or "False" for a boolean value.

Parameters

b	Boolean value.

Returns

String representation matching Python `repr(bool)` style.

8.51.1.2 ap_full_name()

```
func ap_full_name(text_value: string, module_path: StringVector*?, name_qualifier: StringVector*?)
→ string
```

Build the full qualified name for a variable reference or similar node.

Combines module_path, name_qualifier segments, and the local name into a display string like "std.io:: SomeType" or "Enum:: Variant".

Parameters

text_value	Base name.
module_path	Optional module path segments.
name_qualifier	Optional name qualifier segments.

Returns

Fully qualified display name.

8.51.1.3 ap_indent_str()

```
func ap_indent_str(n: int) → string
```

Build an indentation prefix string (2 spaces per level).

Parameters

n	Indent level.

Returns

Indentation string.

8.51.1.4 ap_module_prefix()

```
func ap_module_prefix(module_path:  StringVector*?)  → string
```

Build the module qualifier prefix from an optional path vector.

For a module path `["std", "io"]` this returns `"std.io::"`. Returns `""` when the path is null or empty.

Parameters

module_path	Optional module path segments.

Returns

Formatted prefix ending with `"::"`, or empty string.

8.51.1.5 ap_print_enum_decl()

```
func ap_print_enum_decl(ed:  EnumDecl*, indent:  int)
```

Print an enum declaration including its variants.

Parameters

ed	Enum declaration to print.
indent	Indent level.

8.51.1.6 ap_print_enum_variant()

```
func ap_print_enum_variant(ev:  EnumVariant*, indent:  int)
```

Print an enum variant node, including any payload fields.

Parameters

ev	Enum variant to print.
indent	Indent level.

8.51.1.7 ap_print_expr()

```
func ap_print_expr(id:  ExprId, pr:  ParseResult*, indent:  int)
```

Print an expression node by arena id.

Recursively prints child expressions and child type references.

Parameters

id	Expression id in the arena.
pr	Parse result providing the expression arena.
indent	Indent level.

8.51.1.8 ap_print_field_decl()

```
func ap_print_field_decl(fd:  FieldDecl*, indent:  int)
```

Print a field declaration node.

Parameters

fd	Field declaration to print.
indent	Indent level.

8.51.1.9 ap_print_func_decl()

```
func ap_print_func_decl(fd:  FuncDecl*, pr:  ParseResult*, indent:  int)
```

Print a function declaration including params, return type, and body.

Parameters

fd	Function declaration to print.
pr	Parse result providing statement arenas.
indent	Indent level.

8.51.1.10 ap_print_import()

```
func ap_print_import(imp:  Import*, indent:  int)
```

Print an import node.

Parameters

imp	Import to print.
indent	Indent level.

8.51.1.11 ap_print_label()

```
func ap_print_label(indent:  int, name:  string)
```

Print a child-field label at `indent + 1`.

Parameters

indent	Parent node indent level.
name	Label name without colon.

8.51.1.12 ap_print_let_decl()

```
func ap_print_let_decl(ld:  LetDecl*, pr:  ParseResult*, indent:  int)
```

Print a top-level `let` declaration.

Parameters

ld	Let declaration to print.
pr	Parse result providing expression arenas.
indent	Indent level.

8.51.1.13 ap_print_line()

```
func ap_print_line(indent:  int, text:  string)
```

Print a line at the given indent level.

Parameters

indent	Indent level (2 spaces each).
text	Text to print after the indentation.

8.51.1.14 ap_print_module()

```
func ap_print_module(mod:  Module*, pr:  ParseResult*)
```

Print the AST of a parsed module.

Mirrors the Stage 1 `format_module()` output contract:

- Module header with scalar fields and span on one line.
- `imports:` label and each Import child at indent 2.
- `decls:` label and each top-level declaration at indent 2.
- Empty imports or decls lists are omitted (no label printed).

Parameters

mod	Module to print.
pr	Parse result providing all AST arenas (expr, stmt, pattern).

8.51.1.15 ap_print_param()

```
func ap_print_param(p:  Param*, indent:  int)
```

Print a parameter node, including its type annotation.

Parameters

p	Parameter to print.
indent	Indent level.

8.51.1.16 ap_print_pattern()

```
func ap_print_pattern(id:  PatternId, pr:  ParseResult*, indent:  int)
```

Print a pattern node by arena id.

Parameters

id	Pattern id in the arena.
pr	Parse result providing the pattern arena.
indent	Indent level.

8.51.1.17 ap_print_stmt()

```
func ap_print_stmt(id:  StmtId, pr:  ParseResult*, indent:  int)
```

Print a statement node by arena id.

Recursively prints child statements, expressions, and patterns.

Parameters

id	Statement id in the arena.
pr	Parse result providing all AST arenas.
indent	Indent level.

8.51.1.18 ap_print_struct_decl()

```
func ap_print_struct_decl(sd:  StructDecl*, indent:  int)
```

Print a struct declaration including its fields.

Parameters

sd	Struct declaration to print.
indent	Indent level.

8.51.1.19 ap_print_top_level_decl()

```
func ap_print_top_level_decl(tld:  TopLevelDecl*, pr:  ParseResult*, indent:  int)
```

Dispatch to the correct per-kind declaration printer.

Parameters

tld	Top-level declaration wrapper.
pr	Parse result providing all AST arenas.
indent	Indent level.

8.51.1.20 ap_print_type_alias_decl()

```
func ap_print_type_alias_decl(ta:  TypeAliasDecl*, indent:  int)
```

Print a type-alias declaration.

Parameters

ta	Type-alias declaration to print.
indent	Indent level.

8.51.1.21 ap_print_type_ref()

```
func ap_print_type_ref(tr:  TypeRef*, indent:  int)
```

Print a type reference node.

Parameters

tr	Type reference to print.
indent	Indent level.

8.51.1.22 ap_quote_string()

```
func ap_quote_string(s:  string) → string
```

Return a single-quoted escaped representation of a string.

Parameters

s	String to quote.

Returns

Single-quoted escaped string.

8.51.1.23 ap_span_suffix()

```
func ap_span_suffix(span:  Span) → string
```

Build a span-suffix annotation string `@sl:sc-el:ec`.

Parameters

span	Source span.

Returns

Formatted span annotation.

8.52 compiler/stage2_l0/src/backend.l0 File Reference

Module: `backend`

```
import util.log;
import types;
import type_resolve;
import scope_context;
import std.assert;
import std.string;
import std.hashmap;
import std.hashset;
import c_emitter;
import driver;
import std.vector;
import std.text;
import ast;
import codegen_options;
import name_resolver;
import sem_context;
import symbols;
import signatures;
```

Classes

- struct Backend

  Represents one Stage 2 backend generation session.
- struct LoopLabels

  Represents one active loop label pair.
- struct LoopCleanupScopes

  Represents the cleanup targets for one active loop.

Functions

- func be_ice(message: string)

  Abort on an internal backend error.
- func be_ice_string(message: string) → string

  Abort on an internal backend error from a string-returning helper.
- func be_create(result: AnalysisResult∗, opts: CodegenOptions∗, cfg: LogConfig∗) → Backend∗

  Create a backend state for one code-generation pass.
- func be_free(self: Backend∗)

  Free a backend state.
- func be_type_key(module_name: string, type_name: string) → string

  Build a stable key for one user-defined type.
- func be_type_key_module(key: string) → string

  Return the module portion of a type key.
- func be_type_key_name(key: string) → string

  Return the type-name portion of a type key.
- func be_current_env(self: Backend∗) → ModuleEnv∗

  Return the current module environment.
- func be_current_unit(self: Backend∗) → DriverUnit∗

  Return the current unit.
- func be_current_expr_arena(self: Backend∗) → ExprArena∗

  Return the current expression arena.
- func be_current_stmt_arena(self: Backend∗) → StmtArena∗

  Return the current statement arena.
- func be_find_unit(self: Backend∗, module_name: string) → DriverUnit∗?

  Return a loaded driver unit by module name.
- func be_find_env(self: Backend∗, module_name: string) → ModuleEnv∗?

  Return a module environment by name.
- func be_resolve_type_ref(self: Backend∗, tref: TypeRef∗) → Type∗?

  Resolve one parsed type reference in the current module environment.
- func be_lookup_symbol(self: Backend∗, name: string, module_path: StringVector∗?, name_qualifier: StringVector∗?) → Symbol∗?

  Look up a visible symbol from the current module.
- func be_expr_type(self: Backend∗, expr_id: ExprId) → Type∗?

  Return the inferred type of an expression.
- func be_type_ptr_vec_temp_free(items: VectorBase∗?)

  Free a temporary vector of optional owned `Type∗` values.
- func be_type_ptr_map_temp_free(items: StringPtrMap∗?)

  Free a temporary map of owned `Type∗` values.
- func be_resolve_let_type(self: Backend∗, stmt: StmtNode∗) → Type∗

  Resolve the type of one local `let` declaration.

- func be_emit_with_cleanup_header_let_predecl(self: Backend*, stmt: StmtNode*) → Type*?

  Predeclare a nullable cleanup-block header let to its null literal.
- func be_emit_with_cleanup_header_let_assign(self: Backend*, stmt: StmtNode*, var_ty: Type*)

  Emit the initializer assignment for a predeclared cleanup-block header let.
- func be_fresh_label(self: Backend*, prefix: string) → string

  Return a fresh backend label name.
- func be_push_scope(self: Backend*) → ScopeContext*

  Push one lexical scope and make it current.
- func be_pop_scope(self: Backend*)

  Pop the current lexical scope.
- func be_push_loop_labels(self: Backend*, break_label: string, continue_label: string)

  Push one active loop-label frame.
- func be_pop_loop_labels(self: Backend*)

  Pop one active loop-label frame.
- func be_push_loop_cleanup_scopes(self: Backend*, continue_scope: ScopeContext*, break_scope: ScopeContext*)

  Push one active loop-cleanup frame.
- func be_pop_loop_cleanup_scopes(self: Backend*)

  Pop one active loop-cleanup frame.
- func be_lookup_local_var_type(self: Backend*, var_name: string) → Type*?

  Return the type of a declared local binding when present.
- func be_lookup_owned_local_name(self: Backend*, expr_id: ExprId) → string?

  Return the mangled name when a local variable is currently tracked as owned.
- func be_is_place_expr(self: Backend*, expr_id: ExprId) → bool

  Report whether an expression refers to an existing place.
- func be_has_side_effects(self: Backend*, expr_id: ExprId) → bool

  Report whether an expression may have side effects when evaluated.
- func be_pointer_type_or_null(ty: Type*?) → Type*?

  Return the represented pointer type for pointer-shaped values.
- func be_sizeof_expr_for_type(ty: Type*) → string

  Return a C sizeof expression for the runtime access extent.
- func be_alignof_expr_for_type(ty: Type*) → string

  Return a C alignment expression for the runtime access target.
- func be_emit_checked_pointer_expr(self: Backend*, c_ptr_expr: string, ptr_ty: Type*, access_mode: string) → string

  Emit a pointer expression checked for one pointee-sized access.
- func be_emit_pointer_index_lvalue(self: Backend*, c_base: string, c_index: string, base_ty: Type*, access_mode: string) → string

  Emit a checked pointer-index lvalue expression.
- func be_is_unwrap_cast_from_place(self: Backend*, expr_id: ExprId) → bool

  Report whether an expression is a borrowed cast from a place.
- func be_needs_arc_temp(self: Backend*, expr_id: ExprId) → bool

  Report whether a non-place ARC rvalue needs temporary materialization.
- func be_should_materialize_arc_temp(self: Backend*, expr_id: ExprId, expr_type: Type*) → bool

  Report whether an expression should be materialized into a scope-owned temp.
- func be_materialize_arc_temp(self: Backend*, c_expr: string, expr_type: Type*) → string

  Materialize one ARC rvalue into a scope-owned temporary.
- func be_scope_chain_has_cleanup(self: Backend*) → bool

  Report whether any scope in the current chain has pending cleanup work.
- func be_emit_retain_for_copied_value(self: Backend*, c_expr: string, ty: Type*)

  Emit retain operations for one copied owned value.

- func be_emit_copy_expr_with_retains(self: Backend*, c_expr: string, ty: Type*) → string

  Materialize one copied value into a temp and retain any owned fields.
- func be_is_int_assignable(ty: Type*?) → bool

  Return whether a type behaves like an integer.
- func be_int_type_size(ty: Type*) → int

  Return the byte width of an integer builtin.
- func be_convert_expr_with_expected_type(c_expr: string, natural_ty: Type*?, expected: Type*) → string

  Convert a lowered expression to an expected type when required.
- func be_emit_expr_with_expected_type(self: Backend*, expr_id: ExprId, expected: Type*) → string

  Emit one expression with implicit conversion to an expected type.
- func be_emit_owned_expr_with_expected_type(self: Backend*, expr_id: ExprId, expected: Type*) → string

  Emit one expression for a context that creates a new owner.
- func type_free_opt(ty: Type*?)

  Free an optional cloned type.
- func be_find_struct_decl(self: Backend*, module_name: string, struct_name: string) → StructDecl*?

  Look up a top-level struct declaration.
- func be_find_enum_decl(self: Backend*, module_name: string, enum_name: string) → EnumDecl*?

  Look up a top-level enum declaration.
- func be_find_variant_decl(self: Backend*, module_name: string, enum_name: string, variant_name: string) → EnumVariant*?

  Look up an enum variant declaration.
- func be_collect_value_type_dependencies(ty: Type*, deps: StringSet*)

  Add value-type dependencies for one field type.
- func be_collect_type_dependencies(self: Backend*, type_key: string, deps: StringSet*) → bool

  Collect the value-type dependencies for one user-defined type.
- func be_dependency_sets_free(sets: VectorBase*?)

  Free a vector of dependency sets.
- func be_collect_type_keys(self: Backend*) → StringVector*

  Collect all user-defined types in stable source order.
- func be_emit_type_definitions(self: Backend*)

  Emit user-defined types in dependency order.
- func be_emit_line_directive(self: Backend*, span: Span)

  Emit a `#line` directive for a declaration or statement when enabled.
- func be_emit_bare_variant_static_initializer(self: Backend*, expr_id: ExprId, expected_type: Type*) → string

  Emit a bare zero-field enum variant as a static initializer.
- func be_is_static_let_initializer(self: Backend*, expr_id: ExprId) → bool

  Check whether an expression belongs to the supported static-initializer subset.
- func be_static_variant_symbol(self: Backend*, expr_id: ExprId) → Symbol*?

  Resolve the enum-variant symbol named by one static constructor expression.
- func be_emit_static_ord_initializer(self: Backend*, expr_id: ExprId) → string

  Emit a file-scope `ord` initializer as a direct C enum-tag constant.
- func be_emit_let_initializer(self: Backend*, expr_id: ExprId, expected_type: Type*) → string

  Emit a constant initializer for a top-level `let`.
- func be_emit_const_constructor(self: Backend*, expr_id: ExprId, expected_type: Type*) → string

  Emit a constant constructor initializer for a top-level `let`.
- func be_emit_let_declarations(self: Backend*)

  Emit all top-level `let` declarations.
- func be_emit_function_declarations(self: Backend*)

  Emit all function declarations.
- func be_emit_lvalue(self: Backend*, expr_id: ExprId) → string

  Emit one local or module lvalue.

- func be_emit_binary_op(self: Backend*, expr_id: ExprId) → string

  Emit one binary operation.
- func be_emit_condition_expr(self: Backend*, expr_id: ExprId) → string

  Emit one top-level condition expression for direct statement headers.
- func be_emit_constructor_call(self: Backend*, expr_id: ExprId, sym: Symbol*, result_ty: Type*) → string

  Emit one struct or enum-variant constructor.
- func be_emit_call(self: Backend*, expr_id: ExprId) → string

  Emit one call expression.
- func be_emit_new_expr(self: Backend*, expr_id: ExprId) → string

  Emit one heap-allocation `new` expression.
- func be_emit_expr(self: Backend*, expr_id: ExprId) → string

  Emit one expression.
- func be_emit_value_cleanup(self: Backend*, c_expr: string, ty: Type*)

  Emit cleanup for one by-value expression.
- func be_emit_struct_cleanup(self: Backend*, c_ptr_expr: string, struct_type: Type*)

  Emit cleanup for a pointer-to-struct expression.
- func be_emit_enum_cleanup(self: Backend*, c_ptr_expr: string, enum_type: Type*)

  Emit cleanup for a pointer-to-enum expression.
- func be_emit_cleanup_at_scope_exit(self: Backend*, scope: ScopeContext*)

  Emit cleanup for owned values declared in one scope.
- func be_emit_with_cleanup_from_scope(self: Backend*, scope: ScopeContext*)

  Emit `with` cleanup recorded on one scope.
- func be_register_inline_with_cleanup(scope: ScopeContext*, item: WithItem*)

  Register one inline `with` cleanup item in LIFO order.
- func be_emit_cleanup_for_return(self: Backend*, returned_var: string?)

  Emit cleanup required before returning from the current function.
- func be_emit_cleanup_for_loop_exit(self: Backend*, is_break: bool)

  Emit cleanup required before breaking or continuing out of the innermost loop.
- func be_emit_return_stmt_with_inline_cleanup(self: Backend*, stmt: StmtNode*, inline_scope: ScopeContext*?, inline_cleanup: StmtId)

  Emit a return statement, optionally registering one inline with-item cleanup first.
- func be_emit_pattern_bindings(self: Backend*, pattern_id: PatternId, enum_type: Type*, arm_scope: ScopeContext*)

  Emit one match-pattern payload binding set.
- func be_emit_case_literal(self: Backend*, expr_id: ExprId) → string

  Emit one `case` literal.
- func be_emit_drop_stmt(self: Backend*, stmt: StmtNode*)

  Emit a `drop` statement.
- func be_emit_match_stmt(self: Backend*, stmt: StmtNode*)

  Emit a `match` statement.
- func be_emit_case_stmt(self: Backend*, stmt: StmtNode*)

  Emit a `case` statement.
- func be_emit_inline_with_header_item(self: Backend*, item: WithItem*, with_scope: ScopeContext*)

  Emit one inline `with` header item and register cleanup at the committed point.
- func be_emit_with_stmt(self: Backend*, stmt: StmtNode*)

  Emit a `with` statement.
- func be_emit_block_sequence(self: Backend*, block_id: StmtId)

  Emit one block body sequence without outer braces.
- func be_emit_block_stmt(self: Backend*, stmt_id: StmtId)

  Emit one lexical block statement.
- func be_emit_condition_branch(self: Backend*, expr_id: ExprId, true_label: string, false_label: string)

Emit control flow for one statement condition with short-circuit semantics.

- func be_emit_condition_value(self: Backend∗, expr_id: ExprId) → string

  Evaluate one statement condition into a stable boolean temp.
- func be_emit_if_branch(self: Backend∗, stmt_id: StmtId) → bool

  Emit one `if` branch and report whether it ends unreachable.
- func be_emit_stmt(self: Backend∗, stmt_id: StmtId)

  Emit one statement.
- func be_body_reassigns_param(self: Backend∗, stmt_id: StmtId, param_name: string) → bool

  Check whether the statement sub-tree at `stmt_id` syntactically reassigns `param_name` via a bare `VarRef` target of an `ST_ASSIGN`.
- func be_emit_function_definitions(self: Backend∗)

  Emit all non-extern function definitions.
- func be_emit_main_wrapper_if_needed(self: Backend∗)

  Emit the C `main()` wrapper when the entry module defines `main`.
- func backend_generate(result: AnalysisResult∗, opts: CodegenOptions∗, cfg: LogConfig∗) → string

  Generate complete C source from a typed analysis result.

8.52.1 Function Documentation

8.52.1.1 backend_generate()

```
func backend_generate(result: AnalysisResult*, opts: CodegenOptions*, cfg: LogConfig*) →
string
```

Generate complete C source from a typed analysis result.

Parameters

result	Typed analysis result to lower.
opts	Codegen options to honor.
cfg	Logging configuration to use.

Returns

Complete generated C translation unit.

8.52.1.2 be_alignof_expr_for_type()

```
func be_alignof_expr_for_type(ty: Type*) → string
```

Return a C alignment expression for the runtime access target.

Parameters

ty	Type whose alignment is required.

Returns

C expression string.

8.52.1.3 be_body_reassigns_param()

```
func be_body_reassigns_param(self: Backend*, stmt_id: StmtId, param_name: string) → bool
```

Check whether the statement sub-tree at `stmt_id` syntactically reassigns `param_name` via a bare `VarRef` target of an `ST_ASSIGN`.

Used by the function-prologue emitter to decide whether to defensively retain an ARC-typed parameter on entry.

Parameters

self	Backend state.
stmt_id	Root statement id, or negative when absent.
param_name	Source-level parameter name to match against assignment targets.

Returns

`true` when any reachable `ST_ASSIGN` targets a bare `VarRef` with that name.

8.52.1.4 be_collect_type_dependencies()

```
func be_collect_type_dependencies(self: Backend*, type_key: string, deps: StringSet*) →
bool
```

Collect the value-type dependencies for one user-defined type.

Parameters

self	Backend state to inspect.
type_key	Type key to inspect.
deps	Destination set to populate.

Returns

`true` when the type metadata was found.

8.52.1.5 be_collect_type_keys()

```
func be_collect_type_keys(self: Backend*) → StringVector*
```

Collect all user-defined types in stable source order.

Parameters

self	Backend state to inspect.

Returns

Ordered vector of type keys.

8.52.1.6 be_collect_value_type_dependencies()

```
func be_collect_value_type_dependencies(ty: Type*, deps: StringSet*)
```

Add value-type dependencies for one field type.

Parameters

ty	Type to inspect.
deps	Dependency set to update.

8.52.1.7 be_convert_expr_with_expected_type()

```
func be_convert_expr_with_expected_type(c_expr: string, natural_ty: Type*?, expected: Type*)
→ string
```

Convert a lowered expression to an expected type when required.

Parameters

c_expr	Lowered source expression.
natural_ty	Natural source type.
expected	Expected destination type.

Returns

Lowered expression converted as needed.

8.52.1.8 be_create()

```
func be_create(result: AnalysisResult*, opts: CodegenOptions*, cfg: LogConfig*) → Backend*
```

Create a backend state for one code-generation pass.

Parameters

result	Borrowed semantic analysis result.
opts	Borrowed codegen options.
cfg	Borrowed log configuration.

Returns

New backend state.

8.52.1.9 be_current_env()

```
func be_current_env(self:  Backend*) → ModuleEnv*
```

Return the current module environment.

Parameters

self	Backend state to inspect.

Returns

Borrowed module environment.

8.52.1.10 be_current_expr_arena()

```
func be_current_expr_arena(self:  Backend*) → ExprArena*
```

Return the current expression arena.

Parameters

self	Backend state to inspect.

Returns

Borrowed expression arena.

8.52.1.11 be_current_stmt_arena()

```
func be_current_stmt_arena(self:  Backend*) → StmtArena*
```

Return the current statement arena.

Parameters

self	Backend state to inspect.

Returns

Borrowed statement arena.

8.52.1.12 be_current_unit()

```
func be_current_unit(self:  Backend*) → DriverUnit*
```

Return the current unit.

Parameters

self	Backend state to inspect.

Returns

Borrowed driver unit.

8.52.1.13 be_dependency_sets_free()

```
func be_dependency_sets_free(sets:  VectorBase*?)
```

Free a vector of dependency sets.

Parameters

sets	Optional vector holding `StringSet*` entries.

8.52.1.14 be_emit_bare_variant_static_initializer()

```
func be_emit_bare_variant_static_initializer(self:  Backend*, expr_id:  ExprId, expected_type:
Type*) → string
```

Emit a bare zero-field enum variant as a static initializer.

Parameters

self	Backend state to inspect.
expr_id	Var-ref expression id.
expected_type	Resolved binding type.

Returns

Static enum initializer, or an ICE string when invariants are broken.

8.52.1.15 be_emit_binary_op()

```
func be_emit_binary_op(self:  Backend*, expr_id:  ExprId) → string
```

Emit one binary operation.

Parameters

self	Backend state to inspect.
expr↩ _id	Binary expression id.

Returns

Lowered binary expression.

8.52.1.16 be_emit_block_sequence()

```
func be_emit_block_sequence(self:  Backend*, block_id:  StmtId)
```

Emit one block body sequence without outer braces.

Parameters

self	Backend state to update.
block↩ _id	Block statement id.

8.52.1.17 be_emit_block_stmt()

```
func be_emit_block_stmt(self:  Backend*, stmt_id:  StmtId)
```

Emit one lexical block statement.

Parameters

self	Backend state to update.
stmt↩ _id	Block statement id.

8.52.1.18 be_emit_call()

```
func be_emit_call(self:  Backend*, expr_id:  ExprId) → string
```

Emit one call expression.

Parameters

self	Backend state to inspect.
expr↩ _id	Call expression id.

Returns

Lowered call expression.

8.52.1.19 be_emit_case_literal()

```
func be_emit_case_literal(self: Backend*, expr_id: ExprId) → string
```

Emit one `case` literal.

Parameters

self	Backend state to inspect.
expr_id	Literal expression id.

Returns

Lowered constant expression.

8.52.1.20 be_emit_case_stmt()

```
func be_emit_case_stmt(self: Backend*, stmt: StmtNode*)
```

Emit a `case` statement.

Parameters

self	Backend state to update.
stmt	Parsed case statement.

8.52.1.21 be_emit_checked_pointer_expr()

```
func be_emit_checked_pointer_expr(self: Backend*, c_ptr_expr: string, ptr_ty: Type*, access_mode:
string) → string
```

Emit a pointer expression checked for one pointee-sized access.

Parameters

self	Backend state to inspect.
c_ptr_expr	Pointer expression.
ptr_ty	L0 pointer-shaped type.
access_mode	Runtime access mode expression.

Returns

Checked pointer expression.

8.52.1.22 be_emit_cleanup_at_scope_exit()

```
func be_emit_cleanup_at_scope_exit(self: Backend*, scope: ScopeContext*)
```

Emit cleanup for owned values declared in one scope.

Parameters

self	Backend state to update.
scope	Scope to clean.

8.52.1.23 be_emit_cleanup_for_loop_exit()

```
func be_emit_cleanup_for_loop_exit(self: Backend*, is_break: bool)
```

Emit cleanup required before breaking or continuing out of the innermost loop.

Parameters

self	Backend state to update.
is_break	`true` for break, `false` for continue.

8.52.1.24 be_emit_cleanup_for_return()

```
func be_emit_cleanup_for_return(self: Backend*, returned_var: string?)
```

Emit cleanup required before returning from the current function.

Parameters

self	Backend state to update.
returned_var	Optional owned local to keep alive across cleanup.

8.52.1.25 be_emit_condition_branch()

```
func be_emit_condition_branch(self: Backend*, expr_id: ExprId, true_label: string, false_label:
string)
```

Emit control flow for one statement condition with short-circuit semantics.

ARC temps produced while lowering condition leaves stay inside the emitted condition block instead of being hoisted into an enclosing control-flow header.

Parameters

self	Backend state to update.
expr_id	Condition expression to lower.
true_label	Jump target when the condition is true.
false_label	Jump target when the condition is false.

8.52.1.26 be_emit_condition_expr()

```
func be_emit_condition_expr(self: Backend*, expr_id: ExprId) → string
```

Emit one top-level condition expression for direct statement headers.

Only the outermost condition operator is unwrapped; child expressions keep the normal expression emitter so precedence-preserving inner parentheses stay intact.

Parameters

self	Backend state to inspect.
expr↩ _id	Expression id to lower.

Returns

Lowered condition expression.

8.52.1.27 be_emit_condition_value()

```
func be_emit_condition_value(self: Backend*, expr_id: ExprId) → string
```

Evaluate one statement condition into a stable boolean temp.

The returned temp is safe to reference from an `if`/`while` header because condition ARC temps are scoped to the emitted condition block and cleaned before control continues.

Parameters

self	Backend state to update.
expr↩ _id	Condition expression to lower.

Returns

Name of the generated boolean temp.

8.52.1.28 be_emit_const_constructor()

```
func be_emit_const_constructor(self: Backend*, expr_id: ExprId, expected_type: Type*) →
string
```

Emit a constant constructor initializer for a top-level `let`.

Parameters

self	Backend state to inspect.
expr_id	Call expression id.
expected_type	Expected constructed type.

Returns

Constant initializer expression.

8.52.1.29 be_emit_constructor_call()

```
func be_emit_constructor_call(self: Backend*, expr_id: ExprId, sym: Symbol*, result_ty:
Type*) → string
```

Emit one struct or enum-variant constructor.

Parameters

self	Backend state to inspect.
expr_id	Call expression id.
sym	Resolved constructor symbol.
result↩ _ty	Inferred result type of the call.

Returns

Lowered constructor expression.

8.52.1.30 be_emit_copy_expr_with_retains()

```
func be_emit_copy_expr_with_retains(self: Backend*, c_expr: string, ty: Type*) → string
```

Materialize one copied value into a temp and retain any owned fields.

Parameters

self	Backend state to update.
c_expr	Lowered source expression.
ty	Type of the copied value.

Returns

Lowered expression or new temp name when retain logic is needed.

8.52.1.31 be_emit_drop_stmt()

```
func be_emit_drop_stmt(self: Backend*, stmt: StmtNode*)
```

Emit a `drop` statement.

Parameters

self	Backend state to update.
stmt	Parsed drop statement.

8.52.1.32 be_emit_enum_cleanup()

```
func be_emit_enum_cleanup(self: Backend*, c_ptr_expr: string, enum_type: Type*)
```

Emit cleanup for a pointer-to-enum expression.

Parameters

self	Backend state to update.
c_ptr_expr	Lowered pointer expression.
enum_type	Enum type to clean.

8.52.1.33 be_emit_expr()

```
func be_emit_expr(self: Backend*, expr_id: ExprId) → string
```

Emit one expression.

Parameters

self	Backend state to inspect.
expr↩ _id	Expression id to lower.

Returns

Lowered expression string.

8.52.1.34 be_emit_expr_with_expected_type()

```
func be_emit_expr_with_expected_type(self: Backend*, expr_id: ExprId, expected: Type*) →
string
```

Emit one expression with implicit conversion to an expected type.

Parameters

self	Backend state to inspect.
expr_id	Expression to lower.
expected	Expected destination type.

Returns

Lowered expression string.

8.52.1.35 be_emit_function_declarations()

```
func be_emit_function_declarations(self:  Backend*)
```

Emit all function declarations.

Parameters

self	Backend state to update.

8.52.1.36 be_emit_function_definitions()

```
func be_emit_function_definitions(self:  Backend*)
```

Emit all non-extern function definitions.

Parameters

self	Backend state to update.

8.52.1.37 be_emit_if_branch()

```
func be_emit_if_branch(self:  Backend*, stmt_id:  StmtId) → bool
```

Emit one `if` branch and report whether it ends unreachable.

Parameters

self	Backend state to update.
stmt↩ _id	Statement id for the branch.

Returns

`true` when control cannot fall through the branch.

8.52.1.38 be_emit_inline_with_header_item()

```
func be_emit_inline_with_header_item(self: Backend*, item: WithItem*, with_scope: ScopeContext*)
```

Emit one inline `with` header item and register cleanup at the committed point.

Parameters

self	Backend state to update.
item	Parsed inline with item.
with_scope	Scope carrying inline cleanup state.

8.52.1.39 be_emit_let_declarations()

```
func be_emit_let_declarations(self: Backend*)
```

Emit all top-level `let` declarations.

Parameters

self	Backend state to update.

8.52.1.40 be_emit_let_initializer()

```
func be_emit_let_initializer(self: Backend*, expr_id: ExprId, expected_type: Type*) → string
```

Emit a constant initializer for a top-level `let`.

Parameters

self	Backend state to inspect.
expr_id	Expression id to lower.
expected_type	Resolved `let` type.

Returns

Constant initializer expression.

8.52.1.41 be_emit_line_directive()

```
func be_emit_line_directive(self: Backend*, span: Span)
```

Emit a `#line` directive for a declaration or statement when enabled.

Parameters

self	Backend state to update.
span	Source span to report.

8.52.1.42 be_emit_lvalue()

```
func be_emit_lvalue(self: Backend*, expr_id: ExprId) → string
```

Emit one local or module lvalue.

Parameters

self	Backend state to inspect.
expr↩ _id	Target expression id.

Returns

Lowered lvalue expression.

8.52.1.43 be_emit_main_wrapper_if_needed()

```
func be_emit_main_wrapper_if_needed(self: Backend*)
```

Emit the C `main()` wrapper when the entry module defines `main`.

Parameters

self	Backend state to update.

8.52.1.44 be_emit_match_stmt()

```
func be_emit_match_stmt(self: Backend*, stmt: StmtNode*)
```

Emit a `match` statement.

Parameters

self	Backend state to update.
stmt	Parsed match statement.

8.52.1.45 be_emit_new_expr()

```
func be_emit_new_expr(self: Backend*, expr_id: ExprId) → string
```

Emit one heap-allocation `new` expression.

Parameters

self	Backend state to inspect.
expr↩ _id	New-expression id.

Returns

Lowered pointer expression.

8.52.1.46 be_emit_owned_expr_with_expected_type()

```
func be_emit_owned_expr_with_expected_type(self: Backend*, expr_id: ExprId, expected: Type*)
→ string
```

Emit one expression for a context that creates a new owner.

Parameters

self	Backend state to inspect.
expr_id	Expression to lower.
expected	Expected destination type.

Returns

Lowered expression string, with retain-on-copy when required.

8.52.1.47 be_emit_pattern_bindings()

```
func be_emit_pattern_bindings(self: Backend*, pattern_id: PatternId, enum_type: Type*,
arm_scope: ScopeContext*)
```

Emit one match-pattern payload binding set.

Parameters

self	Backend state to update.
pattern_id	Pattern id to inspect.
enum_type	Scrutinee enum type.
arm_scope	Arm scope receiving borrowed bindings.

8.52.1.48 be_emit_pointer_index_lvalue()

```
func be_emit_pointer_index_lvalue(self: Backend*, c_base: string, c_index: string, base_ty:
Type*, access_mode: string) → string
```

Emit a checked pointer-index lvalue expression.

Parameters

self	Backend state to inspect.
c_base	Base pointer expression.
c_index	Index expression.
base_ty	L0 pointer-shaped base type.
access_mode	Runtime access mode expression.

Returns

Checked lvalue expression.

8.52.1.49 be_emit_retain_for_copied_value()

```
func be_emit_retain_for_copied_value(self: Backend*, c_expr: string, ty: Type*)
```

Emit retain operations for one copied owned value.

Parameters

self	Backend state to update.
c_expr	Lowered expression containing the copied value.
ty	Type of the copied value.

8.52.1.50 be_emit_return_stmt_with_inline_cleanup()

```
func be_emit_return_stmt_with_inline_cleanup(self: Backend*, stmt: StmtNode*, inline_scope:
ScopeContext*?, inline_cleanup: StmtId)
```

Emit a return statement, optionally registering one inline with-item cleanup first.

Parameters

self	Backend state to update.
stmt	Parsed return statement.
inline_scope	Optional with scope for a committed inline header return.
inline_cleanup	Cleanup statement to register when inline_scope is present.

8.52.1.51 be_emit_static_ord_initializer()

```
func be_emit_static_ord_initializer(self: Backend*, expr_id: ExprId) → string
```

Emit a file-scope `ord` initializer as a direct C enum-tag constant.

Parameters

self	Backend state to inspect.
expr↩ _id	Static `ord` call expression.

Returns

C99 constant expression for the resolved variant tag.

8.52.1.52 be_emit_stmt()

```
func be_emit_stmt(self:  Backend*, stmt_id:  StmtId)
```

Emit one statement.

Parameters

self	Backend state to update.
stmt↩ _id	Statement id to lower.

8.52.1.53 be_emit_struct_cleanup()

```
func be_emit_struct_cleanup(self:  Backend*, c_ptr_expr:  string, struct_type:  Type*)
```

Emit cleanup for a pointer-to-struct expression.

Parameters

self	Backend state to update.
c_ptr_expr	Lowered pointer expression.
struct_type	Struct type to clean.

8.52.1.54 be_emit_type_definitions()

```
func be_emit_type_definitions(self:  Backend*)
```

Emit user-defined types in dependency order.

Parameters

self	Backend state to update.

8.52.1.55 be_emit_value_cleanup()

```
func be_emit_value_cleanup(self: Backend*, c_expr: string, ty: Type*)
```

Emit cleanup for one by-value expression.

Parameters

self	Backend state to update.
c_expr	Lowered value expression.
ty	Type of the value.

8.52.1.56 be_emit_with_cleanup_from_scope()

```
func be_emit_with_cleanup_from_scope(self: Backend*, scope: ScopeContext*)
```

Emit `with` cleanup recorded on one scope.

Parameters

self	Backend state to update.
scope	Scope carrying `with` cleanup state.

8.52.1.57 be_emit_with_cleanup_header_let_assign()

```
func be_emit_with_cleanup_header_let_assign(self: Backend*, stmt: StmtNode*, var_ty: Type*)
```

Emit the initializer assignment for a predeclared cleanup-block header let.

Parameters

self	Backend state to update.
stmt	Parsed `let` statement from a `with` header.
var_ty	Resolved type of the predeclared variable.

8.52.1.58 be_emit_with_cleanup_header_let_predecl()

```
func be_emit_with_cleanup_header_let_predecl(self: Backend*, stmt: StmtNode*) → Type*?
```

Predeclare a nullable cleanup-block header let to its null literal.

Parameters

self	Backend state to update.
stmt	Parsed `let` statement from a `with` header.

Returns

Resolved nullable type when predeclared, otherwise `null`.

8.52.1.59 be_emit_with_stmt()

```
func be_emit_with_stmt(self: Backend*, stmt: StmtNode*)
```

Emit a `with` statement.

Parameters

self	Backend state to update.
stmt	Parsed with statement.

8.52.1.60 be_expr_type()

```
func be_expr_type(self: Backend*, expr_id: ExprId) → Type*?
```

Return the inferred type of an expression.

Parameters

self	Backend state to inspect.
expr↩ _id	Expression id to query.

Returns

Cloned inferred type, or `null` if missing.

8.52.1.61 be_find_enum_decl()

```
func be_find_enum_decl(self: Backend*, module_name: string, enum_name: string) → EnumDecl*?
```

Look up a top-level enum declaration.

Parameters

self	Backend state to inspect.
module_name	Defining module name.
enum_name	Enum name.

Returns

Matching declaration, or `null` if absent.

8.52.1.62 be_find_env()

```
func be_find_env(self:  Backend*, module_name:  string) → ModuleEnv*?
```

Return a module environment by name.

Parameters

self	Backend state to inspect.
module_name	Module to look up.

Returns

Matching module environment, or `null` if absent.

8.52.1.63 be_find_struct_decl()

```
func be_find_struct_decl(self:  Backend*, module_name:  string, struct_name:  string) → StructDecl*?
```

Look up a top-level struct declaration.

Parameters

self	Backend state to inspect.
module_name	Defining module name.
struct_name	Struct name.

Returns

Matching declaration, or `null` if absent.

8.52.1.64 be_find_unit()

```
func be_find_unit(self:  Backend*, module_name:  string) → DriverUnit*?
```

Return a loaded driver unit by module name.

Parameters

self	Backend state to inspect.
module_name	Module to look up.

Returns

Matching loaded unit, or `null` if absent.

8.52.1.65 be_find_variant_decl()

```
func be_find_variant_decl(self: Backend*, module_name: string, enum_name: string, variant_name:
string) → EnumVariant*?
```

Look up an enum variant declaration.

Parameters

self	Backend state to inspect.
module_name	Defining module name.
enum_name	Enum name.
variant_name	Variant name.

Returns

Matching variant declaration, or `null` if absent.

8.52.1.66 be_free()

```
func be_free(self: Backend*)
```

Free a backend state.

Parameters

self	Backend state to free.

8.52.1.67 be_fresh_label()

```
func be_fresh_label(self: Backend*, prefix: string) → string
```

Return a fresh backend label name.

Parameters

self	Backend state to update.
prefix	Label prefix.

Returns

Unique label string.

8.52.1.68 be_has_side_effects()

```
func be_has_side_effects(self: Backend*, expr_id: ExprId) → bool
```

Report whether an expression may have side effects when evaluated.

Parameters

self	Backend state to inspect.
expr↩ _id	Expression to classify.

Returns

`true` when the expression should be cached before reuse.

8.52.1.69 be_ice()

```
func be_ice(message:  string)
```

Abort on an internal backend error.

Parameters

message	ICE message to report.

8.52.1.70 be_ice_string()

```
func be_ice_string(message:  string) → string
```

Abort on an internal backend error from a string-returning helper.

Parameters

message	ICE message to report.

Returns

Dummy return used only to satisfy control-flow typing.

8.52.1.71 be_int_type_size()

```
func be_int_type_size(ty:  Type*) → int
```

Return the byte width of an integer builtin.

Parameters

ty	Type to inspect.

Returns

Byte width for supported integer types.

8.52.1.72 be_is_int_assignable()

```
func be_is_int_assignable(ty: Type*?) → bool
```

Return whether a type behaves like an integer.

Parameters

ty	Type to inspect.

Returns

`true` for `int` and `byte`.

8.52.1.73 be_is_place_expr()

```
func be_is_place_expr(self: Backend*, expr_id: ExprId) → bool
```

Report whether an expression refers to an existing place.

Parameters

self	Backend state to inspect.
expr↩ _id	Expression to classify.

Returns

`true` for local/module places, dereferences, indexes, and fields.

8.52.1.74 be_is_static_let_initializer()

```
func be_is_static_let_initializer(self: Backend*, expr_id: ExprId) → bool
```

Check whether an expression belongs to the supported static-initializer subset.

Parameters

self	Backend state to inspect.
expr↩ _id	Expression id to inspect.

Returns

`true` when the expression can be emitted at C file scope.

8.52.1.75 be_is_unwrap_cast_from_place()

```
func be_is_unwrap_cast_from_place(self: Backend*, expr_id: ExprId) → bool
```

Report whether an expression is a borrowed cast from a place.

Outer parentheses are ignored; owner-producing ARC value-optional wraps are excluded.

Parameters

self	Backend state to inspect.
expr↩ _id	Expression to classify.

Returns

`true` for a non-owner-producing cast whose source is a place expression.

8.52.1.76 be_lookup_local_var_type()

```
func be_lookup_local_var_type(self: Backend*, var_name: string) → Type*?
```

Return the type of a declared local binding when present.

Parameters

self	Backend state to inspect.
var_name	Source-level variable name.

Returns

Borrowed declared type, or `null` if absent.

8.52.1.77 be_lookup_owned_local_name()

```
func be_lookup_owned_local_name(self: Backend*, expr_id: ExprId) → string?
```

Return the mangled name when a local variable is currently tracked as owned.

Parameters

self	Backend state to inspect.
expr↩ _id	Variable-reference expression id.

Returns

Mangled local name, or `null` when not owned by the current scope chain.

8.52.1.78 be_lookup_symbol()

```
func be_lookup_symbol(self: Backend*, name: string, module_path: StringVector*?, name_qualifier:
StringVector*?) → Symbol*?
```

Look up a visible symbol from the current module.

Parameters

self	Backend state to inspect.
name	Symbol name.
module_path	Optional explicit module path.
name_qualifier	Optional enum-name qualifier.

Returns

Matching symbol, or `null` if absent.

8.52.1.79 be_materialize_arc_temp()

```
func be_materialize_arc_temp(self: Backend*, c_expr: string, expr_type: Type*) → string
```

Materialize one ARC rvalue into a scope-owned temporary.

Parameters

self	Backend state to update.
c_expr	Lowered expression to store.
expr_type	Type of the stored expression.

Returns

Temporary variable name.

8.52.1.80 be_needs_arc_temp()

```
func be_needs_arc_temp(self: Backend*, expr_id: ExprId) → bool
```

Report whether a non-place ARC rvalue needs temporary materialization.

Parameters

self	Backend state to inspect.
expr↩ id	Expression to classify.

Returns

`true` when the expression should be hoisted into a cleanup temp.

8.52.1.81 be_pointer_type_or_null()

```
func be_pointer_type_or_null(ty:  Type*?)  → Type*?
```

Return the represented pointer type for pointer-shaped values.

Parameters

ty	Type to inspect.

Returns

Pointer type, or `null` when not pointer-shaped.

8.52.1.82 be_pop_loop_cleanup_scopes()

```
func be_pop_loop_cleanup_scopes(self:  Backend*)
```

Pop one active loop-cleanup frame.

Parameters

self	Backend state to update.

8.52.1.83 be_pop_loop_labels()

```
func be_pop_loop_labels(self:  Backend*)
```

Pop one active loop-label frame.

Parameters

self	Backend state to update.

8.52.1.84 be_pop_scope()

```
func be_pop_scope(self:  Backend*)
```

Pop the current lexical scope.

Parameters

self	Backend state to update.

8.52.1.85 be_push_loop_cleanup_scopes()

```
func be_push_loop_cleanup_scopes(self: Backend*, continue_scope: ScopeContext*, break_scope: ScopeContext*)
```

Push one active loop-cleanup frame.

Parameters

self	Backend state to update.
continue_scope	Scope to clean for `continue`.
break_scope	Scope to clean for `break`.

8.52.1.86 be_push_loop_labels()

```
func be_push_loop_labels(self: Backend*, break_label: string, continue_label: string)
```

Push one active loop-label frame.

Parameters

self	Backend state to update.
break_label	Break target label.
continue_label	Continue target label.

8.52.1.87 be_push_scope()

```
func be_push_scope(self: Backend*) → ScopeContext*
```

Push one lexical scope and make it current.

Parameters

self	Backend state to update.

Returns

New current scope.

8.52.1.88 be_register_inline_with_cleanup()

```
func be_register_inline_with_cleanup(scope: ScopeContext*, item: WithItem*)
```

Register one inline `with` cleanup item in LIFO order.

Parameters

scope	Scope carrying inline cleanup state.
item	Parsed with item whose cleanup should be registered.

8.52.1.89 be_resolve_let_type()

```
func be_resolve_let_type(self: Backend*, stmt: StmtNode*) → Type*
```

Resolve the type of one local `let` declaration.

Parameters

self	Backend state to inspect.
stmt	Parsed `let` statement.

Returns

Resolved type.

8.52.1.90 be_resolve_type_ref()

```
func be_resolve_type_ref(self: Backend*, tref: TypeRef*) → Type*?
```

Resolve one parsed type reference in the current module environment.

Parameters

self	Backend state to inspect.
tref	Parsed type reference.

Returns

Resolved type, or `null` if resolution fails.

8.52.1.91 be_scope_chain_has_cleanup()

```
func be_scope_chain_has_cleanup(self: Backend*) → bool
```

Report whether any scope in the current chain has pending cleanup work.

Parameters

self	Backend state to inspect.

Returns

`true` when owned values or `with` cleanup are present.

8.52.1.92 be_should_materialize_arc_temp()

```
func be_should_materialize_arc_temp(self: Backend*, expr_id: ExprId, expr_type: Type*) →
bool
```

Report whether an expression should be materialized into a scope-owned temp.

Parameters

self	Backend state to inspect.
expr_id	Expression to classify.
expr_type	Inferred type of the expression.

Returns

`true` when the expression carries ARC data and is not already a place.

8.52.1.93 be_sizeof_expr_for_type()

```
func be_sizeof_expr_for_type(ty: Type*) → string
```

Return a C sizeof expression for the runtime access extent.

Parameters

ty	Type whose value extent is required.

Returns

C expression string.

8.52.1.94 be_static_variant_symbol()

```
func be_static_variant_symbol(self: Backend*, expr_id: ExprId) → Symbol*?
```

Resolve the enum-variant symbol named by one static constructor expression.

Parameters

self	Backend state to inspect.
expr↩ _id	Bare or called variant constructor expression.

Returns

Resolved variant symbol, or `null` when the expression is not a variant constructor.

8.52.1.95 be_type_key()

```
func be_type_key(module_name:  string, type_name:  string) → string
```

Build a stable key for one user-defined type.

Parameters

module_name	Defining module name.
type_name	Struct or enum name.

Returns

Key in `module::type` form.

8.52.1.96 be_type_key_module()

```
func be_type_key_module(key:  string) → string
```

Return the module portion of a type key.

Parameters

key	Type key in `module::type` form.

Returns

Module-name portion of the key.

8.52.1.97 be_type_key_name()

```
func be_type_key_name(key:  string) → string
```

Return the type-name portion of a type key.

Parameters

key	Type key in `module::type` form.

Returns

Type-name portion of the key.

8.52.1.98 be_type_ptr_map_temp_free()

```
func be_type_ptr_map_temp_free(items:  StringPtrMap*?)
```

Free a temporary map of owned `Type*` values.

Parameters

items	Map to free.

8.52.1.99 be_type_ptr_vec_temp_free()

```
func be_type_ptr_vec_temp_free(items:  VectorBase*?)
```

Free a temporary vector of optional owned `Type*` values.

Parameters

items	Vector to free. Null entries are ignored.

8.52.1.100 type_free_opt()

```
func type_free_opt(ty:  Type*?)
```

Free an optional cloned type.

Parameters

ty	Optional type to free.

8.53 compiler/stage2_l0/src/build_driver.l0 File Reference

Module: `build_driver`

```
import codegen_options;
import cli_args;
import types;
import symbols;
import signatures;
import name_resolver;
import sem_context;
import backend;
import sys.rt;
import std.path;
import std.hashmap;
import diag_print;
```

```
import source_paths;
import compiler_filesystem;
import std.string;
import util.log;
import std.fs;
import std.system;
import std.text;
import analysis;
import std.vector;
import util.diag;
import util.strings;
```

Classes

- struct BuildPreparedInput

  Prepared input bundle for build/run commands.

Functions

- func bd_prepared_input_free(self: BuildPreparedInput*)

  Free a prepared input bundle.
- func bd_prepared_input_free_opt(self: BuildPreparedInput*?)

  Free an optional prepared input bundle.
- func bd_log_effective_roots(cfg: LogConfig*, sys_roots: StringVector*, project_roots: StringVector*)

  Log the effective system and project roots for build/run commands.
- func bd_prepare_input(opts: CliOptions*, diags: DiagCollector*, cfg: LogConfig*) → BuildPreparedInput*?

  Normalize CLI target input into an entry module and source search paths.
- func bd_append_c_option_words(dst: StringVector*, raw: string?)

  Append whitespace-delimited C option words from one source string.
- func bd_merge_c_option_words(env_opts: string?, cli_opts: string?) → StringVector*

  Merge C option words from environment and CLI with CLI-last precedence.
- func bd_format_word_list(words: StringVector*) → string

  Format option words using the Stage 1 verbose display contract.
- func bd_collect_c_option_words(opts: CliOptions*) → StringVector*

  Resolve merged C option words from `L0_CFLAGS` and parsed CLI options.
- func bd_find_last_sep(path: string) → int

  Replace or add a `.c` suffix on a path string.
- func bd_with_c_suffix(path: string) → string

  Replace or add a `.c` suffix on a path string.
- func bd_is_windows() → bool

  Return whether the current host should use Windows command behavior.
- func bd_exe_suffix() → string

  Return the executable suffix for the current host.
- func bd_default_exe_name() → string

  Return the default build output name for the current host.
- func bd_null_device() → string

  Return the shell null device for the current host shell.
- func bd_shell_quote_posix(arg: string) → string

  Quote one shell token for POSIX `system()` execution.
- func bd_shell_quote_windows(arg: string) → string

Quote one shell token for Windows `system()` execution.

- func bd_shell_quote(arg: string) → string

  Quote one shell token for the active host shell.
- func bd_windows_token_is_plain(arg: string) → bool

  Return whether one Windows shell token can remain unquoted.
- func bd_windows_exec_token(arg: string) → string

  Normalize one Windows executable token for `cmd.exe`.
- func bd_join_shell_words(words: StringVector*) → string

  Join command words into one shell command line.
- func bd_join_display_words(words: StringVector*) → string

  Join command words into the Stage 1 visible command-line form.
- func bd_compiler_flag_family(compiler: string) → string

  Return the compiler flag-family name used by Stage 1.
- func bd_has_explicit_opt_flag(extra_opts: StringVector*) → bool

  Return whether the user already supplied an optimization flag.
- func bd_has_debug_flag(extra_opts: StringVector*) → bool

  Return whether the user supplied a debug flag.
- func bd_get_optimize_flag(family: string, extra_opts: StringVector*) → string?

  Return the default optimization flag for a compiler family.
- func bd_find_c_compiler() → string?

  Return the best available C compiler command or `null`.
- func bd_runtime_include_dir(opts: CliOptions*) → string?

  Resolve the runtime include directory for build/run.
- func bd_runtime_lib_dir(opts: CliOptions*) → string?

  Resolve the runtime library directory for build/run.
- func bd_validate_runtime_lib(runtime_lib_path: string, diags: DiagCollector*) → bool

  Validate that a runtime library directory exists.
- func bd_validate_entry_main(result: AnalysisResult*, entry_module: string, diags: DiagCollector*) → bool

  Validate the entry module `main` function for build/run.
- func bd_push_standard_flags(words: StringVector*, family: string)

  Push compiler-family standard flags into a command-word vector.
- func bd_push_output_flags(words: StringVector*, family: string, exe_path: string)

  Push compiler-family output-path flags into a command-word vector.
- func bd_push_c_compile_command_words(words: StringVector*, compiler: string, family: string, extra_opts: StringVector*, optimize: string?, runtime_include: string?, c_path: string, c_sources: StringVector*, exe_path: string, runtime_lib: string?)

  Push a C compiler invocation into a command-word vector.
- func bd_replay_compile_output(cfg: LogConfig*, stdout_path: string, stderr_path: string, failed: bool)

  Replay compiler output captured into temp files.
- func bd_emit_write_error(diags: DiagCollector*, path: string)

  Emit one Stage 2-only write-file diagnostic.
- func bd_prepare_build_result(opts: CliOptions*, cfg: LogConfig*) → AnalysisResult*?

  Analyze and validate one build/run entry before workspace creation.
- func bd_build_to_exe(opts: CliOptions*, cfg: LogConfig*, result: AnalysisResult*, workspace: CompilerWorkspace*, exe_path: string, keep_c_path: string?) → int

  Build an executable using one command-owned workspace.
- func bd_cmd_build(opts: CliOptions*, cfg: LogConfig*) → int

  Execute Stage 2 `--build`.
- func bd_cmd_run(opts: CliOptions*, cfg: LogConfig*) → int

  Execute Stage 2 `--run`.

8.53.1 Function Documentation

8.53.1.1 bd_append_c_option_words()

```
func bd_append_c_option_words(dst:  StringVector*, raw:  string?)
```

Append whitespace-delimited C option words from one source string.

Parameters

dst	Destination vector for parsed option words.
raw	Raw options string, or `null`.

8.53.1.2 bd_build_to_exe()

```
func bd_build_to_exe(opts:  CliOptions*, cfg:  LogConfig*, result:  AnalysisResult*, workspace:
CompilerWorkspace*, exe_path:  string, keep_c_path:  string?)  → int
```

Build an executable using one command-owned workspace.

The caller owns `result` and `workspace`, including the single cleanup epilogue. This helper only registers and uses its fixed scratch children.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.
result	Validated analysis result.
workspace	Command-owned workspace.
exe_path	Output executable path.
keep_c_path	Optional kept-C output path.

Returns

Process-style status code.

8.53.1.3 bd_cmd_build()

```
func bd_cmd_build(opts:  CliOptions*, cfg:  LogConfig*) → int
```

Execute Stage 2 `--build`.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.

Returns

Process-style status code.

8.53.1.4 bd_cmd_run()

```
func bd_cmd_run(opts:  CliOptions*, cfg:  LogConfig*) → int
```

Execute Stage 2 `--run`.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.

Returns

Process-style status code.

8.53.1.5 bd_collect_c_option_words()

```
func bd_collect_c_option_words(opts:  CliOptions*) → StringVector*
```

Resolve merged C option words from `L0_CFLAGS` and parsed CLI options.

Parameters

opts	Parsed CLI options.

Returns

Parsed option words in final order.

8.53.1.6 bd_compiler_flag_family()

```
func bd_compiler_flag_family(compiler:  string) → string
```

Return the compiler flag-family name used by Stage 1.

Parameters

compiler	Compiler command string.

Returns

Flag-family tag.

8.53.1.7 bd_default_exe_name()

```
func bd_default_exe_name() → string
```

Return the default build output name for the current host.

Returns

`a.exe` on Windows, otherwise `a.out`.

8.53.1.8 bd_emit_write_error()

```
func bd_emit_write_error(diags: DiagCollector*, path: string)
```

Emit one Stage 2-only write-file diagnostic.

Parameters

diags	Diagnostic collector.
path	Path that could not be written.

8.53.1.9 bd_exe_suffix()

```
func bd_exe_suffix() → string
```

Return the executable suffix for the current host.

Returns

`.exe` on Windows, otherwise `.out`.

8.53.1.10 bd_find_c_compiler()

```
func bd_find_c_compiler() → string?
```

Return the best available C compiler command or `null`.

Returns

Compiler command string, or `null`.

8.53.1.11 bd_find_last_sep()

```
func bd_find_last_sep(path: string) → int
```

Replace or add a `.c` suffix on a path string.

Parameters

path	Path to rewrite.

Returns

Path with `.c` suffix.

8.53.1.12 bd_format_word_list()

```
func bd_format_word_list(words:  StringVector*) → string
```

Format option words using the Stage 1 verbose display contract.

Parameters

words	Option vector to format.

Returns

Python-list-style rendering for verbose logs.

8.53.1.13 bd_get_optimize_flag()

```
func bd_get_optimize_flag(family:  string, extra_opts:  StringVector*) → string?
```

Return the default optimization flag for a compiler family.

Parameters

family	Compiler flag-family name.
extra_opts	Collected extra C compiler options.

Returns

Optimization flag, or `null`.

8.53.1.14 bd_has_debug_flag()

```
func bd_has_debug_flag(extra_opts:  StringVector*) → bool
```

Return whether the user supplied a debug flag.

Parameters

extra_opts	Collected extra C compiler options.

Returns

`true` when a debug flag is present.

8.53.1.15 bd_has_explicit_opt_flag()

```
func bd_has_explicit_opt_flag(extra_opts:  StringVector*) → bool
```

Return whether the user already supplied an optimization flag.

Parameters

extra_opts	Collected extra C compiler options.

Returns

`true` when an explicit optimization flag is present.

8.53.1.16 bd_is_windows()

```
func bd_is_windows() → bool
```

Return whether the current host should use Windows command behavior.

Returns

`true` when Windows shell rules should apply.

8.53.1.17 bd_join_display_words()

```
func bd_join_display_words(words:  StringVector*) → string
```

Join command words into the Stage 1 visible command-line form.

Parameters

words	Command words to join.

Returns

Space-delimited command line without shell quoting.

8.53.1.18 bd_join_shell_words()

```
func bd_join_shell_words(words:  StringVector*) → string
```

Join command words into one shell command line.

Parameters

words	Command words to quote and join.

Returns

One shell command line.

8.53.1.19 bd_log_effective_roots()

```
func bd_log_effective_roots(cfg: LogConfig*, sys_roots: StringVector*, project_roots: StringVector*)
```

Log the effective system and project roots for build/run commands.

Parameters

cfg	Logging configuration.
sys_roots	Effective system-root vector.
project_roots	Effective project-root vector.

8.53.1.20 bd_merge_c_option_words()

```
func bd_merge_c_option_words(env_opts: string?, cli_opts: string?) → StringVector*
```

Merge C option words from environment and CLI with CLI-last precedence.

Parameters

env_opts	Options from `L0_CFLAGS`.
cli_opts	Options from `--c-options`.

Returns

Parsed option words in final order.

8.53.1.21 bd_null_device()

```
func bd_null_device() → string
```

Return the shell null device for the current host shell.

Returns

`NUL` on Windows, otherwise `/dev/null`.

8.53.1.22 bd_prepare_build_result()

```
func bd_prepare_build_result(opts: CliOptions*, cfg: LogConfig*) → AnalysisResult*?
```

Analyze and validate one build/run entry before workspace creation.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.

Returns

Validated analysis result, or `null` on failure.

8.53.1.23 bd_prepare_input()

```
func bd_prepare_input(opts: CliOptions*, diags: DiagCollector*, cfg: LogConfig*) → BuildPreparedInput*?
```

Normalize CLI target input into an entry module and source search paths.

Parameters

opts	Parsed CLI options.
diags	Diagnostic collector for normalization failures.
cfg	Logging configuration.

Returns

Prepared input bundle, or `null` on failure.

8.53.1.24 bd_prepared_input_free()

```
func bd_prepared_input_free(self: BuildPreparedInput*)
```

Free a prepared input bundle.

Parameters

self	Prepared input to free.

8.53.1.25 bd_prepared_input_free_opt()

```
func bd_prepared_input_free_opt(self: BuildPreparedInput*?)
```

Free an optional prepared input bundle.

Parameters

self	Optional prepared input.

8.53.1.26 bd_push_c_compile_command_words()

```
func bd_push_c_compile_command_words(words: StringVector*, compiler: string, family: string,
extra_opts: StringVector*, optimize: string?, runtime_include: string?, c_path: string,
c_sources: StringVector*, exe_path: string, runtime_lib: string?)
```

Push a C compiler invocation into a command-word vector.

Parameters

words	Command words to extend.
compiler	C compiler executable.
family	Compiler flag-family name.
extra_opts	User C compiler option words.
optimize	Selected default optimization flag, or `null`.
runtime_include	Runtime header include path, or `null`.
c_path	Generated C input path.
c_sources	Additional C source input paths.
exe_path	Output executable path.
runtime_lib	Runtime library search path, or `null`.

8.53.1.27 bd_push_output_flags()

```
func bd_push_output_flags(words: StringVector*, family: string, exe_path: string)
```

Push compiler-family output-path flags into a command-word vector.

Parameters

words	Command words to extend.
family	Compiler flag-family name.
exe_path	Output executable path.

8.53.1.28 bd_push_standard_flags()

```
func bd_push_standard_flags(words: StringVector*, family: string)
```

Push compiler-family standard flags into a command-word vector.

Parameters

words	Command words to extend.
family	Compiler flag-family name.

8.53.1.29 bd_replay_compile_output()

```
func bd_replay_compile_output(cfg: LogConfig*, stdout_path: string, stderr_path: string,
```

```
failed:  bool)
```

Replay compiler output captured into temp files.

Parameters

cfg	Logging configuration.
stdout_path	Captured stdout path.
stderr_path	Captured stderr path.
failed	Whether compilation failed.

8.53.1.30 bd_runtime_include_dir()

```
func bd_runtime_include_dir(opts:  CliOptions*) → string?
```

Resolve the runtime include directory for build/run.

Parameters

opts	Parsed CLI options.

Returns

Runtime include directory, or `null`.

8.53.1.31 bd_runtime_lib_dir()

```
func bd_runtime_lib_dir(opts:  CliOptions*) → string?
```

Resolve the runtime library directory for build/run.

Parameters

opts	Parsed CLI options.

Returns

Runtime library directory, or `null`.

8.53.1.32 bd_shell_quote()

```
func bd_shell_quote(arg:  string) → string
```

Quote one shell token for the active host shell.

Parameters

arg	Raw token.

Returns

Safely quoted token.

8.53.1.33 bd_shell_quote_posix()

```
func bd_shell_quote_posix(arg:  string) → string
```

Quote one shell token for POSIX `system()` execution.

Parameters

arg	Raw token.

Returns

Safely single-quoted token.

8.53.1.34 bd_shell_quote_windows()

```
func bd_shell_quote_windows(arg:  string) → string
```

Quote one shell token for Windows `system()` execution.

Parameters

arg	Raw token.

Returns

Safely double-quoted token.

8.53.1.35 bd_validate_entry_main()

```
func bd_validate_entry_main(result:  AnalysisResult*, entry_module:  string, diags:  DiagCollector*)
→ bool
```

Validate the entry module `main` function for build/run.

Parameters

result	Analysis result to inspect.
entry_module	Entry module name.
diags	Diagnostic collector for failures and warnings.

Returns

`true` when the entry `main` is acceptable.

8.53.1.36 bd_validate_runtime_lib()

```
func bd_validate_runtime_lib(runtime_lib_path:  string, diags:  DiagCollector*) → bool
```

Validate that a runtime library directory exists.

Parameters

runtime_lib_path	Path to validate.
diags	Diagnostic collector for failures.

Returns

`true` on success.

8.53.1.37 bd_windows_exec_token()

```
func bd_windows_exec_token(arg:  string) → string
```

Normalize one Windows executable token for `cmd.exe`.

Relative `./...` command paths are valid under POSIX shells but `cmd.exe` interprets the leading `./` as a command named `..` Rewriting the slash separators to `.\...` keeps caller-selected relative executables runnable.

Parameters

arg	Raw executable token.

Returns

Command token with cmd.exe-safe relative separators.

8.53.1.38 bd_windows_token_is_plain()

```
func bd_windows_token_is_plain(arg:  string) → bool
```

Return whether one Windows shell token can remain unquoted.

Parameters

arg	Raw token.

Returns

`true` when cmd.exe-safe token text needs no surrounding quotes.

8.53.1.39 bd_with_c_suffix()

```
func bd_with_c_suffix(path:  string) → string
```

Replace or add a `.c` suffix on a path string.

Parameters

path	Path to rewrite.

Returns

Path with `.c` suffix.

8.54 compiler/stage2_l0/src/build_info.l0 File Reference

Module: `build_info`

```
import std.text;
```

Functions

- func build_info_has_embedded_version() → bool

  Report whether this compiler binary has embedded build provenance.
- func build_info_build_id() → string

  Return the recorded build identifier.
- func build_info_build_time() → string

  Return the recorded UTC build timestamp rendered for `--version`.
- func build_info_commit() → string

  Return the formatted commit text for `--version`.
- func build_info_host() → string

  Return the recorded host platform string.
- func build_info_compiler() → string

  Return the recorded compiler banner for `--version`.
- func build_info_release_version() → string

  Return the recorded release version string.
- func build_info_identity_with_version(identity: string) → string

  Return the compiler identity with version suffix when provenance is embedded.
- func build_info_version_text(identity: string) → string?

  Build the multiline Stage 2 `--version` report when provenance is embedded.

8.54.1 Function Documentation

8.54.1.1 build_info_build_id()

```
func build_info_build_id() → string
```

Return the recorded build identifier.

Returns

Build identifier, or `"unknown"` for fallback builds.

8.54.1.2 build_info_build_time()

```
func build_info_build_time() → string
```

Return the recorded UTC build timestamp rendered for `--version`.

Returns

Build time text, or `"unknown"` for fallback builds.

8.54.1.3 build_info_commit()

```
func build_info_commit() → string
```

Return the formatted commit text for `--version`.

Returns

Full git commit hash, optionally suffixed with `+dirty`, or `"unknown"` for fallback builds.

8.54.1.4 build_info_compiler()

```
func build_info_compiler() → string
```

Return the recorded compiler banner for `--version`.

Returns

First line of `<compiler> --version`, or `"unknown"` for fallback builds.

8.54.1.5 build_info_has_embedded_version()

```
func build_info_has_embedded_version() → bool
```

Report whether this compiler binary has embedded build provenance.

Source-tree and raw self-hosted builds stay on the static fallback path.

Returns

`true` when an artifact-producing flow embedded provenance.

8.54.1.6 build_info_host()

```
func build_info_host() → string
```

Return the recorded host platform string.

Returns

Host platform string, or `"unknown"` for fallback builds.

8.54.1.7 build_info_identity_with_version()

```
func build_info_identity_with_version(identity:  string) → string
```

Return the compiler identity with version suffix when provenance is embedded.

Parameters

identity	Stable Stage 2 compiler identity line.

Returns

Identity with version (e.g., `"Dea language / L0 compiler 1.2.3"`) when available, or plain identity for fallback builds.

8.54.1.8 build_info_release_version()

```
func build_info_release_version() → string
```

Return the recorded release version string.

Returns

Release version such as `"1.2.3"` or `"dev-abc1234"`, or `"unknown"` for fallback builds.

8.54.1.9 build_info_version_text()

```
func build_info_version_text(identity:  string) → string?
```

Build the multiline Stage 2 `--version` report when provenance is embedded.

Parameters

identity	Stable Stage 2 compiler identity line.

Returns

Multiline report text, or `null` when this binary uses fallback metadata.

8.55 compiler/stage2_l0/src/c_emitter.l0 File Reference

Module: `c_emitter`

```
import std.assert;
import std.hashset;
import std.hashmap;
import std.string;
import std.text;
import std.vector;
import ast;
import codegen_options;
import driver;
import string_escape;
import util.strings;
import sem_context;
import signatures;
import types;
```

Classes

- struct CCodeBuilder

  Represents a line-oriented C source builder with indentation tracking.
- struct CEmitter

  Represents the Stage 2 C emitter state.

Functions

- func cem_ice(message: string)

  Abort on an internal code-emission error.
- func cem_ice_string(message: string) → string

  Abort on an internal code-emission error from a string-returning helper.
- func cem_upper_ascii(text: string) → string

  Build an all-uppercase ASCII variant of an identifier-like string.
- func cem_opt_wrapper_map_free(items: StringPtrMap∗)

  Free every collected optional-wrapper inner type.
- func ccb_create() → CCodeBuilder∗

  Create a new code builder.
- func ccb_free(self: CCodeBuilder∗)

  Free a code builder.

- func ccb_indent(self: CCodeBuilder*)

  Increase indentation level by one.
- func ccb_dedent(self: CCodeBuilder*)

  Decrease indentation level by one.
- func ccb_prefix(self: CCodeBuilder*) → string

  Build the current indentation prefix.
- func ccb_emit(self: CCodeBuilder*, line: string)

  Emit one line with indentation.
- func ccb_emit_raw(self: CCodeBuilder*, line: string)

  Emit one line without indentation.
- func ccb_to_string(self: CCodeBuilder*) → string

  Convert the accumulated lines to one source string with a trailing newline.
- func cem_create() → CEmitter*

  Create a Stage 2 C emitter.
- func cem_reset_optional_wrappers(self: CEmitter*)

  Clear every collected optional-wrapper entry.
- func cem_free(self: CEmitter*)

  Free a Stage 2 C emitter.
- func cem_set_analysis(self: CEmitter*, analysis: AnalysisResult*)

  Attach analysis data to an emitter.
- func cem_analysis(self: CEmitter*) → AnalysisResult*

  Return the borrowed analysis result.
- func cem_get_output(self: CEmitter*) → string

  Return the generated C source text.
- func cem_is_c_keyword(name: string) → bool

  Return whether a name conflicts with a C keyword or restricted identifier.
- func cem_mangle_module(module_name: string) → string

  Replace dots with underscores for C symbol names.
- func cem_mangle_struct_name(module_name: string, struct_name: string) → string

  Mangle a struct name.
- func cem_mangle_enum_name(module_name: string, enum_name: string) → string

  Mangle an enum name.
- func cem_mangle_function_name(module_name: string, func_name: string) → string

  Mangle a function name.
- func cem_mangle_let_name(module_name: string, let_name: string) → string

  Mangle a top-level let name.
- func cem_mangle_identifier(name: string) → string

  Mangle a local identifier when needed.
- func cem_fresh_tmp(self: CEmitter*, kind: string) → string

  Produce a fresh temporary identifier.
- func cem_emit_sizeof_type(typ: Type*) → string

  Emit C code for `sizeof(<type>)`.
- func cem_emit_ord(c_enum_expr: string) → string

  Emit C code for `ord(enum_expr)`.
- func cem_emit_pointer_type(base_type: Type*) → string

  Emit a pointer type from a base type.
- func cem_emit_enum_tag(enum_type: Type*, variant_name: string) → string

  Emit the C tag constant for an enum variant.
- func cem_is_niche_nullable(t: Type*) → bool

  Return whether a nullable type uses pointer-shaped niche encoding.
- func cem_opt_key_for_type(t: Type*) → string

Produce the stable wrapper-key suffix for an optional inner type.

- func cem_opt_wrapper_name_for_inner(inner: Type∗) → string

  Return the wrapper typedef name used for a non-pointer nullable type.
- func cem_emit_type(typ: Type∗) → string

  Emit an L0 type as a C type spelling.
- func cem_emit_none_value_for_nullable(t: Type∗) → string

  Emit a `none` value for a non-pointer nullable type.
- func cem_emit_some_value_for_nullable(t: Type∗, c_inner_expr: string) → string

  Emit a wrapped `some` value for a non-pointer nullable type.
- func cem_emit_null_literal(expected_type: Type∗, for_initializer: bool) → string

  Emit a `null` literal for the expected type.
- func cem_emit_widen_int(c_expr: string, src_type: Type∗, dst_type: Type∗) → string

  Emit a widening cast for an implicit integer conversion.
- func cem_emit_section_comment(self: CEmitter∗, text: string)

  Emit a section comment.
- func cem_emit_module_comment(self: CEmitter∗, module_name: string)

  Emit a module comment.
- func cem_emit_module_separator(self: CEmitter∗, module_name: string)

  Emit a decorative module separator.
- func cem_emit_unreachable_comment(self: CEmitter∗)

  Emit an unreachable-code comment.
- func cem_emit_unreachable_marker(self: CEmitter∗, reason: string)

  Emit an unreachable runtime marker.
- func cem_emit_header(self: CEmitter∗, opts: CodegenOptions∗)

  Emit the fixed header boilerplate for one C translation unit.
- func cem_module_filename(self: CEmitter∗, module_name: string) → string

  Return the source filename for a loaded module.
- func cem_emit_line_directive(self: CEmitter∗, line: int, current_module: string, opts: CodegenOptions∗)

  Emit a `#line` directive when enabled.
- func cem_restore_active_line_directive(self: CEmitter∗)

  Re-emit the current source line directive after generated helper lines.
- func cem_emit_forward_decls(self: CEmitter∗)

  Emit forward declarations for all structs and enums in compilation order.
- func cem_find_enum_variant_info(enum_info: EnumInfo∗, variant_name: string) → EnumVariantInfo∗?

  Look up enum payload metadata by variant name.
- func cem_emit_struct(self: CEmitter∗, module_name: string, decl: StructDecl∗, struct_info: StructInfo∗)

  Emit a complete C struct definition.
- func cem_emit_enum(self: CEmitter∗, module_name: string, decl: EnumDecl∗, enum_info: EnumInfo∗)

  Emit an enum as a C tagged-union struct.
- func cem_emit_let_declaration(self: CEmitter∗, module_name: string, decl: LetDecl∗, let_type: Type∗, c_init: string)

  Emit one top-level `let` declaration as a static C variable.
- func cem_emit_top_level_let_cleanup(self: CEmitter∗)

  Emit final cleanup for ARC-managed top-level lets before process exit.
- func cem_function_params(decl: FuncDecl∗, func_type: Type∗) → string

  Build a function parameter list.
- func cem_emit_function_declaration(self: CEmitter∗, module_name: string, decl: FuncDecl∗, func_type: Type∗)

  Emit a function declaration.
- func cem_emit_function_definition_header(self: CEmitter∗, module_name: string, decl: FuncDecl∗, func_type: Type∗)

Emit a function definition header.

- func cem_emit_function_definition_footer(self: CEmitter∗)

  Emit a function definition footer.
- func cem_emit_main_wrapper(self: CEmitter∗, entry_module: string, func_type: Type∗)

  Emit the C `main()` wrapper for the entry module.
- func cem_note_opt_wrapper(self: CEmitter∗, inner: Type∗)

  Record one optional-wrapper inner type if it was not already collected.
- func cem_collect_opt_wrappers_from_type(self: CEmitter∗, t: Type∗)

  Recursively collect optional-wrapper types reachable from one type.
- func cem_is_early_inner(inner: Type∗) → bool

  Return whether a wrapper inner type can be emitted before user type definitions.
- func cem_emit_optional_wrapper(self: CEmitter∗, name: string, inner: Type∗)

  Emit one collected optional-wrapper typedef.
- func cem_emit_optional_wrapper_for_defined_type(self: CEmitter∗, inner: Type∗)

  Emit the collected optional wrapper for a just-defined user type.
- func cem_prepare_optional_wrappers(self: CEmitter∗)

  Scan analysis metadata and collect every needed optional wrapper.
- func cem_emit_optional_wrappers(self: CEmitter∗, early: bool)

  Emit collected optional-wrapper typedefs for one phase.
- func cem_emit_int_literal(value: int) → string

  Emit an integer literal.
- func cem_emit_bool_literal(value: bool) → string

  Emit a boolean literal.
- func cem_emit_const_bool_literal(value: bool) → string

  Emit a boolean literal for static initialization.
- func cem_encode_c_byte_literal(value: byte) → string

  Encode one byte for a C character literal body.
- func cem_emit_byte_literal(value: byte) → string

  Emit a byte literal as a casted C character constant.
- func cem_emit_string_literal(value: string) → string

  Emit a decoded string literal through `L0_STRING_CONST`.
- func cem_emit_const_string_literal(value: string) → string

  Emit a decoded string literal for static initialization.
- func cem_emit_var_ref(c_name: string) → string

  Emit a variable reference expression.
- func cem_emit_unary_op(op: string, c_operand: string) → string

  Emit a unary operation expression.
- func cem_emit_negated_condition(c_cond: string) → string

  Emit a negated condition expression for control-flow lowering.
- func cem_emit_binary_op(op: string, c_left: string, c_right: string) → string

  Emit a binary operation expression.
- func cem_emit_condition_binary_op(op: string, c_left: string, c_right: string) → string

  Emit a top-level binary condition for direct statement headers.
- func cem_emit_checked_int_div(c_left: string, c_right: string) → string

  Emit a checked integer division call.
- func cem_emit_checked_int_mod(c_left: string, c_right: string) → string

  Emit a checked integer modulo call.
- func cem_emit_checked_int_mul(c_left: string, c_right: string) → string

  Emit a checked integer multiplication call.
- func cem_emit_checked_int_add(c_left: string, c_right: string) → string

  Emit a checked integer addition call.

- func cem_emit_checked_int_sub(c_left: string, c_right: string) → string

  Emit a checked integer subtraction call.
- func cem_emit_function_call(c_func_name: string, c_args: string) → string

  Emit a function-call expression.
- func cem_emit_field_access(c_obj: string, field_name: string, is_pointer: bool) → string

  Emit a field-access expression.
- func cem_emit_paren_expr(c_inner: string) → string

  Emit a parenthesized expression.
- func cem_emit_cast(c_type: string, c_inner: string) → string

  Emit a C cast expression.
- func cem_emit_checked_ptr_access(self: CEmitter∗, c_ptr_expr: string, c_ptr_type: string, c_required_size: string, c_required_align: string, access_mode: string) → string

  Emit a runtime-checked pointer access expression.
- func cem_emit_checked_ptr_index_access(self: CEmitter∗, c_base_expr: string, c_index_expr: string, c_ptr_type: string, c_element_size: string, c_required_align: string, access_mode: string) → string

  Emit a runtime-checked indexed pointer access expression.
- func cem_emit_checked_ptr_access_for_base(self: CEmitter∗, c_ptr_expr: string, c_base_type: string, access_mode: string) → string

  Emit a checked pointer to one complete object of the given C base type.
- func cem_emit_drop_begin_expr(c_ptr_expr: string, c_ptr_type: string, c_required_size: string, c_required_align: string) → string

  Emit the expression that validates and begins a generated drop.
- func cem_emit_drop_finish_call(self: CEmitter∗, c_ptr_expr: string)

  Emit the runtime call that completes a generated drop.
- func cem_emit_checked_narrow_cast(c_dst_type: string, c_inner: string) → string

  Emit a checked narrowing cast call.
- func cem_emit_unwrap_ptr(c_dst_type: string, c_inner: string, type_str: string) → string

  Emit an unwrap call for pointer-shaped nullable values.
- func cem_emit_unwrap_opt(c_src_type: string, c_inner: string, type_str: string) → string

  Emit an unwrap call for wrapper-shaped nullable values.
- func cem_emit_null_check_eq(c_expr: string) → string

  Emit a null-equality check for wrapper-shaped nullable values.
- func cem_emit_null_check_ne(c_expr: string) → string

  Emit a null-inequality check for wrapper-shaped nullable values.
- func cem_emit_pointer_null_check(c_expr: string, op: string) → string

  Emit a pointer null comparison.
- func cem_emit_condition_pointer_null_check(c_expr: string, op: string) → string

  Emit a top-level pointer null comparison for direct statement headers.
- func cem_emit_optional_has_value(c_expr: string) → string

  Emit access to an optional wrapper's `has_value` flag.
- func cem_emit_optional_value(c_expr: string) → string

  Emit access to an optional wrapper's payload value.
- func cem_emit_enum_tag_access(c_expr: string) → string

  Emit access to an enum tag.
- func cem_emit_enum_payload_field_access(c_expr: string, variant: string, field: string) → string

  Emit access to one enum payload field.
- func cem_emit_string_equals_call(lhs: string, rhs: string) → string

  Emit the runtime string-equality helper call.
- func cem_emit_string_concat_call(lhs: string, rhs: string) → string

  Emit the runtime string-concatenation helper call.
- func cem_emit_string_compare_call(op: string, lhs: string, rhs: string) → string

Emit the runtime string-compare helper call plus a relational check.

- func cem_emit_condition_string_compare_call(op: string, lhs: string, rhs: string) → string

  Emit a top-level string relational check for direct statement headers.
- func cem_emit_discard_expr(c_expr: string) → string

  Emit a statement-context discard wrapper for an expression.
- func cem_emit_deref_lvalue(ptr_expr: string) → string

  Emit a dereference lvalue.
- func cem_emit_field_lvalue(obj: string, field: string, is_pointer: bool) → string

  Emit a field lvalue.
- func cem_emit_index_lvalue(base: string, index: string) → string

  Emit an index lvalue.
- func cem_emit_struct_constructor(c_struct_name: string, field_inits: StringVector∗) → string

  Emit a struct constructor literal.
- func cem_emit_struct_static_initializer(field_inits: StringVector∗) → string

  Emit a brace-only struct initializer for static storage duration.
- func cem_emit_struct_constructor_for_type(struct_type: Type∗, field_inits: StringVector∗) → string

  Emit a struct constructor for a resolved struct type.
- func cem_emit_struct_static_initializer_for_type(struct_type: Type∗, field_inits: StringVector∗) → string

  Emit a brace-only struct initializer for a resolved struct type.
- func cem_emit_variant_constructor(c_enum_name: string, variant_name: string, tag_value: string, payload_inits: StringVector∗) → string

  Emit an enum-variant constructor literal.
- func cem_emit_variant_static_initializer(variant_name: string, tag_value: string, payload_inits: StringVector∗) → string

  Emit a brace-only enum-variant initializer for static storage duration.
- func cem_emit_variant_constructor_for_type(enum_type: Type∗, variant_name: string, payload_inits: StringVector∗) → string

  Emit an enum-variant constructor for a resolved enum type.
- func cem_emit_variant_static_initializer_for_type(enum_type: Type∗, variant_name: string, payload_inits: StringVector∗) → string

  Emit a brace-only enum-variant initializer for a resolved enum type.
- func cem_emit_expr_stmt(self: CEmitter∗, c_expr: string)

  Emit an expression statement.
- func cem_emit_return_stmt(self: CEmitter∗, c_value: string)

  Emit a return statement.
- func cem_emit_return_void_stmt(self: CEmitter∗)

  Emit a `return;` statement.
- func cem_emit_label(self: CEmitter∗, label: string)

  Emit a C label.
- func cem_emit_goto(self: CEmitter∗, label: string)

  Emit a `goto` statement.
- func cem_emit_block_start(self: CEmitter∗)

  Emit a block start.
- func cem_emit_block_end(self: CEmitter∗)

  Emit a block end.
- func cem_emit_while_header(self: CEmitter∗, c_cond: string)

  Emit a `while` header.
- func cem_emit_if_header(self: CEmitter∗, c_cond: string)

  Emit an `if` header.
- func cem_emit_else(self: CEmitter∗)

  Emit an `else` keyword.

- func cem_emit_for_loop_start(self: CEmitter*)

  Emit the Stage 1-shaped for-loop wrapper start.
- func cem_emit_for_loop_end(self: CEmitter*)

  Emit the Stage 1-shaped for-loop wrapper end.
- func cem_emit_let_decl(self: CEmitter*, c_type: string, c_var_name: string, c_init: string)

  Emit a local variable declaration.
- func cem_emit_assignment(self: CEmitter*, c_target: string, c_value: string)

  Emit an assignment statement.
- func cem_emit_temp_decl(self: CEmitter*, c_type: string, c_temp_name: string, c_value: string)

  Emit a temporary declaration.
- func cem_emit_comment(self: CEmitter*, comment: string)

  Emit a block comment.
- func cem_emit_exit_switch(self: CEmitter*)

  Emit a switch-exit `break;`.
- func cem_emit_pointer_assignment(self: CEmitter*, c_ptr_name: string, c_value: string)

  Emit an assignment through a pointer.
- func cem_emit_checked_pointer_assignment(self: CEmitter*, c_ptr_name: string, c_base_type: string, c_value: string)

  Emit an assignment through a runtime-checked object pointer.
- func cem_emit_string_retain(self: CEmitter*, c_expr: string)

  Emit an ARC string retain call.
- func cem_emit_string_release(self: CEmitter*, c_expr: string)

  Emit an ARC string release call.
- func cem_emit_match_scrutinee_decl(self: CEmitter*, c_type: string, c_expr: string)

  Emit the temporary scrutinee declaration used by `match`/`case`.
- func cem_emit_switch_start(self: CEmitter*, c_expr: string)

  Emit a `switch` header.
- func cem_emit_match_switch_start(self: CEmitter*, scrutinee_name: string)

  Emit the switch used for enum `match`.
- func cem_emit_switch_end(self: CEmitter*)

  Emit the closing brace of a `switch`.
- func cem_emit_case_label(self: CEmitter*, c_value: string)

  Emit a `case` label.
- func cem_emit_default_label(self: CEmitter*)

  Emit a `default:` label.
- func cem_emit_null_assignment(self: CEmitter*, c_var: string)

  Emit a NULL assignment.
- func cem_emit_alloc_obj(self: CEmitter*, c_ptr_type: string, c_base_type: string, c_temp_name: string)

  Emit one heap-allocation temporary.
- func cem_emit_struct_init(self: CEmitter*, c_temp_name: string, c_base_type: string, c_init_str: string)

  Emit one pointer-target aggregate initialization.
- func cem_emit_struct_init_from_fields(self: CEmitter*, c_temp_name: string, base_type: Type*, field_inits: StringVector*)

  Emit one pointer-target struct initialization from field pairs.
- func cem_emit_enum_variant_init(self: CEmitter*, c_temp_name: string, enum_type: Type*, variant_name: string, payload_inits: StringVector*)

  Emit one pointer-target enum-variant initialization.
- func cem_emit_zero_init(self: CEmitter*, c_temp_name: string, c_base_type: string)

  Emit zero-initialization through a pointer.
- func cem_emit_try_check_niche(self: CEmitter*, c_tmp: string, ret_none: string)

  Emit the null-check used by `?` on niche optionals.

- func cem_emit_try_check_value(self: CEmitter*, c_tmp: string, ret_none: string)

  Emit the `has_value` check used by `?` on value optionals.
- func cem_emit_try_extract_value(c_tmp: string) → string

  Emit the value extraction used by `?` on value optionals.
- func cem_emit_pattern_binding_init(scrutinee: string, variant: string, field: string) → string

  Emit one match-arm payload binding initializer.
- func cem_find_enum_decl(self: CEmitter*, module_name: string, enum_name: string) → EnumDecl*?

  Look up a loaded enum declaration by module and name.
- func cem_enum_has_arc_data(self: CEmitter*, enum_info: EnumInfo*, decl: EnumDecl*) → bool

  Report whether any variant field in an enum carries ARC-managed data.
- func cem_emit_value_cleanup(self: CEmitter*, c_expr: string, ty: Type*)

  Emit cleanup for a by-value expression according to its type.
- func cem_emit_struct_cleanup(self: CEmitter*, c_ptr_expr: string, struct_type: Type*)

  Emit cleanup for a pointer-to-struct value.
- func cem_emit_enum_cleanup(self: CEmitter*, c_ptr_expr: string, enum_type: Type*)

  Emit cleanup for a pointer-to-enum value.

8.55.1 Function Documentation

8.55.1.1 ccb_create()

```
func ccb_create() → CCodeBuilder*
```

Create a new code builder.

Returns

New builder with four-space indentation.

8.55.1.2 ccb_dedent()

```
func ccb_dedent(self:  CCodeBuilder*)
```

Decrease indentation level by one.

Parameters

self	Builder to update.

8.55.1.3 ccb_emit()

```
func ccb_emit(self:  CCodeBuilder*, line:  string)
```

Emit one line with indentation.

Parameters

self	Builder to update.
line	Line text, or " " for a blank line.

8.55.1.4 ccb_emit_raw()

```
func ccb_emit_raw(self:  CCodeBuilder*, line:  string)
```

Emit one line without indentation.

Parameters

self	Builder to update.
line	Raw line text.

8.55.1.5 ccb_free()

```
func ccb_free(self:  CCodeBuilder*)
```

Free a code builder.

Parameters

self	Builder to free.

8.55.1.6 ccb_indent()

```
func ccb_indent(self:  CCodeBuilder*)
```

Increase indentation level by one.

Parameters

self	Builder to update.

8.55.1.7 ccb_prefix()

```
func ccb_prefix(self:  CCodeBuilder*) → string
```

Build the current indentation prefix.

Parameters

self	Builder to inspect.

Returns

Indentation string for the current level.

8.55.1.8 ccb_to_string()

```
func ccb_to_string(self:  CCodeBuilder*) → string
```

Convert the accumulated lines to one source string with a trailing newline.

Parameters

self	Builder to inspect.

Returns

Combined source text.

8.55.1.9 cem_analysis()

```
func cem_analysis(self:  CEmitter*) → AnalysisResult*
```

Return the borrowed analysis result.

Parameters

self	Emitter to inspect.

Returns

Borrowed analysis result.

8.55.1.10 cem_collect_opt_wrappers_from_type()

```
func cem_collect_opt_wrappers_from_type(self:  CEmitter*, t:  Type*)
```

Recursively collect optional-wrapper types reachable from one type.

Parameters

self	Emitter to update.
t	Type to scan.

8.55.1.11 cem_create()

```
func cem_create() → CEmitter*
```

Create a Stage 2 C emitter.

Returns

New emitter.

8.55.1.12 cem_emit_alloc_obj()

```
func cem_emit_alloc_obj(self: CEmitter*, c_ptr_type: string, c_base_type: string, c_temp_name:
string)
```

Emit one heap-allocation temporary.

Parameters

self	Emitter to update.
c_ptr_type	Lowered pointer type.
c_base_type	Lowered base type.
c_temp_name	Temporary variable name.

8.55.1.13 cem_emit_assignment()

```
func cem_emit_assignment(self: CEmitter*, c_target: string, c_value: string)
```

Emit an assignment statement.

Parameters

self	Emitter to update.
c_target	Lowered lvalue.
c_value	Lowered rvalue.

8.55.1.14 cem_emit_binary_op()

```
func cem_emit_binary_op(op: string, c_left: string, c_right: string) → string
```

Emit a binary operation expression.

Parameters

op	Binary operator token.
c_left	Lowered left operand.
c_right	Lowered right operand.

Returns

Parenthesized C binary expression.

8.55.1.15 cem_emit_block_end()

```
func cem_emit_block_end(self:  CEmitter*)
```

Emit a block end.

Parameters

self	Emitter to update.

8.55.1.16 cem_emit_block_start()

```
func cem_emit_block_start(self:  CEmitter*)
```

Emit a block start.

Parameters

self	Emitter to update.

8.55.1.17 cem_emit_bool_literal()

```
func cem_emit_bool_literal(value:  bool) → string
```

Emit a boolean literal.

Parameters

value	Boolean value.

Returns

C spelling for the literal.

8.55.1.18 cem_emit_byte_literal()

```
func cem_emit_byte_literal(value:  byte) → string
```

Emit a byte literal as a casted C character constant.

Parameters

value	Byte value.

Returns

C spelling for the literal.

8.55.1.19 cem_emit_case_label()

```
func cem_emit_case_label(self:  CEmitter*, c_value:  string)
```

Emit a `case` label.

Parameters

self	Emitter to update.
c_value	Lowered tag or literal.

8.55.1.20 cem_emit_cast()

```
func cem_emit_cast(c_type:  string, c_inner:  string) → string
```

Emit a C cast expression.

Parameters

c_type	Destination C type.
c_inner	Inner expression.

Returns

Cast expression.

8.55.1.21 cem_emit_checked_int_add()

```
func cem_emit_checked_int_add(c_left:  string, c_right:  string) → string
```

Emit a checked integer addition call.

Parameters

c_left	Lowered left operand.
c_right	Lowered right operand.

Returns

Runtime-call expression.

8.55.1.22 cem_emit_checked_int_div()

```
func cem_emit_checked_int_div(c_left:  string, c_right:  string) → string
```

Emit a checked integer division call.

Parameters

c_left	Lowered left operand.
c_right	Lowered right operand.

Returns

Runtime-call expression.

8.55.1.23 cem_emit_checked_int_mod()

```
func cem_emit_checked_int_mod(c_left:  string, c_right:  string) → string
```

Emit a checked integer modulo call.

Parameters

c_left	Lowered left operand.
c_right	Lowered right operand.

Returns

Runtime-call expression.

8.55.1.24 cem_emit_checked_int_mul()

```
func cem_emit_checked_int_mul(c_left:  string, c_right:  string) → string
```

Emit a checked integer multiplication call.

Parameters

c_left	Lowered left operand.
c_right	Lowered right operand.

Returns

Runtime-call expression.

8.55.1.25 cem_emit_checked_int_sub()

```
func cem_emit_checked_int_sub(c_left:  string, c_right:  string) → string
```

Emit a checked integer subtraction call.

Parameters

c_left	Lowered left operand.
c_right	Lowered right operand.

Returns

Runtime-call expression.

8.55.1.26 cem_emit_checked_narrow_cast()

```
func cem_emit_checked_narrow_cast(c_dst_type:  string, c_inner:  string) → string
```

Emit a checked narrowing cast call.

Parameters

c_dst_type	Destination C type name.
c_inner	Inner expression.

Returns

Runtime-call expression.

8.55.1.27 cem_emit_checked_pointer_assignment()

```
func cem_emit_checked_pointer_assignment(self:  CEmitter*, c_ptr_name:  string, c_base_type:
string, c_value:  string)
```

Emit an assignment through a runtime-checked object pointer.

Parameters

self	Emitter to update.
c_ptr_name	C pointer expression.
c_base_type	Lowered pointee type.
c_value	Lowered value expression.

8.55.1.28 cem_emit_checked_ptr_access()

```
func cem_emit_checked_ptr_access(self: CEmitter*, c_ptr_expr: string, c_ptr_type: string,
c_required_size: string, c_required_align: string, access_mode: string) → string
```

Emit a runtime-checked pointer access expression.

Declares one static per-call-site cache slot so the runtime validates repeated accesses from the same site with a single range check.

Parameters

self	Emitter to update.
c_ptr_expr	Pointer expression.
c_ptr_type	Lowered pointer type.
c_required_size	Required byte extent expression.
c_required_align	Required alignment expression.
access_mode	Runtime access mode expression.

Returns

Checked pointer expression.

8.55.1.29 cem_emit_checked_ptr_access_for_base()

```
func cem_emit_checked_ptr_access_for_base(self: CEmitter*, c_ptr_expr: string, c_base_type:
string, access_mode: string) → string
```

Emit a checked pointer to one complete object of the given C base type.

Parameters

self	Emitter to update.
c_ptr_expr	Pointer expression.
c_base_type	Lowered pointee type.
access_mode	Runtime access mode expression.

Returns

Checked pointer expression.

8.55.1.30 cem_emit_checked_ptr_index_access()

```
func cem_emit_checked_ptr_index_access(self: CEmitter*, c_base_expr: string, c_index_expr:
string, c_ptr_type: string, c_element_size: string, c_required_align: string, access_mode:
string) → string
```

Emit a runtime-checked indexed pointer access expression.

Parameters

self	Emitter to update.
c_base_expr	Base pointer expression.
c_index_expr	Index expression.
c_ptr_type	Lowered pointer type.
c_element_size	Element byte extent expression.
c_required_align	Required alignment expression.
access_mode	Runtime access mode expression.

Returns

Checked pointer expression.

8.55.1.31 cem_emit_comment()

```
func cem_emit_comment(self:  CEmitter*, comment:  string)
```

Emit a block comment.

Parameters

self	Emitter to update.
comment	Comment text.

8.55.1.32 cem_emit_condition_binary_op()

```
func cem_emit_condition_binary_op(op:  string, c_left:  string, c_right:  string) → string
```

Emit a top-level binary condition for direct statement headers.

Parameters

op	Binary operator token.
c_left	Lowered left operand.
c_right	Lowered right operand.

Returns

Non-parenthesized C binary expression.

8.55.1.33 cem_emit_condition_pointer_null_check()

```
func cem_emit_condition_pointer_null_check(c_expr:  string, op:  string) → string
```

Emit a top-level pointer null comparison for direct statement headers.

Parameters

c_expr	Pointer expression.
op	Comparison operator.

Returns

Non-parenthesized comparison expression.

8.55.1.34 cem_emit_condition_string_compare_call()

```
func cem_emit_condition_string_compare_call(op:  string, lhs:  string, rhs:  string) → string
```

Emit a top-level string relational check for direct statement headers.

Parameters

op	Relational operator token.
lhs	Left operand.
rhs	Right operand.

Returns

Non-parenthesized comparison expression.

8.55.1.35 cem_emit_const_bool_literal()

```
func cem_emit_const_bool_literal(value:  bool) → string
```

Emit a boolean literal for static initialization.

Parameters

value	Boolean value.

Returns

C spelling for the static constant.

8.55.1.36 cem_emit_const_string_literal()

```
func cem_emit_const_string_literal(value:  string) → string
```

Emit a decoded string literal for static initialization.

Parameters

value	Decoded string bytes.

Returns

C spelling for the static constant.

8.55.1.37 cem_emit_default_label()

```
func cem_emit_default_label(self:  CEmitter*)
```

Emit a `default:` label.

Parameters

self	Emitter to update.

8.55.1.38 cem_emit_deref_lvalue()

```
func cem_emit_deref_lvalue(ptr_expr:  string) → string
```

Emit a dereference lvalue.

Parameters

ptr_expr	Pointer expression.

Returns

Dereference lvalue text.

8.55.1.39 cem_emit_discard_expr()

```
func cem_emit_discard_expr(c_expr:  string) → string
```

Emit a statement-context discard wrapper for an expression.

Parameters

c_expr	Expression to discard.

Returns

Wrapped expression.

8.55.1.40 cem_emit_drop_begin_expr()

```
func cem_emit_drop_begin_expr(c_ptr_expr: string, c_ptr_type: string, c_required_size: string,
c_required_align: string) → string
```

Emit the expression that validates and begins a generated drop.

Parameters

c_ptr_expr	Pointer expression.
c_ptr_type	Lowered pointer type.
c_required_size	Required pointee byte extent.
c_required_align	Required pointee alignment.

Returns

Checked pointer expression.

8.55.1.41 cem_emit_drop_finish_call()

```
func cem_emit_drop_finish_call(self: CEmitter*, c_ptr_expr: string)
```

Emit the runtime call that completes a generated drop.

Parameters

self	Emitter to update.
c_ptr_expr	Checked drop pointer expression.

8.55.1.42 cem_emit_else()

```
func cem_emit_else(self: CEmitter*)
```

Emit an `else` keyword.

Parameters

self	Emitter to update.

8.55.1.43 cem_emit_enum()

```
func cem_emit_enum(self: CEmitter*, module_name: string, decl: EnumDecl*, enum_info: EnumInfo*)
```

Emit an enum as a C tagged-union struct.

Parameters

self	Emitter to update.
module_name	Name of the defining module.
decl	Enum declaration node.
enum_info	Resolved enum metadata.

8.55.1.44 cem_emit_enum_cleanup()

```
func cem_emit_enum_cleanup(self:  CEmitter*, c_ptr_expr:  string, enum_type:  Type*)
```

Emit cleanup for a pointer-to-enum value.

Parameters

self	Emitter to update.
c_ptr_expr	Lowered pointer expression.
enum_type	Enum type to clean.

8.55.1.45 cem_emit_enum_payload_field_access()

```
func cem_emit_enum_payload_field_access(c_expr:  string, variant:  string, field:  string) →
string
```

Emit access to one enum payload field.

Parameters

c_expr	Enum expression.
variant	Active variant name.
field	Field name.

Returns

Member-access expression.

8.55.1.46 cem_emit_enum_tag()

```
func cem_emit_enum_tag(enum_type:  Type*, variant_name:  string) → string
```

Emit the C tag constant for an enum variant.

Parameters

enum_type	Enum type to inspect.
variant_name	Variant name.

Returns

Mangled enum-tag identifier.

8.55.1.47 cem_emit_enum_tag_access()

```
func cem_emit_enum_tag_access(c_expr:  string) → string
```

Emit access to an enum tag.

Parameters

c_expr	Enum expression.

Returns

Member-access expression.

8.55.1.48 cem_emit_enum_variant_init()

```
func cem_emit_enum_variant_init(self:  CEmitter*, c_temp_name:  string, enum_type:  Type*,
variant_name:  string, payload_inits:  StringVector*)
```

Emit one pointer-target enum-variant initialization.

Parameters

self	Emitter to update.
c_temp_name	Pointer temporary name.
enum_type	Enum type being initialized.
variant_name	Active variant name.
payload_inits	Vector of alternating field-name and field-value strings.

8.55.1.49 cem_emit_exit_switch()

```
func cem_emit_exit_switch(self:  CEmitter*)
```

Emit a switch-exit `break;`.

Parameters

self	Emitter to update.

8.55.1.50 cem_emit_expr_stmt()

```
func cem_emit_expr_stmt(self:  CEmitter*, c_expr:  string)
```

Emit an expression statement.

Parameters

self	Emitter to update.
c_expr	Lowered expression.

8.55.1.51 cem_emit_field_access()

```
func cem_emit_field_access(c_obj:  string, field_name:  string, is_pointer:  bool) → string
```

Emit a field-access expression.

Parameters

c_obj	Object expression.
field_name	Field identifier.
is_pointer	Whether the object is a pointer.

Returns

Field-access expression using `.` or →.

8.55.1.52 cem_emit_field_lvalue()

```
func cem_emit_field_lvalue(obj:  string, field:  string, is_pointer:  bool) → string
```

Emit a field lvalue.

Parameters

obj	Object expression.
field	Field identifier.
is_pointer	Whether the object is a pointer.

Returns

Lvalue field-access text.

8.55.1.53 cem_emit_for_loop_end()

```
func cem_emit_for_loop_end(self:  CEmitter*)
```

Emit the Stage 1-shaped for-loop wrapper end.

Parameters

self	Emitter to update.

8.55.1.54 cem_emit_for_loop_start()

```
func cem_emit_for_loop_start(self: CEmitter*)
```

Emit the Stage 1-shaped for-loop wrapper start.

Parameters

self	Emitter to update.

8.55.1.55 cem_emit_forward_decls()

```
func cem_emit_forward_decls(self: CEmitter*)
```

Emit forward declarations for all structs and enums in compilation order.

Parameters

self	Emitter to update.

8.55.1.56 cem_emit_function_call()

```
func cem_emit_function_call(c_func_name: string, c_args: string) → string
```

Emit a function-call expression.

Parameters

c_func_name	Function expression.
c_args	Comma-separated lowered arguments.

Returns

Function-call expression.

8.55.1.57 cem_emit_function_declaration()

```
func cem_emit_function_declaration(self: CEmitter*, module_name: string, decl: FuncDecl*,
func_type: Type*)
```

Emit a function declaration.

Parameters

self	Emitter to update.
module_name	Name of the defining module.
decl	Function declaration node.
func_type	Resolved function type.

8.55.1.58 cem_emit_function_definition_footer()

```
func cem_emit_function_definition_footer(self:  CEmitter*)
```

Emit a function definition footer.

Parameters

self	Emitter to update.

8.55.1.59 cem_emit_function_definition_header()

```
func cem_emit_function_definition_header(self:  CEmitter*, module_name:  string, decl:  FuncDecl*,
func_type:  Type*)
```

Emit a function definition header.

Parameters

self	Emitter to update.
module_name	Name of the defining module.
decl	Function declaration node.
func_type	Resolved function type.

8.55.1.60 cem_emit_goto()

```
func cem_emit_goto(self:  CEmitter*, label:  string)
```

Emit a `goto` statement.

Parameters

self	Emitter to update.
label	Target label.

8.55.1.61 cem_emit_header()

```
func cem_emit_header(self:  CEmitter*, opts:  CodegenOptions*)
```

Emit the fixed header boilerplate for one C translation unit.

Parameters

self	Emitter to update.
opts	Codegen options to honor.

8.55.1.62 cem_emit_if_header()

```
func cem_emit_if_header(self:  CEmitter*, c_cond:  string)
```

Emit an `if` header.

Parameters

self	Emitter to update.
c_cond	Lowered condition expression.

8.55.1.63 cem_emit_index_lvalue()

```
func cem_emit_index_lvalue(base:  string, index:  string) → string
```

Emit an index lvalue.

Parameters

base	Base expression.
index	Index expression.

Returns

Indexed lvalue text.

8.55.1.64 cem_emit_int_literal()

```
func cem_emit_int_literal(value:  int) → string
```

Emit an integer literal.

Parameters

value	Integer value.

Returns

C spelling for the literal.

8.55.1.65 cem_emit_label()

```
func cem_emit_label(self:  CEmitter*, label:  string)
```

Emit a C label.

Parameters

self	Emitter to update.
label	Label name.

8.55.1.66 cem_emit_let_decl()

```
func cem_emit_let_decl(self:  CEmitter*, c_type:  string, c_var_name:  string, c_init:  string)
```

Emit a local variable declaration.

Parameters

self	Emitter to update.
c_type	Lowered C type.
c_var_name	Lowered variable name.
c_init	Lowered initializer.

8.55.1.67 cem_emit_let_declaration()

```
func cem_emit_let_declaration(self:  CEmitter*, module_name:  string, decl:  LetDecl*, let_type:
Type*, c_init:  string)
```

Emit one top-level `let` declaration as a static C variable.

Parameters

self	Emitter to update.
module_name	Name of the defining module.
decl	Let declaration node.
let_type	Resolved let type.
c_init	C initializer expression.

8.55.1.68 cem_emit_line_directive()

```
func cem_emit_line_directive(self:  CEmitter*, line:  int, current_module:  string, opts:
CodegenOptions*)
```

Emit a `#line` directive when enabled.

Parameters

self	Emitter to update.
line	One-based source line.
current_module	Current source module.
opts	Codegen options controlling line directives.

8.55.1.69 cem_emit_main_wrapper()

```
func cem_emit_main_wrapper(self: CEmitter*, entry_module: string, func_type: Type*)
```

Emit the C `main()` wrapper for the entry module.

Parameters

self	Emitter to update.
entry_module	Entry-module name.
func_type	Resolved entry-function type.

8.55.1.70 cem_emit_match_scrutinee_decl()

```
func cem_emit_match_scrutinee_decl(self: CEmitter*, c_type: string, c_expr: string)
```

Emit the temporary scrutinee declaration used by `match`/`case`.

Parameters

self	Emitter to update.
c_type	Lowered scrutinee type.
c_expr	Lowered scrutinee expression.

8.55.1.71 cem_emit_match_switch_start()

```
func cem_emit_match_switch_start(self: CEmitter*, scrutinee_name: string)
```

Emit the switch used for enum `match`.

Parameters

self	Emitter to update.
scrutinee_name	Scrutinee variable name.

8.55.1.72 cem_emit_module_comment()

```
func cem_emit_module_comment(self: CEmitter*, module_name: string)
```

Emit a module comment.

Parameters

self	Emitter to update.
module_name	Module name to report.

8.55.1.73 cem_emit_module_separator()

```
func cem_emit_module_separator(self:  CEmitter*, module_name:  string)
```

Emit a decorative module separator.

Parameters

self	Emitter to update.
module_name	Module name to report.

8.55.1.74 cem_emit_negated_condition()

```
func cem_emit_negated_condition(c_cond:  string) → string
```

Emit a negated condition expression for control-flow lowering.

Parameters

c_cond	Lowered condition expression.

Returns

Negated condition expression.

8.55.1.75 cem_emit_none_value_for_nullable()

```
func cem_emit_none_value_for_nullable(t:  Type*) → string
```

Emit a `none` value for a non-pointer nullable type.

Parameters

t	Nullable type to lower.

Returns

C initializer/value for `null`.

8.55.1.76 cem_emit_null_assignment()

```
func cem_emit_null_assignment(self:  CEmitter*, c_var:  string)
```

Emit a NULL assignment.

Parameters

self	Emitter to update.
c_var	Lowered lvalue expression.

8.55.1.77 cem_emit_null_check_eq()

```
func cem_emit_null_check_eq(c_expr:  string) → string
```

Emit a null-equality check for wrapper-shaped nullable values.

Parameters

c_expr	Wrapper expression.

Returns

Equality-check expression.

8.55.1.78 cem_emit_null_check_ne()

```
func cem_emit_null_check_ne(c_expr:  string) → string
```

Emit a null-inequality check for wrapper-shaped nullable values.

Parameters

c_expr	Wrapper expression.

Returns

Inequality-check expression.

8.55.1.79 cem_emit_null_literal()

```
func cem_emit_null_literal(expected_type:  Type*, for_initializer:  bool) → string
```

Emit a `null` literal for the expected type.

Parameters

expected_type	Expected target type.
for_initializer	Whether the literal is used in static initialization.

Returns

C spelling for the literal.

8.55.1.80 cem_emit_optional_has_value()

```
func cem_emit_optional_has_value(c_expr:  string) → string
```

Emit access to an optional wrapper's `has_value` flag.

Parameters

c_expr	Wrapper expression.

Returns

Member-access expression.

8.55.1.81 cem_emit_optional_value()

```
func cem_emit_optional_value(c_expr:  string) → string
```

Emit access to an optional wrapper's payload value.

Parameters

c_expr	Wrapper expression.

Returns

Member-access expression.

8.55.1.82 cem_emit_optional_wrapper()

```
func cem_emit_optional_wrapper(self:  CEmitter*, name:  string, inner:  Type*)
```

Emit one collected optional-wrapper typedef.

Parameters

self	Emitter to update.
name	Wrapper typedef name.
inner	Wrapped type.

8.55.1.83 cem_emit_optional_wrapper_for_defined_type()

```
func cem_emit_optional_wrapper_for_defined_type(self:  CEmitter*, inner:  Type*)
```

Emit the collected optional wrapper for a just-defined user type.

Parameters

self	Emitter to update.
inner	Just-defined struct or enum type.

8.55.1.84 cem_emit_optional_wrappers()

```
func cem_emit_optional_wrappers(self:  CEmitter*, early:  bool)
```

Emit collected optional-wrapper typedefs for one phase.

Parameters

self	Emitter to update.
early	`true` for builtin-only wrappers, `false` for user-defined wrappers.

8.55.1.85 cem_emit_ord()

```
func cem_emit_ord(c_enum_expr:  string) → string
```

Emit C code for `ord(enum_expr)`.

Parameters

c_enum_expr	C expression producing an enum value.

Returns

Casted tag extraction expression.

8.55.1.86 cem_emit_paren_expr()

```
func cem_emit_paren_expr(c_inner:  string) → string
```

Emit a parenthesized expression.

Parameters

c_inner	Inner expression.

Returns

Parenthesized expression.

8.55.1.87 cem_emit_pattern_binding_init()

```
func cem_emit_pattern_binding_init(scrutinee: string, variant: string, field: string) →
string
```

Emit one match-arm payload binding initializer.

Parameters

scrutinee	Scrutinee variable name.
variant	Active variant name.
field	Payload field name.

Returns

Lowered payload access.

8.55.1.88 cem_emit_pointer_assignment()

```
func cem_emit_pointer_assignment(self: CEmitter*, c_ptr_name: string, c_value: string)
```

Emit an assignment through a pointer.

Parameters

self	Emitter to update.
c_ptr_name	C pointer expression.
c_value	Lowered value expression.

8.55.1.89 cem_emit_pointer_null_check()

```
func cem_emit_pointer_null_check(c_expr: string, op: string) → string
```

Emit a pointer null comparison.

Parameters

c_expr	Pointer expression.
op	Comparison operator.

Returns

Comparison expression.

8.55.1.90 cem_emit_pointer_type()

```
func cem_emit_pointer_type(base_type:  Type*) → string
```

Emit a pointer type from a base type.

Parameters

base_type	Base type to wrap.

Returns

Pointer type spelling.

8.55.1.91 cem_emit_return_stmt()

```
func cem_emit_return_stmt(self:  CEmitter*, c_value:  string)
```

Emit a return statement.

Parameters

self	Emitter to update.
c_value	Lowered return value.

8.55.1.92 cem_emit_return_void_stmt()

```
func cem_emit_return_void_stmt(self:  CEmitter*)
```

Emit a `return;` statement.

Parameters

self	Emitter to update.

8.55.1.93 cem_emit_section_comment()

```
func cem_emit_section_comment(self:  CEmitter*, text:  string)
```

Emit a section comment.

Parameters

self	Emitter to update.
text	Comment text.

8.55.1.94 cem_emit_sizeof_type()

```
func cem_emit_sizeof_type(typ:  Type*) → string
```

Emit C code for `sizeof(<type>)`.

Parameters

typ	Type to measure.

Returns

Casted `sizeof` expression.

8.55.1.95 cem_emit_some_value_for_nullable()

```
func cem_emit_some_value_for_nullable(t:  Type*, c_inner_expr:  string) → string
```

Emit a wrapped `some` value for a non-pointer nullable type.

Parameters

t	Nullable type to lower.
c_inner_expr	C expression for the inner value.

Returns

Wrapper construction expression.

8.55.1.96 cem_emit_string_compare_call()

```
func cem_emit_string_compare_call(op:  string, lhs:  string, rhs:  string) → string
```

Emit the runtime string-compare helper call plus a relational check.

Parameters

op	Relational operator token.
lhs	Left operand.
rhs	Right operand.

Returns

Parenthesized comparison expression.

8.55.1.97 cem_emit_string_concat_call()

```
func cem_emit_string_concat_call(lhs:  string, rhs:  string) → string
```

Emit the runtime string-concatenation helper call.

Parameters

lhs	Left operand.
rhs	Right operand.

Returns

Call expression.

8.55.1.98 cem_emit_string_equals_call()

```
func cem_emit_string_equals_call(lhs:  string, rhs:  string) → string
```

Emit the runtime string-equality helper call.

Parameters

lhs	Left operand.
rhs	Right operand.

Returns

Call expression.

8.55.1.99 cem_emit_string_literal()

```
func cem_emit_string_literal(value:  string) → string
```

Emit a decoded string literal through `L0_STRING_CONST`.

Parameters

value	Decoded string bytes.

Returns

C spelling for the runtime string value.

8.55.1.100 cem_emit_string_release()

```
func cem_emit_string_release(self:  CEmitter*, c_expr:  string)
```

Emit an ARC string release call.

Parameters

self	Emitter to update.
c_expr	Lowered string expression.

8.55.1.101 cem_emit_string_retain()

```
func cem_emit_string_retain(self:  CEmitter*, c_expr:  string)
```

Emit an ARC string retain call.

Parameters

self	Emitter to update.
c_expr	Lowered string expression.

8.55.1.102 cem_emit_struct()

```
func cem_emit_struct(self:  CEmitter*, module_name:  string, decl:  StructDecl*, struct_info:
StructInfo*)
```

Emit a complete C struct definition.

Parameters

self	Emitter to update.
module_name	Name of the defining module.
decl	Struct declaration node.
struct_info	Resolved struct metadata.

8.55.1.103 cem_emit_struct_cleanup()

```
func cem_emit_struct_cleanup(self:  CEmitter*, c_ptr_expr:  string, struct_type:  Type*)
```

Emit cleanup for a pointer-to-struct value.

Parameters

self	Emitter to update.
c_ptr_expr	Lowered pointer expression.
struct_type	Struct type to clean.

8.55.1.104 cem_emit_struct_constructor()

```
func cem_emit_struct_constructor(c_struct_name: string, field_inits: StringVector*) → string
```

Emit a struct constructor literal.

Parameters

c_struct_name	Mangled C struct name.
field_inits	Vector of alternating field-name and field-value strings.

Returns

Compound-literal expression.

8.55.1.105 cem_emit_struct_constructor_for_type()

```
func cem_emit_struct_constructor_for_type(struct_type: Type*, field_inits: StringVector*) →
string
```

Emit a struct constructor for a resolved struct type.

Parameters

struct_type	Struct type to construct.
field_inits	Vector of alternating field-name and field-value strings.

Returns

Compound-literal expression.

8.55.1.106 cem_emit_struct_init()

```
func cem_emit_struct_init(self: CEmitter*, c_temp_name: string, c_base_type: string, c_init_str:
string)
```

Emit one pointer-target aggregate initialization.

Parameters

self	Emitter to update.
c_temp_name	Pointer temporary name.
c_base_type	Lowered base type.
c_init_str	Lowered initializer body without outer braces.

8.55.1.107 cem_emit_struct_init_from_fields()

```
func cem_emit_struct_init_from_fields(self: CEmitter*, c_temp_name: string, base_type: Type*,
field_inits: StringVector*)
```

Emit one pointer-target struct initialization from field pairs.

Parameters

self	Emitter to update.
c_temp_name	Pointer temporary name.
base_type	Struct type being initialized.
field_inits	Vector of alternating field-name and field-value strings.

8.55.1.108 cem_emit_struct_static_initializer()

```
func cem_emit_struct_static_initializer(field_inits: StringVector*) → string
```

Emit a brace-only struct initializer for static storage duration.

Parameters

field_inits	Vector of alternating field-name and field-value strings.

Returns

Brace-only initializer text.

8.55.1.109 cem_emit_struct_static_initializer_for_type()

```
func cem_emit_struct_static_initializer_for_type(struct_type: Type*, field_inits: StringVector*)
→ string
```

Emit a brace-only struct initializer for a resolved struct type.

Parameters

struct_type	Struct type to construct.
field_inits	Vector of alternating field-name and field-value strings.

Returns

Brace-only initializer text.

8.55.1.110 cem_emit_switch_end()

```
func cem_emit_switch_end(self:  CEmitter*)
```

Emit the closing brace of a `switch`.

Parameters

self	Emitter to update.

8.55.1.111 cem_emit_switch_start()

```
func cem_emit_switch_start(self:  CEmitter*, c_expr:  string)
```

Emit a `switch` header.

Parameters

self	Emitter to update.
c_expr	Lowered switch expression.

8.55.1.112 cem_emit_temp_decl()

```
func cem_emit_temp_decl(self:  CEmitter*, c_type:  string, c_temp_name:  string, c_value:
string)
```

Emit a temporary declaration.

Parameters

self	Emitter to update.
c_type	Lowered C type.
c_temp_name	Temporary identifier.
c_value	Lowered initializer.

8.55.1.113 cem_emit_top_level_let_cleanup()

```
func cem_emit_top_level_let_cleanup(self:  CEmitter*)
```

Emit final cleanup for ARC-managed top-level lets before process exit.

Parameters

self	Emitter to update.

8.55.1.114 cem_emit_try_check_niche()

```
func cem_emit_try_check_niche(self:  CEmitter*, c_tmp:  string, ret_none:  string)
```

Emit the null-check used by `?` on niche optionals.

Parameters

self	Emitter to update.
c_tmp	Temporary variable name.
ret_none	Lowered `none` return value.

8.55.1.115 cem_emit_try_check_value()

```
func cem_emit_try_check_value(self:  CEmitter*, c_tmp:  string, ret_none:  string)
```

Emit the `has_value` check used by `?` on value optionals.

Parameters

self	Emitter to update.
c_tmp	Temporary variable name.
ret_none	Lowered `none` return value.

8.55.1.116 cem_emit_try_extract_value()

```
func cem_emit_try_extract_value(c_tmp:  string) → string
```

Emit the value extraction used by `?` on value optionals.

Parameters

c_tmp	Temporary variable name.

Returns

Lowered extraction expression.

8.55.1.117 cem_emit_type()

```
func cem_emit_type(typ:  Type*) → string
```

Emit an L0 type as a C type spelling.

Parameters

typ	Type to lower.

Returns

C spelling for `typ`.

8.55.1.118 cem_emit_unary_op()

```
func cem_emit_unary_op(op:  string, c_operand:  string) → string
```

Emit a unary operation expression.

Parameters

op	Unary operator token.
c_operand	Lowered operand expression.

Returns

Parenthesized C unary expression.

8.55.1.119 cem_emit_unreachable_comment()

```
func cem_emit_unreachable_comment(self:  CEmitter*)
```

Emit an unreachable-code comment.

Parameters

self	Emitter to update.

8.55.1.120 cem_emit_unreachable_marker()

```
func cem_emit_unreachable_marker(self:  CEmitter*, reason:  string)
```

Emit an unreachable runtime marker.

Parameters

self	Emitter to update.
reason	Marker reason string.

8.55.1.121 cem_emit_unwrap_opt()

```
func cem_emit_unwrap_opt(c_src_type:  string, c_inner:  string, type_str:  string) → string
```

Emit an unwrap call for wrapper-shaped nullable values.

Parameters

c_src_type	Source wrapper type.
c_inner	Inner nullable expression.
type_str	Human-readable source type.

Returns

Unwrap expression.

8.55.1.122 cem_emit_unwrap_ptr()

```
func cem_emit_unwrap_ptr(c_dst_type:  string, c_inner:  string, type_str:  string) → string
```

Emit an unwrap call for pointer-shaped nullable values.

Parameters

c_dst_type	Destination C type.
c_inner	Inner nullable expression.
type_str	Human-readable source type.

Returns

Unwrap expression.

8.55.1.123 cem_emit_value_cleanup()

```
func cem_emit_value_cleanup(self:  CEmitter*, c_expr:  string, ty:  Type*)
```

Emit cleanup for a by-value expression according to its type.

Parameters

self	Emitter to update.
c_expr	Lowered value expression.
ty	Type of the value.

8.55.1.124 cem_emit_var_ref()

```
func cem_emit_var_ref(c_name:  string) → string
```

Emit a variable reference expression.

Parameters

c_name	C identifier to reference.

Returns

The identifier text.

8.55.1.125 cem_emit_variant_constructor()

```
func cem_emit_variant_constructor(c_enum_name:  string, variant_name:  string, tag_value:
string, payload_inits:  StringVector*) → string
```

Emit an enum-variant constructor literal.

Parameters

c_enum_name	Mangled C enum name.
variant_name	Variant name.
tag_value	Mangled tag constant.
payload_inits	Vector of alternating field-name and field-value strings.

Returns

Tagged-union literal expression.

8.55.1.126 cem_emit_variant_constructor_for_type()

```
func cem_emit_variant_constructor_for_type(enum_type:  Type*, variant_name:  string, payload_inits:
StringVector*) → string
```

Emit an enum-variant constructor for a resolved enum type.

Parameters

enum_type	Enum type to construct.
variant_name	Variant name.
payload_inits	Vector of alternating field-name and field-value strings.

Returns

Tagged-union literal expression.

8.55.1.127 cem_emit_variant_static_initializer()

```
func cem_emit_variant_static_initializer(variant_name:  string, tag_value:  string, payload_inits:
StringVector*) → string
```

Emit a brace-only enum-variant initializer for static storage duration.

Parameters

variant_name	Variant name.
tag_value	Mangled tag constant.
payload_inits	Vector of alternating field-name and field-value strings.

Returns

Brace-only tagged-union initializer.

8.55.1.128 cem_emit_variant_static_initializer_for_type()

```
func cem_emit_variant_static_initializer_for_type(enum_type:  Type*, variant_name:  string,
payload_inits:  StringVector*) → string
```

Emit a brace-only enum-variant initializer for a resolved enum type.

Parameters

enum_type	Enum type to construct.
variant_name	Variant name.
payload_inits	Vector of alternating field-name and field-value strings.

Returns

Brace-only tagged-union initializer.

8.55.1.129 cem_emit_while_header()

```
func cem_emit_while_header(self:  CEmitter*, c_cond:  string)
```

Emit a `while` header.

Parameters

self	Emitter to update.
c_cond	Lowered condition expression.

8.55.1.130 cem_emit_widen_int()

```
func cem_emit_widen_int(c_expr:  string, src_type:  Type*, dst_type:  Type*) → string
```

Emit a widening cast for an implicit integer conversion.

Parameters

c_expr	Source expression.
src_type	Source builtin type.
dst_type	Destination builtin type.

Returns

Widened C expression.

8.55.1.131 cem_emit_zero_init()

```
func cem_emit_zero_init(self:  CEmitter*, c_temp_name:  string, c_base_type:  string)
```

Emit zero-initialization through a pointer.

Parameters

self	Emitter to update.
c_temp_name	Pointer temporary name.
c_base_type	Lowered base type.

8.55.1.132 cem_encode_c_byte_literal()

```
func cem_encode_c_byte_literal(value:  byte) → string
```

Encode one byte for a C character literal body.

Parameters

value	Byte value.

Returns

Escaped character-literal body without surrounding quotes.

8.55.1.133 cem_enum_has_arc_data()

```
func cem_enum_has_arc_data(self:  CEmitter*, enum_info:  EnumInfo*, decl:  EnumDecl*) → bool
```

Report whether any variant field in an enum carries ARC-managed data.

Parameters

self	Emitter to inspect.
enum_info	Resolved enum metadata.
decl	Parsed enum declaration.

Returns

`true` when cleanup code must inspect the active variant.

8.55.1.134 cem_find_enum_decl()

```
func cem_find_enum_decl(self: CEmitter*, module_name: string, enum_name: string) → EnumDecl*?
```

Look up a loaded enum declaration by module and name.

Parameters

self	Emitter to inspect.
module_name	Defining module name.
enum_name	Enum name.

Returns

Matching declaration, or `null` if absent.

8.55.1.135 cem_find_enum_variant_info()

```
func cem_find_enum_variant_info(enum_info: EnumInfo*, variant_name: string) → EnumVariantInfo*?
```

Look up enum payload metadata by variant name.

Parameters

enum_info	Enum metadata to inspect.
variant_name	Variant name to search for.

Returns

Matching variant metadata, or `null` if missing.

8.55.1.136 cem_free()

```
func cem_free(self: CEmitter*)
```

Free a Stage 2 C emitter.

Parameters

self	Emitter to free.

8.55.1.137 cem_fresh_tmp()

```
func cem_fresh_tmp(self:  CEmitter*, kind:  string) → string
```

Produce a fresh temporary identifier.

Parameters

self	Emitter to update.
kind	Temp-kind prefix.

Returns

Unique temp identifier.

8.55.1.138 cem_function_params()

```
func cem_function_params(decl:  FuncDecl*, func_type:  Type*) → string
```

Build a function parameter list.

Parameters

decl	Function declaration node.
func_type	Resolved function type.

Returns

Parameter-list text without surrounding parentheses.

8.55.1.139 cem_get_output()

```
func cem_get_output(self:  CEmitter*) → string
```

Return the generated C source text.

Parameters

self	Emitter to inspect.

Returns

Complete source text.

8.55.1.140 cem_ice()

```
func cem_ice(message:  string)
```

Abort on an internal code-emission error.

Parameters

message	ICE message to report.

8.55.1.141 cem_ice_string()

```
func cem_ice_string(message:  string) → string
```

Abort on an internal code-emission error from a string-returning helper.

Parameters

message	ICE message to report.

Returns

Dummy return used only to satisfy control-flow typing.

8.55.1.142 cem_is_c_keyword()

```
func cem_is_c_keyword(name:  string) → bool
```

Return whether a name conflicts with a C keyword or restricted identifier.

Parameters

name	Identifier text to inspect.

Returns

`true` if the name must be mangled.

8.55.1.143 cem_is_early_inner()

```
func cem_is_early_inner(inner:  Type*) → bool
```

Return whether a wrapper inner type can be emitted before user type definitions.

Parameters

inner	Inner wrapped type.

Returns

`true` for builtin-only wrapper dependencies.

8.55.1.144 cem_is_niche_nullable()

```
func cem_is_niche_nullable(t:  Type*) → bool
```

Return whether a nullable type uses pointer-shaped niche encoding.

Parameters

t	Nullable type to inspect.

Returns

`true` when `null` lowers to `NULL`.

8.55.1.145 cem_mangle_enum_name()

```
func cem_mangle_enum_name(module_name:  string, enum_name:  string) → string
```

Mangle an enum name.

Parameters

module_name	Defining module name.
enum_name	Enum name.

Returns

Mangled C enum name.

8.55.1.146 cem_mangle_function_name()

```
func cem_mangle_function_name(module_name:  string, func_name:  string) → string
```

Mangle a function name.

Parameters

module_name	Defining module name.
func_name	Function name.

Returns

Mangled C function name.

8.55.1.147 cem_mangle_identifier()

```
func cem_mangle_identifier(name:  string) → string
```

Mangle a local identifier when needed.

Parameters

name	Identifier to inspect.

Returns

Original or suffixed identifier.

8.55.1.148 cem_mangle_let_name()

```
func cem_mangle_let_name(module_name:  string, let_name:  string) → string
```

Mangle a top-level let name.

Parameters

module_name	Defining module name.
let_name	Let name.

Returns

Mangled C identifier.

8.55.1.149 cem_mangle_module()

```
func cem_mangle_module(module_name:  string) → string
```

Replace dots with underscores for C symbol names.

Parameters

module_name	Module name to sanitize.

Returns

C-safe module spelling.

8.55.1.150 cem_mangle_struct_name()

```
func cem_mangle_struct_name(module_name:  string, struct_name:  string) → string
```

Mangle a struct name.

Parameters

module_name	Defining module name.
struct_name	Struct name.

Returns

Mangled C struct name.

8.55.1.151 cem_module_filename()

```
func cem_module_filename(self:  CEmitter*, module_name:  string) → string
```

Return the source filename for a loaded module.

Parameters

self	Emitter to inspect.
module_name	Loaded module name.

Returns

Source filename when available.

8.55.1.152 cem_note_opt_wrapper()

```
func cem_note_opt_wrapper(self:  CEmitter*, inner:  Type*)
```

Record one optional-wrapper inner type if it was not already collected.

Parameters

self	Emitter to update.
inner	Inner type to store.

8.55.1.153 cem_opt_key_for_type()

```
func cem_opt_key_for_type(t:  Type*) → string
```

Produce the stable wrapper-key suffix for an optional inner type.

Parameters

t	Inner type to encode.

Returns

Stable wrapper-key suffix.

8.55.1.154 cem_opt_wrapper_map_free()

```
func cem_opt_wrapper_map_free(items:  StringPtrMap*)
```

Free every collected optional-wrapper inner type.

Parameters

items	Wrapper map to free.

8.55.1.155 cem_opt_wrapper_name_for_inner()

```
func cem_opt_wrapper_name_for_inner(inner:  Type*) → string
```

Return the wrapper typedef name used for a non-pointer nullable type.

Parameters

inner	Inner type of the nullable wrapper.

Returns

Wrapper typedef name.

8.55.1.156 cem_prepare_optional_wrappers()

```
func cem_prepare_optional_wrappers(self:  CEmitter*)
```

Scan analysis metadata and collect every needed optional wrapper.

Parameters

self	Emitter to update.

8.55.1.157 cem_reset_optional_wrappers()

```
func cem_reset_optional_wrappers(self:  CEmitter*)
```

Clear every collected optional-wrapper entry.

Parameters

self	Emitter to update.

8.55.1.158 cem_restore_active_line_directive()

```
func cem_restore_active_line_directive(self:  CEmitter*)
```

Re-emit the current source line directive after generated helper lines.

Parameters

self	Emitter to update.

8.55.1.159 cem_set_analysis()

```
func cem_set_analysis(self:  CEmitter*, analysis:  AnalysisResult*)
```

Attach analysis data to an emitter.

Parameters

self	Emitter to update.
analysis	Analysis result to borrow.

8.55.1.160 cem_upper_ascii()

```
func cem_upper_ascii(text:  string) → string
```

Build an all-uppercase ASCII variant of an identifier-like string.

Parameters

text	Input text.

Returns

Uppercased ASCII text.

8.56 compiler/stage2_l0/src/cli_args.l0 File Reference

Module: `cli_args`

```
import build_info;
import util.diag;
import std.vector;
import std.text;
import std.string;
import std.io;
```

Classes

- struct CliOptions

  Represents the CliOptions structure.
- struct CliParseResult

  Represents the CliParseResult structure.

Enumerations

- enum CliMode

  Defines the CliMode enumeration.

Functions

- func cli_options_create() → CliOptions∗

  Create a `CliOptions` instance with default values.
- func cli_options_free(self: CliOptions∗)

  Free a `CliOptions` instance.
- func cli_options_free_opt(self: CliOptions∗?)

  Free a nullable `CliOptions` instance when present.
- func cli_diag_free_opt(self: DiagCollector∗?)

  Free a nullable diagnostic collector when present.
- func cli_parse_result_free(self: CliParseResult∗)

  Free a CLI parse result and any owned options.
- func cli_emit_error(diags: DiagCollector∗, code: string, message: string)

  Emit a command-line diagnostic without source coordinates.
- func cli_emit_reserved_option_error(diags: DiagCollector∗, option: string, purpose: string)

  Emit the standard diagnostic for a recognized canonical option whose behavior is not implemented yet.
- func cli_has_diagnostic_code(diags: DiagCollector∗, code: string) → bool

  Return whether a diagnostic collector contains one diagnostic code.
- func cli_compiler_identity_text() → string

  Return the user-facing compiler identity/version text.
- func cli_mode_flag(mode: CliMode) → string

  Return the canonical long flag for a CLI mode.
- func cli_mode_name(mode: CliMode) → string

  Return the short display name for a CLI mode.
- func cli_try_set_mode(opts: CliOptions∗, diags: DiagCollector∗, mode: CliMode)

  Set the active CLI mode or emit an error for conflicting mode flags.
- func cli_split_index(argv: StringVector∗) → int

  Return the index of the -- argument separator.
- func cli_has_help_flag(argv: StringVector∗, cli_end: int) → bool

  Report whether the compiler CLI args explicitly request help output.

- func cli_has_version_flag(argv: StringVector∗, cli_end: int) → bool

  Report whether the compiler CLI args explicitly request version output.
- func cli_print_line(to_stderr: bool, text: string)

  Print one CLI help/usage line to standard output or standard error.
- func cli_print_blank_line(to_stderr: bool)

  Print a blank line to standard output or standard error.
- func cli_print_usage(to_stderr: bool)

  Print the short Stage 2 usage banner.
- func cli_print_help()

  Print the full Stage 2 help text.
- func cli_print_version()

  Print the compiler identity/version text.
- func cli_verbose_cluster_count(token: string) → int

  Count verbosity flags in a short option cluster.
- func cli_require_next_value(argv: StringVector∗, i: int, option: string, diags: DiagCollector∗) → string?

  Return the value following an option that requires one.
- func cli_assign_target(opts: CliOptions∗, token: string)

  Assign a positional target argument.
- func cli_validate_mode_scoped_flags(opts: CliOptions∗, diags: DiagCollector∗)

  Validate that mode-specific flags are used with compatible modes.
- func cli_parse(argv: StringVector∗) → CliParseResult∗

  Parse command-line arguments into structured options.

8.56.1 Enumeration Type Documentation

8.56.1.1 CliMode

```
enum CliMode
```

Defines the CliMode enumeration.

Enumerator

CM_RUN	
CM_BUILD	
CM_COMPILE	
CM_GEN	
CM_CHECK	
CM_TOK	
CM_AST	
CM_SYM	
CM_TYPE	

8.56.2 Function Documentation

8.56.2.1 cli_assign_target()

```
func cli_assign_target(opts:  CliOptions∗, token:  string)
```

Assign a positional target argument.

Parameters

opts	Parsed CLI options.
token	Positional argument to record.

8.56.2.2 cli_compiler_identity_text()

```
func cli_compiler_identity_text() → string
```

Return the user-facing compiler identity/version text.

Returns

Stable compiler identity text for help and `--version`.

8.56.2.3 cli_diag_free_opt()

```
func cli_diag_free_opt(self: DiagCollector*?)
```

Free a nullable diagnostic collector when present.

Parameters

self	Optional diagnostic collector to free.

8.56.2.4 cli_emit_error()

```
func cli_emit_error(diags: DiagCollector*, code: string, message: string)
```

Emit a command-line diagnostic without source coordinates.

Parameters

diags	Diagnostic collector.
code	Diagnostic code.
message	Diagnostic message.

8.56.2.5 cli_emit_reserved_option_error()

```
func cli_emit_reserved_option_error(diags: DiagCollector*, option: string, purpose: string)
```

Emit the standard diagnostic for a recognized canonical option whose behavior is not implemented yet.

Parameters

diags	Diagnostic collector.
option	Canonical short option.
purpose	Reserved option purpose.

8.56.2.6 cli_has_diagnostic_code()

```
func cli_has_diagnostic_code(diags:  DiagCollector*, code:  string) → bool
```

Return whether a diagnostic collector contains one diagnostic code.

Parameters

diags	Diagnostic collector to inspect.
code	Diagnostic code to find.

Returns

`true` when the code is present.

8.56.2.7 cli_has_help_flag()

```
func cli_has_help_flag(argv:  StringVector*, cli_end:  int) → bool
```

Report whether the compiler CLI args explicitly request help output.

Only tokens before the `--` separator count as compiler CLI arguments.

Parameters

argv	Full argument vector excluding the executable name.
cli_end	Exclusive end index for compiler CLI arguments.

Returns

`true` when `-h` or `--help` is present before `cli_end`.

8.56.2.8 cli_has_version_flag()

```
func cli_has_version_flag(argv:  StringVector*, cli_end:  int) → bool
```

Report whether the compiler CLI args explicitly request version output.

Only tokens before the `--` separator count as compiler CLI arguments.

Parameters

argv	Full argument vector excluding the executable name.
cli_end	Exclusive end index for compiler CLI arguments.

Returns

`true` when `-V` or `--version` is present before `cli_end`.

8.56.2.9 cli_mode_flag()

```
func cli_mode_flag(mode: CliMode) → string
```

Return the canonical long flag for a CLI mode.

Parameters

mode	CLI mode to format.

Returns

Long-form flag such as `--check`.

8.56.2.10 cli_mode_name()

```
func cli_mode_name(mode: CliMode) → string
```

Return the short display name for a CLI mode.

Parameters

mode	CLI mode to format.

Returns

Short mode name such as `check`.

8.56.2.11 cli_options_create()

```
func cli_options_create() → CliOptions*
```

Create a `CliOptions` instance with default values.

Returns

New CLI options object.

8.56.2.12 cli_options_free()

```
func cli_options_free(self: CliOptions*)
```

Free a `CliOptions` instance.

Parameters

self	CLI options to free.

8.56.2.13 cli_options_free_opt()

```
func cli_options_free_opt(self: CliOptions*?)
```

Free a nullable `CliOptions` instance when present.

Parameters

self	Optional CLI options to free.

8.56.2.14 cli_parse()

```
func cli_parse(argv: StringVector*) → CliParseResult*
```

Parse command-line arguments into structured options.

Parameters

argv	Argument vector excluding the executable name.

Returns

Parsed options together with any diagnostics produced during parsing.

8.56.2.15 cli_parse_result_free()

```
func cli_parse_result_free(self: CliParseResult*)
```

Free a CLI parse result and any owned options.

Parameters

self	Parse result to free.

8.56.2.16 cli_print_blank_line()

```
func cli_print_blank_line(to_stderr:  bool)
```

Print a blank line to standard output or standard error.

Parameters

to_stderr	Whether the newline should be written to standard error.

8.56.2.17 cli_print_help()

```
func cli_print_help()
```

Print the full Stage 2 help text.

8.56.2.18 cli_print_line()

```
func cli_print_line(to_stderr:  bool, text:  string)
```

Print one CLI help/usage line to standard output or standard error.

Parameters

to_stderr	Whether the text should be written to standard error.
text	Line text without a trailing newline.

8.56.2.19 cli_print_usage()

```
func cli_print_usage(to_stderr:  bool)
```

Print the short Stage 2 usage banner.

Parameters

to_stderr	Whether the banner should be written to standard error.

8.56.2.20 cli_print_version()

```
func cli_print_version()
```

Print the compiler identity/version text.

8.56.2.21 cli_require_next_value()

```
func cli_require_next_value(argv: StringVector*, i: int, option: string, diags: DiagCollector*)
→ string?
```

Return the value following an option that requires one.

Parameters

argv	Full argument vector.
i	Index of the option token.
option	Exact option token used in diagnostics.
diags	Diagnostic collector.

Returns

Following argument value, or `null` after emitting a missing-value error.

8.56.2.22 cli_split_index()

```
func cli_split_index(argv: StringVector*) → int
```

Return the index of the `--` argument separator.

Parameters

argv	Full argument vector.

Returns

Separator index, or `-1` if `--` is not present.

8.56.2.23 cli_try_set_mode()

```
func cli_try_set_mode(opts: CliOptions*, diags: DiagCollector*, mode: CliMode)
```

Set the active CLI mode or emit an error for conflicting mode flags.

Parameters

opts	Parsed CLI options.
diags	Diagnostic collector.
mode	Mode requested by the current flag.

8.56.2.24 cli_validate_mode_scoped_flags()

```
func cli_validate_mode_scoped_flags(opts: CliOptions*, diags: DiagCollector*)
```

Validate that mode-specific flags are used with compatible modes.

Parameters

opts	Parsed CLI options.
diags	Diagnostic collector for incompatibility errors.

8.56.2.25 cli_verbose_cluster_count()

```
func cli_verbose_cluster_count(token: string) → int
```

Count verbosity flags in a short option cluster.

Parameters

token	Argument token to inspect.

Returns

Number of `v` characters in a token like `-vvv`, or `-1` if the token is not such a cluster.

8.57 compiler/stage2_l0/src/codegen_options.l0 File Reference

Module: `codegen_options`

Classes

- struct CodegenOptions

  Represents backend and emitter options that affect generated C text.

Functions

- func cg_options_create() → CodegenOptions*

  Create codegen options with Stage 1-aligned defaults.
- func cg_options_free(self: CodegenOptions*)

  Free a codegen-options object.

8.57.1 Function Documentation

8.57.1.1 cg_options_create()

```
func cg_options_create() → CodegenOptions*
```

Create codegen options with Stage 1-aligned defaults.

Returns

New codegen options object.

8.57.1.2 cg_options_free()

```
func cg_options_free(self:  CodegenOptions*)
```

Free a codegen-options object.

Parameters

self	Options object to free.

8.58 compiler/stage2_l0/src/compiler_filesystem.l0 File Reference

Module: `compiler_filesystem`

```
import std.vector;
import util.diag;
import std.text;
import std.time;
import std.system;
import std.string;
import std.array;
import sys.rt;
```

Classes

- struct CompilerWorkspace

  Invocation-owned native build/run workspace.

Functions

- extern func l0c_fs_mkdir(path: byte*, path_len: int, mode: int) → int
- extern func l0c_fs_join_child(parent: byte*, parent_len: int, child: byte*, child_len: int, output: byte*?, output_capacity: int) → int
- extern func l0c_fs_path_kind_nofollow(path: byte*, path_len: int) → int

- extern func l0c_fs_path_kind_follow(path: byte∗, path_len: int) → int
- extern func l0c_fs_remove_regular_file(path: byte∗, path_len: int) → int
- extern func l0c_fs_remove_empty_dir(path: byte∗, path_len: int) → int
- extern func l0c_fs_resolve_trusted_temp_parent(path: byte∗, path_len: int, output: byte∗?, output_capacity: int) → int
- func cfs_error(diags: DiagCollector∗, code: string, message: string, path: string)

  Emit one compiler-filesystem diagnostic without source coordinates.
- func cfs_path_kind(path: string) → int

  Call the compiler-internal no-follow path classifier.
- func cfs_path_kind_follow(path: string) → int

  Call the compiler-internal follow path classifier.
- func cfs_join_child(parent: string, child: string) → string

  Join a trusted parent and internal child using actual-host separators.
- func cfs_mkdir(path: string, mode: int) → int

  Create one directory using the compiler-internal raw-byte ABI.
- func cfs_remove_regular_file(path: string) → int

  Remove one regular file without following a substituted path.
- func cfs_remove_empty_dir(path: string) → int

  Remove one empty real directory through the compiler-internal ABI.
- func cfs_resolve_trusted_temp_parent(path: string) → string?

  Resolve and validate one selected temporary parent.
- func cfs_select_temp_parent_from_values(tmpdir: string?, temp: string?, tmp: string?, system_tmp: string, fallback: string, diags: DiagCollector∗) → string?

  Select a temporary parent from explicit candidate values.
- func cfs_select_temp_parent(diags: DiagCollector∗) → string?

  Return the first existing temporary-parent directory by stable precedence.
- func cfs_workspace_free(self: CompilerWorkspace∗)

  Release a workspace path bundle after explicit cleanup.
- func cfs_workspace_child(workspace: CompilerWorkspace∗, leaf: string) → string

  Register and return one fixed driver-selected workspace child.
- func cfs_workspace_create_for_identity(parent_path: string, tag: string, pid: int, sec: int, nsec: int, diags: DiagCollector∗) → CompilerWorkspace∗?

  Exclusively reserve one workspace from a deterministic identity tuple.
- func cfs_workspace_create(tag: string, diags: DiagCollector∗) → CompilerWorkspace∗?

  Select, resolve, validate, and exclusively reserve one compiler workspace.
- func cfs_workspace_cleanup(workspace: CompilerWorkspace∗, diags: DiagCollector∗) → bool

  Remove only known regular children, then the empty owned directory.
- func cfs_apply_cleanup_status(primary_status: int, cleanup_ok: bool) → int

  Apply cleanup failure to a command's already-determined status.

Variables

- let CFS_PATH_ABSENT: int = 0

  No filesystem object exists at the path.
- let CFS_PATH_REGULAR: int = 1

  A regular file exists at the path.
- let CFS_PATH_DIRECTORY: int = 2

  A directory exists at the path.
- let CFS_PATH_OTHER: int = 3

  A symlink, reparse point, or other unsupported object exists at the path.

8.58.1 Function Documentation

8.58.1.1 cfs_apply_cleanup_status()

```
func cfs_apply_cleanup_status(primary_status:  int, cleanup_ok:  bool) → int
```

Apply cleanup failure to a command's already-determined status.

Cleanup turns success into status 1, while an existing failure or program status remains authoritative.

8.58.1.2 cfs_error()

```
func cfs_error(diags:  DiagCollector*, code:  string, message:  string, path:  string)
```

Emit one compiler-filesystem diagnostic without source coordinates.

8.58.1.3 cfs_join_child()

```
func cfs_join_child(parent:  string, child:  string) → string
```

Join a trusted parent and internal child using actual-host separators.

8.58.1.4 cfs_mkdir()

```
func cfs_mkdir(path:  string, mode:  int) → int
```

Create one directory using the compiler-internal raw-byte ABI.

8.58.1.5 cfs_path_kind()

```
func cfs_path_kind(path:  string) → int
```

Call the compiler-internal no-follow path classifier.

8.58.1.6 cfs_path_kind_follow()

```
func cfs_path_kind_follow(path:  string) → int
```

Call the compiler-internal follow path classifier.

8.58.1.7 cfs_remove_empty_dir()

```
func cfs_remove_empty_dir(path:  string) → int
```

Remove one empty real directory through the compiler-internal ABI.

8.58.1.8 cfs_remove_regular_file()

```
func cfs_remove_regular_file(path:  string) → int
```

Remove one regular file without following a substituted path.

8.58.1.9 cfs_resolve_trusted_temp_parent()

```
func cfs_resolve_trusted_temp_parent(path:  string) → string?
```

Resolve and validate one selected temporary parent.

The support ABI determines the actual host. POSIX validates the complete resolved owner/sticky-bit chain; Windows retains the trusted-parent ACL assumption. Platform-alias environment variables are deliberately ignored.

Parameters

path	Selected existing directory.

Returns

Canonical trusted path, or `null` on failure.

8.58.1.10 cfs_select_temp_parent()

```
func cfs_select_temp_parent(diags: DiagCollector*) → string?
```

Return the first existing temporary-parent directory by stable precedence.

8.58.1.11 cfs_select_temp_parent_from_values()

```
func cfs_select_temp_parent_from_values(tmpdir: string?, temp: string?, tmp: string?, system_tmp:
string, fallback: string, diags: DiagCollector*) → string?
```

Select a temporary parent from explicit candidate values.

Empty values, absent paths, and non-directories fall through. Inspection errors are fatal; once a directory is selected, canonical resolution and trust validation are also authoritative.

The explicit `system_tmp` and `fallback` parameters are an internal test seam. Production supplies `/tmp` and `.` respectively.

8.58.1.12 cfs_workspace_child()

```
func cfs_workspace_child(workspace: CompilerWorkspace*, leaf: string) → string
```

Register and return one fixed driver-selected workspace child.

Parameters

workspace	Workspace that owns the child.
leaf	Fixed child filename.

Returns

Joined child path.

8.58.1.13 cfs_workspace_cleanup()

```
func cfs_workspace_cleanup(workspace: CompilerWorkspace*, diags: DiagCollector*) → bool
```

Remove only known regular children, then the empty owned directory.

Cleanup first verifies the workspace root without following aliases. An unexpected child or substituted path retains the workspace for inspection.

8.58.1.14 cfs_workspace_create()

```
func cfs_workspace_create(tag:  string, diags:  DiagCollector*) → CompilerWorkspace*?
```

Select, resolve, validate, and exclusively reserve one compiler workspace.

8.58.1.15 cfs_workspace_create_for_identity()

```
func cfs_workspace_create_for_identity(parent_path:  string, tag:  string, pid:  int, sec:
int, nsec:  int, diags:  DiagCollector*) → CompilerWorkspace*?
```

Exclusively reserve one workspace from a deterministic identity tuple.

Pre-existing candidates are skipped and no unchecked fallback is returned.

8.58.1.16 cfs_workspace_free()

```
func cfs_workspace_free(self:  CompilerWorkspace*)
```

Release a workspace path bundle after explicit cleanup.

8.58.1.17 l0c_fs_join_child()

```
extern func l0c_fs_join_child(parent:  byte*, parent_len:  int, child:  byte*, child_len:  int,
output:  byte*?, output_capacity:  int) → int
```

8.58.1.18 l0c_fs_mkdir()

```
extern func l0c_fs_mkdir(path:  byte*, path_len:  int, mode:  int) → int
```

8.58.1.19 l0c_fs_path_kind_follow()

```
extern func l0c_fs_path_kind_follow(path:  byte*, path_len:  int) → int
```

8.58.1.20 l0c_fs_path_kind_nofollow()

```
extern func l0c_fs_path_kind_nofollow(path:  byte*, path_len:  int) → int
```

8.58.1.21 l0c_fs_remove_empty_dir()

```
extern func l0c_fs_remove_empty_dir(path:  byte*, path_len:  int) → int
```

8.58.1.22 l0c_fs_remove_regular_file()

```
extern func l0c_fs_remove_regular_file(path:  byte*, path_len:  int) → int
```

8.58.1.23 l0c_fs_resolve_trusted_temp_parent()

```
extern func l0c_fs_resolve_trusted_temp_parent(path:  byte*, path_len:  int, output:  byte*?,
output_capacity:  int) → int
```

8.58.2 Variable Documentation

8.58.2.1 CFS_PATH_ABSENT

```
let CFS_PATH_ABSENT: int = 0
```

No filesystem object exists at the path.

8.58.2.2 CFS_PATH_DIRECTORY

```
let CFS_PATH_DIRECTORY: int = 2
```

A directory exists at the path.

8.58.2.3 CFS_PATH_OTHER

```
let CFS_PATH_OTHER: int = 3
```

A symlink, reparse point, or other unsupported object exists at the path.

8.58.2.4 CFS_PATH_REGULAR

```
let CFS_PATH_REGULAR: int = 1
```

A regular file exists at the path.

8.59 compiler/stage2_l0/src/diag_print.l0 File Reference

Module: `diag_print`

```
import std.vector;
import util.diag;
import std.text;
import std.string;
import std.io;
import std.fs;
import std.integer;
import sys.rt;
```

Functions

- func dp_severity_name(sev: DiagnosticSeverity) → string

  Return the lowercase display name for a diagnostic severity.
- func dp_format_header(diag: Diagnostic∗) → string

  Format the first line of a diagnostic message.
- func dp_print_header(diag: Diagnostic∗)

  Prints the dp print header.
- func dp_render_snippet(diag: Diagnostic∗, src: string)

  Render a source snippet for a diagnostic.
- func dp_print_snippet(diag: Diagnostic∗)

  Prints the dp print snippet.
- func dp_source_cached(cache_names: StringVector∗, cache_texts: StringVector∗, filename: string) → string

  Return cached source text for a filename, loading and caching it on demand.
- func dp_print_diag(diag: Diagnostic∗)

  Prints the dp print diag.
- func dp_print_collector_with_sources(dc: DiagCollector∗, source_names: StringVector∗, source_texts: StringVector∗)

  Print all diagnostics using caller-provided source-text caches.
- func dp_print_collector(dc: DiagCollector∗)

  Prints the dp print collector.

8.59.1 Function Documentation

8.59.1.1 dp_format_header()

```
func dp_format_header(diag: Diagnostic*) → string
```

Format the first line of a diagnostic message.

Parameters

diag	Diagnostic to format.

Returns

Header string containing location, severity, code, and message.

8.59.1.2 dp_print_collector()

```
func dp_print_collector(dc: DiagCollector*)
```

Prints the dp print collector.

Parameters

dc	The dc.

8.59.1.3 dp_print_collector_with_sources()

```
func dp_print_collector_with_sources(dc: DiagCollector*, source_names: StringVector*, source_texts: StringVector*)
```

Print all diagnostics using caller-provided source-text caches.

Parameters

dc	Diagnostic collector to print.
source_names	Cached filenames aligned with `source_texts`.
source_texts	Cached source texts aligned with `source_names`.

8.59.1.4 dp_print_diag()

```
func dp_print_diag(diag: Diagnostic*)
```

Prints the dp print diag.

Parameters

diag	The diag.

8.59.1.5 dp_print_header()

```
func dp_print_header(diag: Diagnostic*)
```

Prints the dp print header.

Parameters

diag	The diag.

8.59.1.6 dp_print_snippet()

```
func dp_print_snippet(diag: Diagnostic*)
```

Prints the dp print snippet.

Parameters

diag	The diag.

8.59.1.7 dp_render_snippet()

```
func dp_render_snippet(diag:  Diagnostic*, src:  string)
```

Render a source snippet for a diagnostic.

Parameters

diag	Diagnostic to render.
src	Source text corresponding to `diag.filename`.

8.59.1.8 dp_severity_name()

```
func dp_severity_name(sev:  DiagnosticSeverity) → string
```

Return the lowercase display name for a diagnostic severity.

Parameters

sev	Severity to format.

Returns

Lowercase severity name.

8.59.1.9 dp_source_cached()

```
func dp_source_cached(cache_names:  StringVector*, cache_texts:  StringVector*, filename:
string) → string
```

Return cached source text for a filename, loading and caching it on demand.

Parameters

cache_names	Cached filenames.
cache_texts	Cached source texts aligned with `cache_names`.
filename	Filename to load.

Returns

Cached or newly loaded source text, or an empty string on read failure.

8.60 compiler/stage2_l0/src/driver.l0 File Reference

Module: `driver`

```
import parser;
import source_paths;
import lexer;
import util.log;
import ast;
import util.strings;
import tokens;
import util.diag;
import std.vector;
import std.string;
import std.text;
import std.fs;
import sys.rt;
```

Classes

- struct DriverUnit

  Represents the DriverUnit structure.
- struct DriverState

  Represents the DriverState structure.

Functions

- func dr_unit_free(self: DriverUnit∗)

  Free a loaded driver unit and its parse result.
- func dr_units_free(units: VectorBase∗?)

  Free an optional vector of driver units.
- func driver_state_free(self: DriverState∗)

  Free driver state and all owned parse/cache data.
- func driver_has_errors(self: DriverState∗) → bool

  Report whether driver analysis produced any errors.
- func driver_diag_count(self: DriverState∗) → int

  Return the number of diagnostics collected by the driver.
- func driver_diag_get(self: DriverState∗, index: int) → Diagnostic∗

  Return a driver diagnostic by index.
- func driver_unit_count(self: DriverState∗) → int

  Return the number of loaded driver units.
- func driver_unit_get(self: DriverState∗, index: int) → DriverUnit∗

  Return a loaded driver unit by index.
- func driver_find_unit(self: DriverState∗, module_name: string) → DriverUnit∗?

  Find a loaded driver unit by module name.
- func driver_module_names_sorted(self: DriverState∗) → StringVector∗

  Return loaded module names in sorted order.
- func dr_copy_parse_diags(dst: DiagCollector∗, pr: ParseResult∗)

  Copy parse diagnostics into the driver collector.
- func dr_emit_error(diags: DiagCollector∗, code: string, message: string, filename: string)

  Emit a driver diagnostic without a precise source span.
- func dr_sv_has(items: StringVector∗, module_name: string) → bool

  Check whether a string vector contains a module name.
- func dr_mark_loaded(loaded: StringVector∗, module_name: string)

Mark a module as loaded if it is not already recorded.

- func dr_parse_source(path: string, source: string, cfg: LogConfig∗) → ParseResult∗

  Lex and parse one module source file while emitting Stage 1-style debug logs.

- func dr_visit_closure(state: DriverState∗, search_paths: SourceSearchPaths∗, module_name: string, visited: StringVector∗, cfg: LogConfig∗)

  Revisit the loaded module closure to mirror Stage 1 cache-hit logging.

- func dr_load_module(state: DriverState∗, search_paths: SourceSearchPaths∗, loading_chain: string, module_name: string, cfg: LogConfig∗) → bool

  Load one module and its import closure into the driver state.

- func driver_analyze_entry(search_paths: SourceSearchPaths∗, entry_module: string, cfg: LogConfig∗) → DriverState∗

  Load and parse an entry module and its imports.

8.60.1 Function Documentation

8.60.1.1 dr_copy_parse_diags()

```
func dr_copy_parse_diags(dst:  DiagCollector*, pr:  ParseResult*)
```

Copy parse diagnostics into the driver collector.

Parameters

dst	Destination diagnostic collector.
pr	Parse result providing diagnostics.

8.60.1.2 dr_emit_error()

```
func dr_emit_error(diags:  DiagCollector*, code:  string, message:  string, filename:  string)
```

Emit a driver diagnostic without a precise source span.

Parameters

diags	Diagnostic collector.
code	Diagnostic code.
message	Diagnostic message.
filename	Filename to attach when known.

8.60.1.3 dr_load_module()

```
func dr_load_module(state:  DriverState*, search_paths:  SourceSearchPaths*, loading_chain:
string, module_name:  string, cfg:  LogConfig*) → bool
```

Load one module and its import closure into the driver state.

Parameters

state	Driver state that caches loaded modules and diagnostics.
search_paths	Source search paths used to resolve module names.
loading_chain	Import chain encoded with `|name|` sentinels for cycle detection.
module_name	Module name to load.
cfg	Logging configuration.

Returns

`true` if the module was loaded successfully or was already cached.

8.60.1.4 dr_mark_loaded()

```
func dr_mark_loaded(loaded:  StringVector*, module_name:  string)
```

Mark a module as loaded if it is not already recorded.

Parameters

loaded	Vector of loaded module names.
module_name	Module name to record.

8.60.1.5 dr_parse_source()

```
func dr_parse_source(path:  string, source:  string, cfg:  LogConfig*) → ParseResult*
```

Lex and parse one module source file while emitting Stage 1-style debug logs.

Parameters

path	Source path being processed.
source	Source text to lex and parse.
cfg	Logging configuration.

Returns

Parse result for the module.

8.60.1.6 dr_sv_has()

```
func dr_sv_has(items:  StringVector*, module_name:  string) → bool
```

Check whether a string vector contains a module name.

Parameters

items	Vector to search.
module_name	Module name to look for.

Returns

`true` if `module_name` is present in `items`.

8.60.1.7 dr_unit_free()

```
func dr_unit_free(self: DriverUnit*)
```

Free a loaded driver unit and its parse result.

Parameters

self	Driver unit to free.

8.60.1.8 dr_units_free()

```
func dr_units_free(units: VectorBase*?)
```

Free an optional vector of driver units.

Parameters

units	Optional driver-unit vector to free.

8.60.1.9 dr_visit_closure()

```
func dr_visit_closure(state: DriverState*, search_paths: SourceSearchPaths*, module_name:
string, visited: StringVector*, cfg: LogConfig*)
```

Revisit the loaded module closure to mirror Stage 1 cache-hit logging.

Parameters

state	Driver state containing loaded modules.
search_paths	Source search paths used for lookups.
module_name	Current module name.
visited	Module names already traversed by this closure walk.
cfg	Logging configuration.

8.60.1.10 driver_analyze_entry()

```
func driver_analyze_entry(search_paths: SourceSearchPaths*, entry_module: string, cfg: LogConfig*)
→ DriverState*
```

Load and parse an entry module and its imports.

Parameters

search_paths	Source search paths used for module resolution.
entry_module	Entry module name.
cfg	Logging configuration.

Returns

Driver state containing parsed modules, caches, and diagnostics.

8.60.1.11 driver_diag_count()

```
func driver_diag_count(self: DriverState*) → int
```

Return the number of diagnostics collected by the driver.

Parameters

self	Driver state to inspect.

Returns

Total diagnostic count.

8.60.1.12 driver_diag_get()

```
func driver_diag_get(self: DriverState*, index: int) → Diagnostic*
```

Return a driver diagnostic by index.

Parameters

self	Driver state to inspect.
index	Zero-based diagnostic index.

Returns

Diagnostic at `index`.

8.60.1.13 driver_find_unit()

```
func driver_find_unit(self:  DriverState*, module_name:  string) → DriverUnit*?
```

Find a loaded driver unit by module name.

Parameters

self	Driver state to inspect.
module_name	Module name to search for.

Returns

Matching driver unit, or `null` if not loaded.

8.60.1.14 driver_has_errors()

```
func driver_has_errors(self:  DriverState*) → bool
```

Report whether driver analysis produced any errors.

Parameters

self	Driver state to inspect.

Returns

`true` if the driver's diagnostics contain an error.

8.60.1.15 driver_module_names_sorted()

```
func driver_module_names_sorted(self:  DriverState*) → StringVector*
```

Return loaded module names in sorted order.

Parameters

self	Driver state to inspect.

Returns

New vector containing sorted module names.

8.60.1.16 driver_state_free()

```
func driver_state_free(self:  DriverState*)
```

Free driver state and all owned parse/cache data.

Parameters

self	Driver state to free.

8.60.1.17 driver_unit_count()

```
func driver_unit_count(self:  DriverState*) → int
```

Return the number of loaded driver units.

Parameters

self	Driver state to inspect.

Returns

Loaded module count.

8.60.1.18 driver_unit_get()

```
func driver_unit_get(self:  DriverState*, index:  int) → DriverUnit*
```

Return a loaded driver unit by index.

Parameters

self	Driver state to inspect.
index	Zero-based unit index.

Returns

Driver unit at `index`.

8.61 compiler/stage2_l0/src/expr_types.l0 File Reference

Module: `expr_types`

```
import types;
import type_resolve;
import signatures;
import std.hashmap;
import std.string;
import sem_context;
import std.hashset;
import util.diag;
import std.text;
```

```
import ast;
import symbols;
import std.io;
import util.intset;
import util.strings;
import locals;
import std.vector;
import driver;
import name_resolver;
```

Classes

- struct ExpressionTypeChecker

  Represents the ExpressionTypeChecker structure.
- struct LoopFlowCapture

  Captured liveness states for loop-control exits from one loop body.
- struct LoopIterationFlow

  Result of checking one loop iteration for liveness flow.
- struct LoopFixedPointFlow

  Result of fixed-point liveness convergence for one loop.
- struct CleanupHeaderGuard

  Linked stack entry for guarded `with` header locals during cleanup-block checking.

Enumerations

- enum StmtFlowKind

  Describes whether a statement can reach the next statement.
- enum AssignCheckResult { ASSIGN_REJECT , ASSIGN_ACCEPT , ASSIGN_REPORTED }

  Result of checking assignability in a typed expression context.

Functions

- func etc_diag_error(self: ExpressionTypeChecker∗, code: string, message: string, filename: string, span: Span)

  Emit a type-checking error for a source span.
- func etc_diag_warn(self: ExpressionTypeChecker∗, code: string, message: string, filename: string, span: Span)

  Emit a type-checking warning for a source span.
- func etc_nested_symbol_path_full(expr: ExprNode∗) → string

  Format the full display form for one nested symbol path expression.
- func etc_nested_symbol_path_simple(expr: ExprNode∗) → string

  Format the direct symbol suggestion for one nested symbol path expression.
- func etc_nested_symbol_path_message(expr: ExprNode∗) → string

  Build the `TYP-0158` message for one nested symbol path expression.
- func etc_key(module_name: string, id: int) → string

  Build the storage key used for per-expression analysis tables.
- func etc_alive_scope_stack_free(stack: VectorBase∗?)

  Free a stack of liveness maps.
- func etc_alive_scope_clone(src: StringIntMap∗) → StringIntMap∗

Clone one liveness map.

- func etc_alive_scope_stack_clone(stack: VectorBase∗) → VectorBase∗

  Clone a stack of liveness maps.

- func etc_alive_scope_stack_meet(left: VectorBase∗, right: VectorBase∗) → VectorBase∗

  Meet two definite-liveness stacks.

- func etc_alive_scope_stack_equal(left: VectorBase∗, right: VectorBase∗) → bool

  Return whether two definite-liveness stacks contain the same state.

- func etc_alive_scope_stack_restore(self: ExpressionTypeChecker∗, state: VectorBase∗)

  Replace the checker's liveness stack with a deep copy of one state.

- func etc_alive_state_vec_free(states: VectorBase∗?)

  Free a vector of liveness-stack states.

- func etc_alive_state_vec_push_clone(states: VectorBase∗, state: VectorBase∗)

  Push a deep copy of one liveness state.

- func etc_alive_state_vec_clone(states: VectorBase∗) → VectorBase∗

  Clone a vector of liveness-stack states.

- func etc_alive_scope_stack_meet_many(base: VectorBase∗, states: VectorBase∗) → VectorBase∗

  Meet one base state with every state in a vector.

- func etc_loop_flow_capture_free(capture: LoopFlowCapture∗?)

  Free one loop-flow capture and its owned state vectors.

- func etc_capture_break_state(self: ExpressionTypeChecker∗)

  Append one break state to the active loop capture.

- func etc_capture_continue_state(self: ExpressionTypeChecker∗)

  Append one continue state to the active loop capture.

- func etc_check_update_from_backedge_states(self: ExpressionTypeChecker∗, update: StmtId, states: VectorBase∗) → VectorBase∗

  Check a `for` update from all reachable body backedges.

- func etc_check_loop_iteration(self: ExpressionTypeChecker∗, body: StmtId, update: StmtId) → LoopIterationFlow∗

  Check one loop iteration and collect reachable backedge/exit states.

- func etc_loop_liveness_fixed_point(self: ExpressionTypeChecker∗, pre_loop: VectorBase∗, cond: ExprId, body: StmtId, update: StmtId) → LoopFixedPointFlow∗

  Converge one loop's head state, then diagnose uses under the settled state.

- func etc_alive_scope_current(self: ExpressionTypeChecker∗) → StringIntMap∗?

  Return the innermost liveness map.

- func etc_alive_scope_push(self: ExpressionTypeChecker∗, scope: Scope∗?)

  Push one new liveness map for a scope.

- func etc_alive_scope_pop(self: ExpressionTypeChecker∗)

  Pop the innermost liveness map.

- func etc_decl_scope_current(self: ExpressionTypeChecker∗) → StringIntMap∗?

  Return the innermost declaration-tracking map.

- func etc_decl_scope_push(self: ExpressionTypeChecker∗, scope: Scope∗)

  Push one declaration-tracking map for a scope.

- func etc_decl_scope_pop(self: ExpressionTypeChecker∗)

  Pop the innermost declaration-tracking map.

- func etc_scope_chain_push(self: ExpressionTypeChecker∗, target: Scope∗) → int

  Activate every missing lexical scope from the tracked scope to `target`.

- func etc_decl_outer_has(self: ExpressionTypeChecker∗, name: string) → bool

  Check whether an outer declaration-tracking scope contains a name.

- func etc_check_local_declaration(self: ExpressionTypeChecker∗, name: string, span: Span)

  Validate a local declaration name before binding it in the current scope.

- func etc_alive_lookup(self: ExpressionTypeChecker∗, name: string) → int?

Look up definite liveness for a visible local.

- func etc_alive_set(self: ExpressionTypeChecker∗, name: string, alive: bool)

  Update definite liveness for the nearest tracked local binding.
- func etc_alive_declare(self: ExpressionTypeChecker∗, name: string)

  Declare the current-scope local as definitely alive.
- func etc_alive_revive_assignment_target(self: ExpressionTypeChecker∗, expr_id: ExprId)

  Mark a simple local assignment target alive again.
- func etc_set_expr_type(self: ExpressionTypeChecker∗, expr_id: ExprId, ty: Type∗)

  Record the inferred type for an expression.
- func etc_pattern_var_key(self: ExpressionTypeChecker∗, pattern_id: PatternId, binding_slot: int) → string

  Build the type-map key for one exact pattern-bound variable.
- func etc_set_pattern_var_type(self: ExpressionTypeChecker∗, pattern_id: PatternId, binding_slot: int, ty: Type∗)

  Record the inferred type for one exact pattern-bound variable.
- func etc_get_pattern_var_type(self: ExpressionTypeChecker∗, pattern_id: PatternId, binding_slot: int) → Type∗?

  Return a cloned stored type for one exact pattern-bound variable.
- func etc_set_var_res(self: ExpressionTypeChecker∗, expr_id: ExprId, res: VarRefResolution)

  Record whether a variable reference resolved locally or at module scope.
- func etc_set_intrinsic_target(self: ExpressionTypeChecker∗, expr_id: ExprId, ty: Type∗)

  Record the target type of an intrinsic expression.
- func etc_cleanup_header_guard_stack_free(stack: CleanupHeaderGuard∗?)

  Free a temporary cleanup-header guard stack.
- func etc_is_guarded_cleanup_header_local(self: ExpressionTypeChecker∗, name: string) → bool

  Return whether the current cleanup-block guard stack marks the visible local as maybe uninitialized.
- func etc_expr_contains_try(arena: ExprArena∗, expr_id: ExprId) → bool

  Return whether an expression contains postfix `?`.
- func etc_stmt_contains_try(unit: DriverUnit∗, stmt_id: StmtId) → bool

  Return whether a statement contains postfix `?`.
- func etc_lookup_visible_local_symbol(self: ExpressionTypeChecker∗, name: string) → LocalSymbol∗?

  Resolve the type of a local binding visible in the current scope.
- func etc_local_symbol_type(self: ExpressionTypeChecker∗, l: LocalSymbol∗) → Type∗?

  Resolve the type represented by one exact local symbol.
- func etc_lookup_local(self: ExpressionTypeChecker∗, name: string) → Type∗?

  Resolve the type of a local binding visible at the current statement position.
- func etc_lookup_symbol(self: ExpressionTypeChecker∗, name: string, module_path: StringVector∗?, name_qualifier: StringVector∗?) → Symbol∗?

  Resolve a visible symbol from the current expression-checking environment.
- func etc_is_void(ty: Type∗?) → bool

  Check whether a type is `void`.
- func etc_symbol_as_type(self: ExpressionTypeChecker∗, sym: Symbol∗) → Type∗?

  Resolve a visible symbol as a type usable by `sizeof`.
- func etc_diag_type_ref_failure(self: ExpressionTypeChecker∗, tref: TypeRef∗, fallback_code: string, span: Span)

  Emit the most specific diagnostic for a failed type-reference resolution.
- func etc_try_resolve_type_name(self: ExpressionTypeChecker∗, expr: ExprNode∗) → Type∗?

  Try to resolve one identifier expression as a type name for intrinsics.
- func etc_infer_sizeof_intrinsic(self: ExpressionTypeChecker∗, expr_id: ExprId, expr: ExprNode∗) → Type∗

  Infer the type of one `sizeof` intrinsic call.
- func etc_infer_ord_intrinsic(self: ExpressionTypeChecker∗, expr: ExprNode∗) → Type∗

  Infer the type of one `ord` intrinsic call.

- func etc_bind_pattern_types(self: ExpressionTypeChecker∗, match_ty: Type∗, pattern_id: PatternId)

  Bind enum payload types for variables introduced by a match pattern.
- func etc_find_enum_variant_info(info: EnumInfo∗, name: string) → EnumVariantInfo∗?

  Find one enum-variant metadata entry by name.
- func etc_pattern_symbol_name(pattern: PatternNode∗) → string

  Format one pattern symbol name for diagnostics.
- func etc_nested_pattern_path_message(pattern: PatternNode∗) → string

  Build the `TYP-0158` message for one nested match-pattern path.
- func etc_check_match_pattern(self: ExpressionTypeChecker∗, match_ty: Type∗, pattern_id: PatternId) → EnumVariantInfo∗?

  Check whether one `match` arm pattern is valid for the matched enum type.
- func etc_format_missing_match_variants(info: EnumInfo∗, covered: StringSet∗) → string

  Format the missing enum variants for a non-exhaustive `match`.
- func etc_match_variants_fully_covered(info: EnumInfo∗, covered: StringSet∗) → bool

  Return whether every declared enum variant is present in validated coverage.
- func etc_check_match_exhaustiveness(self: ExpressionTypeChecker∗, match_ty: Type∗, covered: StringSet∗, wildcard_present: bool, wildcard_is_unreachable: bool, span: Span) → bool

  Check whether an enum `match` covers all variants.
- func etc_match_wildcard_is_unreachable(self: ExpressionTypeChecker∗, match_ty: Type∗, covered: StringSet∗) → bool

  Return whether wildcard match arms are unreachable because variants cover the enum.
- func etc_is_case_type(ty: Type∗?) → bool

  Return whether a type is valid as a `case` scrutinee.
- func etc_case_literal_key(expr: ExprNode∗) → string?

  Build a stable duplicate-detection key for a `case` literal.
- func etc_check_dead_liveness_stmt(self: ExpressionTypeChecker∗, stmt_id: StmtId) → StmtFlowKind

  Type-check an unreachable branch while suppressing only liveness errors and loop exits.
- func etc_check_stmt(self: ExpressionTypeChecker∗, stmt_id: StmtId) → StmtFlowKind

  Type-check a statement and its nested statements.
- func etc_is_int(ty: Type∗?) → bool

  Check whether a type behaves like an integer in arithmetic.
- func etc_is_bool(ty: Type∗?) → bool

  Check whether a type is `bool`.
- func etc_is_string(ty: Type∗?) → bool

  Check whether a type is `string`.
- func etc_types_equal(a: Type∗?, b: Type∗?) → bool

  Compare two types for equality in the expression checker.
- func etc_type_free_opt(ty: Type∗?)

  Free an optional owned type.
- func etc_can_assign(self: ExpressionTypeChecker∗, dst: Type∗, src: Type∗) → bool

  Check whether a value of one type can be assigned to another.
- func etc_check_assign_expr(self: ExpressionTypeChecker∗, dst: Type∗, src: Type∗, expr_id: ExprId, filename: string, span: Span) → AssignCheckResult

  Check whether an expression is assignable in a typed context.
- func etc_check_initializer_expr(self: ExpressionTypeChecker∗, expected: Type∗, expr_id: ExprId, name: string, filename: string, span: Span) → bool

  Check an initializer expression against its contextual binding type.
- func etc_can_cast(self: ExpressionTypeChecker∗, dst: Type∗, src: Type∗) → bool

  Check whether a value can be cast explicitly using Stage 1 parity rules.
- func etc_cast_const_int(self: ExpressionTypeChecker∗, expr_id: ExprId) → int?

  Extract a compile-time integer value from a cast operand when available.

- func etc_cast_is_const_null(self: ExpressionTypeChecker∗, expr_id: ExprId) → bool

  Check whether a cast operand is provably null at compile time.
- func etc_infer_unary(self: ExpressionTypeChecker∗, expr_id: ExprId, expr: ExprNode∗) → Type∗?

  Infers the type for a unary.
- func etc_infer_binary(self: ExpressionTypeChecker∗, expr_id: ExprId, expr: ExprNode∗) → Type∗?

  Infers the type for a binary.
- func etc_infer_call(self: ExpressionTypeChecker∗, expr_id: ExprId, expr: ExprNode∗) → Type∗?

  Infers the type for a call.
- func etc_infer_field(self: ExpressionTypeChecker∗, expr_id: ExprId, expr: ExprNode∗) → Type∗?

  Infers the type for a field.
- func etc_infer_index(self: ExpressionTypeChecker∗, expr_id: ExprId, expr: ExprNode∗) → Type∗?

  Infers the type for a index.
- func etc_check_expr_liveness(self: ExpressionTypeChecker∗, expr_id: ExprId)

  Replay local-variable use checks without repeating expression semantics.
- func etc_infer_expr(self: ExpressionTypeChecker∗, expr_id: ExprId) → Type∗?

  Infers the type for a expr.
- func etc_check_function(self: ExpressionTypeChecker∗, env: FunctionEnv∗)

  Type-check one function body.
- func etc_check_top_level_initializers(self: ExpressionTypeChecker∗, root: Module∗)

  Type-check module-scope `let` initializer expressions.
- func expr_types_check(result: AnalysisResult∗)

  Run expression and statement type checking for the analysis result.

8.61.1 Enumeration Type Documentation

8.61.1.1 AssignCheckResult

```
enum AssignCheckResult
```

Result of checking assignability in a typed expression context.

Enumerator

ASSIGN_REJECT	
ASSIGN_ACCEPT	
ASSIGN_REPORTED	

8.61.1.2 StmtFlowKind

```
enum StmtFlowKind
```

Describes whether a statement can reach the next statement.

Enumerator

FLOW_FALLTHROUGH	
FLOW_RETURNS	
FLOW_STOPS	

8.61.2 Function Documentation

8.61.2.1 etc_alive_declare()

```
func etc_alive_declare(self:  ExpressionTypeChecker*, name:  string)
```

Declare the current-scope local as definitely alive.

Parameters

self	Expression checker state.
name	Local name.

8.61.2.2 etc_alive_lookup()

```
func etc_alive_lookup(self:  ExpressionTypeChecker*, name:  string) → int?
```

Look up definite liveness for a visible local.

Parameters

self	Expression checker state.
name	Local name.

Returns

`1` if alive, `0` if dropped, or `null` if not tracked.

8.61.2.3 etc_alive_revive_assignment_target()

```
func etc_alive_revive_assignment_target(self:  ExpressionTypeChecker*, expr_id:  ExprId)
```

Mark a simple local assignment target alive again.

Parameters

self	Expression checker state.
expr↩ _id	Potential local-variable lvalue expression.

8.61.2.4 etc_alive_scope_clone()

```
func etc_alive_scope_clone(src:  StringIntMap*) → StringIntMap*
```

Clone one liveness map.

Parameters

src	Source map.

Returns

Deep copy of `src`.

8.61.2.5 etc_alive_scope_current()

```
func etc_alive_scope_current(self:  ExpressionTypeChecker*) → StringIntMap*?
```

Return the innermost liveness map.

Parameters

self	Expression checker state.

Returns

Current `StringIntMap*`, or `null`.

8.61.2.6 etc_alive_scope_pop()

```
func etc_alive_scope_pop(self:  ExpressionTypeChecker*)
```

Pop the innermost liveness map.

Parameters

self	Expression checker state.

8.61.2.7 etc_alive_scope_push()

```
func etc_alive_scope_push(self:  ExpressionTypeChecker*, scope:  Scope*?)
```

Push one new liveness map for a scope.

Parameters and pattern variables start alive immediately; local `let` bindings become alive when their declaration is reached.

Parameters

self	Expression checker state.
scope	Scope whose immediate bindings should seed the map.

8.61.2.8 etc_alive_scope_stack_clone()

```
func etc_alive_scope_stack_clone(stack:  VectorBase*) → VectorBase*
```

Clone a stack of liveness maps.

Parameters

stack	Source vector of `StringIntMap*`.

Returns

Deep copy of `stack`.

8.61.2.9 etc_alive_scope_stack_equal()

```
func etc_alive_scope_stack_equal(left:  VectorBase*, right:  VectorBase*) → bool
```

Return whether two definite-liveness stacks contain the same state.

8.61.2.10 etc_alive_scope_stack_free()

```
func etc_alive_scope_stack_free(stack:  VectorBase*?)
```

Free a stack of liveness maps.

Parameters

stack	Optional vector of `StringIntMap*`.

8.61.2.11 etc_alive_scope_stack_meet()

```
func etc_alive_scope_stack_meet(left:  VectorBase*, right:  VectorBase*) → VectorBase*
```

Meet two definite-liveness stacks.

A binding remains alive only when both incoming states keep it alive.

8.61.2.12 etc_alive_scope_stack_meet_many()

```
func etc_alive_scope_stack_meet_many(base:  VectorBase*, states:  VectorBase*) → VectorBase*
```

Meet one base state with every state in a vector.

8.61.2.13 etc_alive_scope_stack_restore()

```
func etc_alive_scope_stack_restore(self: ExpressionTypeChecker*, state: VectorBase*)
```

Replace the checker's liveness stack with a deep copy of one state.

8.61.2.14 etc_alive_set()

```
func etc_alive_set(self: ExpressionTypeChecker*, name: string, alive: bool)
```

Update definite liveness for the nearest tracked local binding.

Parameters

self	Expression checker state.
name	Local name.
alive	`true` when usable, `false` after `drop`.

8.61.2.15 etc_alive_state_vec_clone()

```
func etc_alive_state_vec_clone(states: VectorBase*) → VectorBase*
```

Clone a vector of liveness-stack states.

8.61.2.16 etc_alive_state_vec_free()

```
func etc_alive_state_vec_free(states: VectorBase*?)
```

Free a vector of liveness-stack states.

8.61.2.17 etc_alive_state_vec_push_clone()

```
func etc_alive_state_vec_push_clone(states: VectorBase*, state: VectorBase*)
```

Push a deep copy of one liveness state.

8.61.2.18 etc_bind_pattern_types()

```
func etc_bind_pattern_types(self: ExpressionTypeChecker*, match_ty: Type*, pattern_id: PatternId)
```

Bind enum payload types for variables introduced by a match pattern.

Parameters

self	Expression checker state.
match_ty	Type of the matched expression.
pattern↩	Pattern whose variables should be bound.
_id	

8.61.2.19 etc_can_assign()

```
func etc_can_assign(self: ExpressionTypeChecker*, dst: Type*, src: Type*) → bool
```

Check whether a value of one type can be assigned to another.

Parameters

self	Expression checker state.
dst	Destination type.
src	Source type.

Returns

`true` if assignment is allowed.

8.61.2.20 etc_can_cast()

```
func etc_can_cast(self: ExpressionTypeChecker*, dst: Type*, src: Type*) → bool
```

Check whether a value can be cast explicitly using Stage 1 parity rules.

Parameters

self	Expression checker state.
dst	Destination type.
src	Source type.

Returns

`true` if the explicit cast is allowed.

8.61.2.21 etc_capture_break_state()

```
func etc_capture_break_state(self: ExpressionTypeChecker*)
```

Append one break state to the active loop capture.

8.61.2.22 etc_capture_continue_state()

```
func etc_capture_continue_state(self: ExpressionTypeChecker*)
```

Append one continue state to the active loop capture.

8.61.2.23 etc_case_literal_key()

```
func etc_case_literal_key(expr: ExprNode*) → string?
```

Build a stable duplicate-detection key for a `case` literal.

Parameters

expr	Literal expression node.

Returns

A type-prefixed key, or `null` if the expression is unsupported.

8.61.2.24 etc_cast_const_int()

```
func etc_cast_const_int(self: ExpressionTypeChecker*, expr_id: ExprId) → int?
```

Extract a compile-time integer value from a cast operand when available.

Parameters

self	Expression checker state.
expr↩ _id	Operand expression id.

Returns

Integer value, or `null` when not a supported constant form.

8.61.2.25 etc_cast_is_const_null()

```
func etc_cast_is_const_null(self: ExpressionTypeChecker*, expr_id: ExprId) → bool
```

Check whether a cast operand is provably null at compile time.

Parameters

self	Expression checker state.
expr↩ _id	Operand expression id.

Returns

`true` if the operand is known-null.

8.61.2.26 etc_check_assign_expr()

```
func etc_check_assign_expr(self: ExpressionTypeChecker*, dst: Type*, src: Type*, expr_id:
ExprId, filename: string, span: Span) → AssignCheckResult
```

Check whether an expression is assignable in a typed context.

Integer literals may narrow from their natural `int` type to `byte` when their compile-time value fits the destination range.

Parameters

self	Expression checker state.
dst	Destination type.
src	Natural source type.
expr_id	Source expression id.
filename	Source filename.
span	Diagnostic span.

Returns

Typed-context assignability result.

8.61.2.27 etc_check_dead_liveness_stmt()

```
func etc_check_dead_liveness_stmt(self: ExpressionTypeChecker*, stmt_id: StmtId) → StmtFlowKind
```

Type-check an unreachable branch while suppressing only liveness errors and loop exits.

Parameters

self	Expression checker state.
stmt↩ _id	Statement to check.

Returns

Statement flow reported by the checked statement.

8.61.2.28 etc_check_expr_liveness()

```
func etc_check_expr_liveness(self: ExpressionTypeChecker*, expr_id: ExprId)
```

Replay local-variable use checks without repeating expression semantics.

8.61.2.29 etc_check_function()

```
func etc_check_function(self: ExpressionTypeChecker*, env: FunctionEnv*)
```

Type-check one function body.

Parameters

self	Expression checker state.
env	Function environment to check.

8.61.2.30 etc_check_initializer_expr()

```
func etc_check_initializer_expr(self: ExpressionTypeChecker*, expected: Type*, expr_id:
ExprId, name: string, filename: string, span: Span) → bool
```

Check an initializer expression against its contextual binding type.

Parameters

self	Expression checker state.
expected	Contextual binding type.
expr_id	Initializer expression id.
name	Binding name used in diagnostics.
filename	Source filename.
span	Declaration span for assignment diagnostics.

Returns

`true` when the initializer is assignable.

8.61.2.31 etc_check_local_declaration()

```
func etc_check_local_declaration(self: ExpressionTypeChecker*, name: string, span: Span)
```

Validate a local declaration name before binding it in the current scope.

Reports duplicate local errors and shadowing warnings for outer locals, enum variants, and imported symbols.

Parameters

self	Expression checker state.
name	Local name being declared.
span	Source span to report for diagnostics.

8.61.2.32 etc_check_loop_iteration()

```
func etc_check_loop_iteration(self: ExpressionTypeChecker*, body: StmtId, update: StmtId) →
LoopIterationFlow*
```

Check one loop iteration and collect reachable backedge/exit states.

8.61.2.33 etc_check_match_exhaustiveness()

```
func etc_check_match_exhaustiveness(self: ExpressionTypeChecker*, match_ty: Type*, covered:
StringSet*, wildcard_present: bool, wildcard_is_unreachable: bool, span: Span) → bool
```

Check whether an enum `match` covers all variants.

Parameters

self	Expression checker state.
match_ty	Type of the matched expression.
covered	Canonical names from successfully validated variant patterns.
wildcard_present	Whether the match contains a wildcard arm.
wildcard_is_unreachable	Whether prior validated variants already cover the enum.
span	Source span for diagnostics.

Returns

`true` if the match is exhaustive.

8.61.2.34 etc_check_match_pattern()

```
func etc_check_match_pattern(self: ExpressionTypeChecker*, match_ty: Type*, pattern_id:
PatternId) → EnumVariantInfo*?
```

Check whether one `match` arm pattern is valid for the matched enum type.

Parameters

self	Expression checker state.
match_ty	Enum type of the scrutinee.
pattern↩ _id	Pattern to validate.

Returns

Canonical metadata for the validated variant, or `null`.

8.61.2.35 etc_check_stmt()

```
func etc_check_stmt(self: ExpressionTypeChecker*, stmt_id: StmtId) → StmtFlowKind
```

Type-check a statement and its nested statements.

Parameters

self	Expression checker state.
stmt↩ _id	Statement id to check.

Returns

Whether the statement definitely returns.

8.61.2.36 etc_check_top_level_initializers()

```
func etc_check_top_level_initializers(self: ExpressionTypeChecker*, root: Module*)
```

Type-check module-scope `let` initializer expressions.

Signature resolution owns top-level binding type resolution. This pass checks every successfully resolved binding and records expression metadata required by backend lowering.

Parameters

self	Expression checker state.
root	Module whose top-level declarations should be checked.

8.61.2.37 etc_check_update_from_backedge_states()

```
func etc_check_update_from_backedge_states(self: ExpressionTypeChecker*, update: StmtId,
states: VectorBase*) → VectorBase*
```

Check a `for` update from all reachable body backedges.

Consumes `states` and returns a new vector of post-update backedge states.

8.61.2.38 etc_cleanup_header_guard_stack_free()

```
func etc_cleanup_header_guard_stack_free(stack: CleanupHeaderGuard*?)
```

Free a temporary cleanup-header guard stack.

Parameters

stack	Linked cleanup-header guard entries to free.

8.61.2.39 etc_decl_outer_has()

```
func etc_decl_outer_has(self: ExpressionTypeChecker*, name: string) → bool
```

Check whether an outer declaration-tracking scope contains a name.

The current innermost scope is skipped because duplicate checks inspect it separately.

Parameters

self	Expression checker state.
name	Local name to look up.

Returns

`true` when an outer scope already declares `name`.

8.61.2.40 etc_decl_scope_current()

```
func etc_decl_scope_current(self:  ExpressionTypeChecker*) → StringIntMap*?
```

Return the innermost declaration-tracking map.

Parameters

self	Expression checker state.

Returns

Current declaration map, or `null` when no scope is active.

8.61.2.41 etc_decl_scope_pop()

```
func etc_decl_scope_pop(self:  ExpressionTypeChecker*)
```

Pop the innermost declaration-tracking map.

Parameters

self	Expression checker state.

8.61.2.42 etc_decl_scope_push()

```
func etc_decl_scope_push(self:  ExpressionTypeChecker*, scope:  Scope*)
```

Push one declaration-tracking map for a scope.

Parameters and pattern variables are predeclared so later local checks can reject duplicate names and warn about shadowing.

Parameters

self	Expression checker state.
scope	Scope whose immediate non-local bindings should seed the map.

8.61.2.43 etc_diag_error()

```
func etc_diag_error(self:  ExpressionTypeChecker*, code:  string, message:  string, filename:
string, span:  Span)
```

Emit a type-checking error for a source span.

Parameters

self	Expression checker collecting diagnostics.
code	Diagnostic code.
message	Diagnostic message.
filename	Source filename.
span	Source span to highlight.

8.61.2.44 etc_diag_type_ref_failure()

```
func etc_diag_type_ref_failure(self: ExpressionTypeChecker*, tref: TypeRef*, fallback_code:
string, span: Span)
```

Emit the most specific diagnostic for a failed type-reference resolution.

Special cases preserve shared diagnostic codes for nullable `void`, ambiguous names, unresolved aliases, and value symbols used as types. Other failures use `fallback_code` so callers can keep context-specific unknown-type codes.

Parameters

self	Expression checker state.
tref	Type reference that failed to resolve.
fallback_code	Diagnostic code for generic unknown-type failures.
span	Source span to report for diagnostics.

8.61.2.45 etc_diag_warn()

```
func etc_diag_warn(self: ExpressionTypeChecker*, code: string, message: string, filename:
string, span: Span)
```

Emit a type-checking warning for a source span.

Parameters

self	Expression checker collecting diagnostics.
code	Diagnostic code.
message	Diagnostic message.
filename	Source filename.
span	Source span to highlight.

8.61.2.46 etc_expr_contains_try()

```
func etc_expr_contains_try(arena: ExprArena*, expr_id: ExprId) → bool
```

Return whether an expression contains postfix `?`.

Parameters

arena	Expression arena to inspect.
expr↩ _id	Expression id to walk.

Returns

`true` when the expression tree contains `EX_TRY`.

8.61.2.47 etc_find_enum_variant_info()

```
func etc_find_enum_variant_info(info:  EnumInfo*, name:  string) → EnumVariantInfo*?
```

Find one enum-variant metadata entry by name.

Parameters

info	Enum metadata to inspect.
name	Variant name to find.

Returns

Matching variant metadata, or `null`.

8.61.2.48 etc_format_missing_match_variants()

```
func etc_format_missing_match_variants(info:  EnumInfo*, covered:  StringSet*) → string
```

Format the missing enum variants for a non-exhaustive `match`.

Parameters

info	Enum metadata for the matched type.
covered	Covered variant-name set.

Returns

Comma-separated missing variant names in enum declaration order.

8.61.2.49 etc_get_pattern_var_type()

```
func etc_get_pattern_var_type(self:  ExpressionTypeChecker*, pattern_id:  PatternId, binding_slot:
int) → Type*?
```

Return a cloned stored type for one exact pattern-bound variable.

Parameters

self	Expression checker state.
pattern_id	Owning pattern id.
binding_slot	Binder position within the pattern.

Returns

Cloned stored type, or `null` if absent.

8.61.2.50 etc_infer_binary()

```
func etc_infer_binary(self: ExpressionTypeChecker*, expr_id: ExprId, expr: ExprNode*) →
Type*?
```

Infers the type for a binary.

Parameters

self	Expression checker state.
expr↩ _id	Binary expression id.
expr	Binary expression node.

Returns

The inferred or resolved type, or null on error.

8.61.2.51 etc_infer_call()

```
func etc_infer_call(self: ExpressionTypeChecker*, expr_id: ExprId, expr: ExprNode*) → Type*?
```

Infers the type for a call.

Parameters

self	Expression checker state.
expr↩ _id	Call expression id.
expr	Call expression node.

Returns

The inferred or resolved type, or null on error.

8.61.2.52 etc_infer_expr()

```
func etc_infer_expr(self: ExpressionTypeChecker*, expr_id: ExprId) → Type*?
```

Infers the type for a expr.

Parameters

self	Expression checker state.
expr↩ _id	Expression id to inspect.

Returns

The inferred or resolved type, or null on error.

8.61.2.53 etc_infer_field()

```
func etc_infer_field(self: ExpressionTypeChecker*, expr_id: ExprId, expr: ExprNode*) →
Type*?
```

Infers the type for a field.

Parameters

self	Expression checker state.
expr↩ _id	Field-access expression id.
expr	Field-access expression node.

Returns

The inferred or resolved type, or null on error.

8.61.2.54 etc_infer_index()

```
func etc_infer_index(self: ExpressionTypeChecker*, expr_id: ExprId, expr: ExprNode*) →
Type*?
```

Infers the type for a index.

Parameters

self	Expression checker state.
expr↩ _id	Index expression id.
expr	Index expression node.

Returns

The inferred or resolved type, or null on error.

8.61.2.55 etc_infer_ord_intrinsic()

```
func etc_infer_ord_intrinsic(self: ExpressionTypeChecker*, expr: ExprNode*) → Type*
```

Infer the type of one `ord` intrinsic call.

Parameters

self	Expression checker state.
expr	Call expression node.

Returns

`int`, or `int` after reporting an error.

8.61.2.56 etc_infer_sizeof_intrinsic()

```
func etc_infer_sizeof_intrinsic(self: ExpressionTypeChecker*, expr_id: ExprId, expr: ExprNode*)
→ Type*
```

Infer the type of one `sizeof` intrinsic call.

Parameters

self	Expression checker state.
expr↩ _id	Call expression id.
expr	Call expression node.

Returns

`int`, with the target type recorded for backend use.

8.61.2.57 etc_infer_unary()

```
func etc_infer_unary(self: ExpressionTypeChecker*, expr_id: ExprId, expr: ExprNode*) →
Type*?
```

Infers the type for a unary.

Parameters

self	Expression checker state.
expr↩ _id	Unary expression id.
expr	Unary expression node.

Returns

The inferred or resolved type, or null on error.

8.61.2.58 etc_is_bool()

```
func etc_is_bool(ty:  Type*?)  → bool
```

Check whether a type is `bool`.

Parameters

ty	Type to inspect.

Returns

`true` if `ty` is `bool`.

8.61.2.59 etc_is_case_type()

```
func etc_is_case_type(ty:  Type*?)  → bool
```

Return whether a type is valid as a `case` scrutinee.

Parameters

ty	Type to inspect.

Returns

`true` for `int`, `byte`, `bool`, and `string`.

8.61.2.60 etc_is_guarded_cleanup_header_local()

```
func etc_is_guarded_cleanup_header_local(self:  ExpressionTypeChecker*, name:  string) → bool
```

Return whether the current cleanup-block guard stack marks the visible local as maybe uninitialized.

Parameters

self	Expression checker state.
name	Variable name to inspect.

Returns

`true` when the visible local binding matches a guarded header declaration.

8.61.2.61 etc_is_int()

```
func etc_is_int(ty:  Type*?)  → bool
```

Check whether a type behaves like an integer in arithmetic.

Parameters

ty	Type to inspect.

Returns

`true` for `int` and `byte`.

8.61.2.62 etc_is_string()

```
func etc_is_string(ty:  Type*?)  → bool
```

Check whether a type is `string`.

Parameters

ty	Type to inspect.

Returns

`true` if `ty` is `string`.

8.61.2.63 etc_is_void()

```
func etc_is_void(ty:  Type*?)  → bool
```

Check whether a type is `void`.

Parameters

ty	Type to inspect.

Returns

`true` if `ty` is `void`.

8.61.2.64 etc_key()

```
func etc_key(module_name:  string, id:  int) → string
```

Build the storage key used for per-expression analysis tables.

Parameters

module_name	Owning module name.
id	Expression id.

Returns

Key in `module::id` form.

8.61.2.65 etc_local_symbol_type()

```
func etc_local_symbol_type(self: ExpressionTypeChecker*, l: LocalSymbol*) → Type*?
```

Resolve the type represented by one exact local symbol.

8.61.2.66 etc_lookup_local()

```
func etc_lookup_local(self: ExpressionTypeChecker*, name: string) → Type*?
```

Resolve the type of a local binding visible at the current statement position.

Parameters

self	Expression checker state.
name	Local name to look up.

Returns

Resolved type, or `null` if the name is not a visible local.

8.61.2.67 etc_lookup_symbol()

```
func etc_lookup_symbol(self: ExpressionTypeChecker*, name: string, module_path: StringVector*?,
name_qualifier: StringVector*?) → Symbol*?
```

Resolve a visible symbol from the current expression-checking environment.

Parameters

self	Expression checker state.
name	Unqualified symbol name.
module_path	Optional parsed module path for cross-module lookups.
name_qualifier	Optional enum-variant qualifier.

Returns

Resolved symbol, or `null` if lookup fails.

8.61.2.68 etc_lookup_visible_local_symbol()

```
func etc_lookup_visible_local_symbol(self: ExpressionTypeChecker*, name: string) → LocalSymbol*?
```

Resolve the type of a local binding visible in the current scope.

Parameters

self	Expression checker state.
name	Local name to look up.

Returns

Resolved type, or `null` if the name is not a visible local.

8.61.2.69 etc_loop_flow_capture_free()

```
func etc_loop_flow_capture_free(capture: LoopFlowCapture*?)
```

Free one loop-flow capture and its owned state vectors.

8.61.2.70 etc_loop_liveness_fixed_point()

```
func etc_loop_liveness_fixed_point(self: ExpressionTypeChecker*, pre_loop: VectorBase*, cond:
ExprId, body: StmtId, update: StmtId) → LoopFixedPointFlow*
```

Converge one loop's head state, then diagnose uses under the settled state.

`body` is checked in the inner loop context. `update` is checked after that context is removed so header loop control continues to target an outer loop.

8.61.2.71 etc_match_variants_fully_covered()

```
func etc_match_variants_fully_covered(info: EnumInfo*, covered: StringSet*) → bool
```

Return whether every declared enum variant is present in validated coverage.

8.61.2.72 etc_match_wildcard_is_unreachable()

```
func etc_match_wildcard_is_unreachable(self: ExpressionTypeChecker*, match_ty: Type*, covered:
StringSet*) → bool
```

Return whether wildcard match arms are unreachable because variants cover the enum.

8.61.2.73 etc_nested_pattern_path_message()

```
func etc_nested_pattern_path_message(pattern: PatternNode*) → string
```

Build the `TYP-0158` message for one nested match-pattern path.

Parameters

pattern	Variant pattern carrying module and qualifier segments.

Returns

Diagnostic text matching expression nested-path diagnostics.

8.61.2.74 etc_nested_symbol_path_full()

```
func etc_nested_symbol_path_full(expr: ExprNode*) → string
```

Format the full display form for one nested symbol path expression.

Parameters

expr	Variable-reference expression carrying module and qualifier segments.

Returns

Full nested path such as `color:: Color:: Red`.

8.61.2.75 etc_nested_symbol_path_message()

```
func etc_nested_symbol_path_message(expr: ExprNode*) → string
```

Build the `TYP-0158` message for one nested symbol path expression.

Parameters

expr	Variable-reference expression carrying module and qualifier segments.

Returns

Diagnostic text matching the shared nested-path wording.

8.61.2.76 etc_nested_symbol_path_simple()

```
func etc_nested_symbol_path_simple(expr: ExprNode*) → string
```

Format the direct symbol suggestion for one nested symbol path expression.

Parameters

expr	Variable-reference expression carrying module and qualifier segments.

Returns

Simplified path such as `color::  Red`.

8.61.2.77 etc_pattern_symbol_name()

```
func etc_pattern_symbol_name(pattern:  PatternNode*) → string
```

Format one pattern symbol name for diagnostics.

Parameters

pattern	Pattern node to describe.

Returns

Unqualified or module-qualified pattern name.

8.61.2.78 etc_pattern_var_key()

```
func etc_pattern_var_key(self:  ExpressionTypeChecker*, pattern_id:  PatternId, binding_slot:
int) → string
```

Build the type-map key for one exact pattern-bound variable.

Parameters

self	Expression checker state.
pattern_id	Owning pattern id.
binding_slot	Binder position within the pattern.

Returns

Stable binding-specific type-map key.

8.61.2.79 etc_scope_chain_push()

```
func etc_scope_chain_push(self:  ExpressionTypeChecker*, target:  Scope*) → int
```

Activate every missing lexical scope from the tracked scope to `target`.

Synthetic single-statement scopes can sit between the currently active scope and a mapped loop or `with` header scope, so each ancestor must have its own declaration and liveness state.

Returns

Number of scope states pushed.

8.61.2.80 etc_set_expr_type()

```
func etc_set_expr_type(self: ExpressionTypeChecker*, expr_id: ExprId, ty: Type*)
```

Record the inferred type for an expression.

Parameters

self	Expression checker state.
expr↩ _id	Expression id.
ty	Type to store.

8.61.2.81 etc_set_intrinsic_target()

```
func etc_set_intrinsic_target(self: ExpressionTypeChecker*, expr_id: ExprId, ty: Type*)
```

Record the target type of an intrinsic expression.

Parameters

self	Expression checker state.
expr↩ _id	Expression id.
ty	Target type to store.

8.61.2.82 etc_set_pattern_var_type()

```
func etc_set_pattern_var_type(self: ExpressionTypeChecker*, pattern_id: PatternId, binding_slot:
int, ty: Type*)
```

Record the inferred type for one exact pattern-bound variable.

Parameters

self	Expression checker state.
pattern_id	Owning pattern id.
binding_slot	Binder position within the pattern.
ty	Type to store.

8.61.2.83 etc_set_var_res()

```
func etc_set_var_res(self: ExpressionTypeChecker*, expr_id: ExprId, res: VarRefResolution)
```

Record whether a variable reference resolved locally or at module scope.

Parameters

self	Expression checker state.
expr↩ _id	Expression id.
res	Resolution kind to store.

8.61.2.84 etc_stmt_contains_try()

```
func etc_stmt_contains_try(unit:  DriverUnit*, stmt_id:  StmtId) → bool
```

Return whether a statement contains postfix `?`.

Parameters

unit	Driver unit owning the statement.
stmt↩ _id	Statement id to inspect.

Returns

`true` when the statement contains `EX_TRY`.

8.61.2.85 etc_symbol_as_type()

```
func etc_symbol_as_type(self:  ExpressionTypeChecker*, sym:  Symbol*) → Type*?
```

Resolve a visible symbol as a type usable by `sizeof`.

Parameters

self	Expression checker state.
sym	Symbol to interpret as a type.

Returns

Resolved type, or `null` when the symbol is not a type name.

8.61.2.86 etc_try_resolve_type_name()

```
func etc_try_resolve_type_name(self:  ExpressionTypeChecker*, expr:  ExprNode*) → Type*?
```

Try to resolve one identifier expression as a type name for intrinsics.

Parameters

self	Expression checker state.
expr	Identifier expression to interpret as a type.

Returns

Resolved type, or `null` when the identifier should be treated as a value expression.

8.61.2.87 etc_type_free_opt()

```
func etc_type_free_opt(ty:  Type*?)
```

Free an optional owned type.

Parameters

ty	Optional type to free.

8.61.2.88 etc_types_equal()

```
func etc_types_equal(a:  Type*?, b:  Type*?)  → bool
```

Compare two types for equality in the expression checker.

Parameters

a	First type.
b	Second type.

Returns

`true` if the types are equal.

8.61.2.89 expr_types_check()

```
func expr_types_check(result:  AnalysisResult*)
```

Run expression and statement type checking for the analysis result.

Resolved top-level initializers remain safe to check after an earlier semantic error. Function bodies are checked only when this pass starts with a clean diagnostic state.

Parameters

result	Analysis result to populate with expression types and diagnostics.

8.62 compiler/stage2_l0/src/l0c.l0 File Reference

Module: `l0c`

```
import l0c_lib;
import std.vector;
import std.system;
```

Functions

- func main()

  Run the Stage 2 `l0c` entry point with process arguments.

8.62.1 Function Documentation

8.62.1.1 main()

```
func main()
```

Run the Stage 2 `l0c` entry point with process arguments.

8.63 compiler/stage2_l0/src/l0c_lib.l0 File Reference

Module: `l0c_lib`

```
import signatures;
import util.log;
import lexer;
import tokens;
import diag_print;
import std.hashmap;
import name_resolver;
import std.io;
import build_driver;
import driver;
import std.text;
import types;
import codegen_options;
import analysis;
import std.vector;
import ast_printer;
import std.fs;
import std.string;
import util.diag;
import symbols;
import cli_args;
import build_info;
import ast;
import sem_context;
import source_paths;
import util.strings;
import backend;
```

Classes

- struct PreparedInput

  Represents the PreparedInput structure.

Functions

- func prepared_input_free(self: PreparedInput∗)

  Free a prepared CLI input bundle.
- func prepared_input_free_opt(self: PreparedInput∗?)

  Free a nullable prepared input bundle when present.
- func l0c_copy_collector(dst: DiagCollector∗, src: DiagCollector∗)

  Copy every diagnostic from one collector into another.
- func l0c_log_effective_roots(cfg: LogConfig∗, sys_roots: StringVector∗, project_roots: StringVector∗)

  Log the effective system and project roots for one CLI invocation.
- func l0c_prepare_input(opts: CliOptions∗, diags: DiagCollector∗, cfg: LogConfig∗) → PreparedInput∗?

  Normalize CLI input into an entry module and search paths.
- func l0c_print_token(path: string, tok: Token)

  Print one token with source location metadata.
- func l0c_print_module_token_header(module_name: string, path: string)

  Print a header before dumping tokens for a module.
- func l0c_dump_tokens_for_path(path: string, include_eof: bool, diags: DiagCollector∗) → int

  Lex a source file and print its tokens.
- func l0c_cmd_check(opts: CliOptions∗, cfg: LogConfig∗) → int

  Execute Stage 2 `--check` mode.
- func l0c_cmd_tok(opts: CliOptions∗, cfg: LogConfig∗) → int

  Execute Stage 2 `--tok` mode.
- func l0c_cmd_gen(opts: CliOptions∗, cfg: LogConfig∗) → int

  Execute Stage 2 `--gen` mode.
- func l0c_cmd_sym(opts: CliOptions∗, cfg: LogConfig∗) → int

  Execute Stage 2 `--sym` mode.
- func l0c_cmd_type(opts: CliOptions∗, cfg: LogConfig∗) → int

  Execute Stage 2 `--type` mode.
- func l0c_append_c_option_words(dst: StringVector∗, raw: string?)

  Append whitespace-delimited C option words from one source string.
- func l0c_merge_c_option_words(env_opts: string?, cli_opts: string?) → StringVector∗

  Merge C option words from environment and CLI with CLI-last precedence.
- func l0c_collect_c_option_words(opts: CliOptions∗) → StringVector∗

  Resolve merged C option words from `L0_CFLAGS` and parsed CLI options.
- func l0c_cmd_ast(opts: CliOptions∗, cfg: LogConfig∗) → int

  Execute Stage 2 `--ast` mode.
- func l0c_cmd_nyi(mode: CliMode, cfg: LogConfig∗) → int

  Report that a requested CLI mode is not implemented in Stage 2 yet.
- func run_with_argv(argv: StringVector∗) → int

  Parse CLI arguments and dispatch the requested command.

8.63.1 Function Documentation

8.63.1.1 l0c_append_c_option_words()

```
func l0c_append_c_option_words(dst:  StringVector*, raw:  string?)
```

Append whitespace-delimited C option words from one source string.

Parameters

dst	Destination vector for parsed option words.
raw	Raw options string, or `null`.

8.63.1.2 l0c_cmd_ast()

```
func l0c_cmd_ast(opts: CliOptions*, cfg: LogConfig*) → int
```

Execute Stage 2 `--ast` mode.

Without `--all-modules`, prints the AST for the entry module only. With `--all-modules`, prints all modules in the compilation unit in sorted order, separated by `=== Module <name> ===` headers.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.

Returns

Process-style status code.

8.63.1.3 l0c_cmd_check()

```
func l0c_cmd_check(opts: CliOptions*, cfg: LogConfig*) → int
```

Execute Stage 2 `--check` mode.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.

Returns

Process-style status code.

8.63.1.4 l0c_cmd_gen()

```
func l0c_cmd_gen(opts: CliOptions*, cfg: LogConfig*) → int
```

Execute Stage 2 `--gen` mode.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.

Returns

Process-style status code.

8.63.1.5 l0c_cmd_nyi()

```
func l0c_cmd_nyi(mode: CliMode, cfg: LogConfig*) → int
```

Report that a requested CLI mode is not implemented in Stage 2 yet.

Parameters

mode	Requested CLI mode.
cfg	Logging configuration.

Returns

Process-style status code.

8.63.1.6 l0c_cmd_sym()

```
func l0c_cmd_sym(opts: CliOptions*, cfg: LogConfig*) → int
```

Execute Stage 2 `--sym` mode.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.

Returns

Process-style status code.

8.63.1.7 l0c_cmd_tok()

```
func l0c_cmd_tok(opts: CliOptions*, cfg: LogConfig*) → int
```

Execute Stage 2 `--tok` mode.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.

Returns

Process-style status code.

8.63.1.8 l0c_cmd_type()

```
func l0c_cmd_type(opts:  CliOptions*, cfg:  LogConfig*) → int
```

Execute Stage 2 `--type` mode.

Parameters

opts	Parsed CLI options.
cfg	Logging configuration.

Returns

Process-style status code.

8.63.1.9 l0c_collect_c_option_words()

```
func l0c_collect_c_option_words(opts:  CliOptions*) → StringVector*
```

Resolve merged C option words from `L0_CFLAGS` and parsed CLI options.

Parameters

opts	Parsed CLI options.

Returns

Parsed option words in final order.

8.63.1.10 l0c_copy_collector()

```
func l0c_copy_collector(dst:  DiagCollector*, src:  DiagCollector*)
```

Copy every diagnostic from one collector into another.

Parameters

dst	Destination collector.
src	Source collector.

8.63.1.11 l0c_dump_tokens_for_path()

```
func l0c_dump_tokens_for_path(path: string, include_eof: bool, diags: DiagCollector*) → int
```

Lex a source file and print its tokens.

Parameters

path	Source file to read.
include_eof	Whether to print the trailing EOF token.
diags	Diagnostic collector for I/O and lexing errors.

Returns

Process-style status code.

8.63.1.12 l0c_log_effective_roots()

```
func l0c_log_effective_roots(cfg: LogConfig*, sys_roots: StringVector*, project_roots: StringVector*)
```

Log the effective system and project roots for one CLI invocation.

Parameters

cfg	Logging configuration.
sys_roots	Effective system-root vector.
project_roots	Effective project-root vector.

8.63.1.13 l0c_merge_c_option_words()

```
func l0c_merge_c_option_words(env_opts: string?, cli_opts: string?) → StringVector*
```

Merge C option words from environment and CLI with CLI-last precedence.

Parameters

env_opts	Options from `L0_CFLAGS`.
cli_opts	Options from `--c-options`.

Returns

Parsed option words in final order.

8.63.1.14 l0c_prepare_input()

```
func l0c_prepare_input(opts: CliOptions*, diags: DiagCollector*, cfg: LogConfig*) → PreparedInput*?
```

Normalize CLI input into an entry module and search paths.

Parameters

opts	Parsed CLI options.
diags	Diagnostic collector for normalization errors.
cfg	Logging configuration.

Returns

Prepared input bundle, or `null` if normalization fails.

8.63.1.15 l0c_print_module_token_header()

```
func l0c_print_module_token_header(module_name: string, path: string)
```

Print a header before dumping tokens for a module.

Parameters

module_name	Module being printed.
path	Source path for the module.

8.63.1.16 l0c_print_token()

```
func l0c_print_token(path: string, tok: Token)
```

Print one token with source location metadata.

Parameters

path	Source path owning the token.
tok	Token to print.

8.63.1.17 prepared_input_free()

```
func prepared_input_free(self: PreparedInput*)
```

Free a prepared CLI input bundle.

Parameters

self	Prepared input to free.

8.63.1.18 prepared_input_free_opt()

```
func prepared_input_free_opt(self:  PreparedInput*?)
```

Free a nullable prepared input bundle when present.

Parameters

self	Optional prepared input to free.

8.63.1.19 run_with_argv()

```
func run_with_argv(argv:  StringVector*) → int
```

Parse CLI arguments and dispatch the requested command.

Parameters

argv	Argument vector excluding the executable name.

Returns

Process-style status code.

8.64 compiler/stage2_l0/src/lexer.l0 File Reference

Module: `lexer`

```
import std.linear_map;
import std.vector;
import util.strings;
import std.text;
import std.unit;
import tokens;
import std.string;
import util.diag;
import std.io;
```

Classes

- struct LexerState

  Lexer state object that keeps track of the current position in the source code being lexed.
- struct EscapedChar

  Represents the EscapedChar structure.

Functions

- func is_ident_start(c: byte) → bool

  Check whether a byte can start an identifier.
- func is_ident_part(c: byte) → bool

  Check whether a byte can continue an identifier.
- func is_escape_char(c: byte) → bool

  Check whether a byte is a supported escape-code character.
- func escape_char_value(c: byte) → byte

  Decode one simple escape-code character into its byte value.
- func is_octal_digit(c: byte) → bool

  Check whether a byte is an octal digit.
- func is_hex_digit(c: byte) → bool

  Check whether a byte is a hexadecimal digit.
- func hex_char_to_int(c: byte) → int

  Convert a hexadecimal digit character to its integer value.
- func is_printable_ascii(c: byte) → bool

  Check whether a byte is a printable ASCII character (space through tilde).
- func ls_create(source: string, filename: string) → LexerState∗

  Create a lexer state for source text and filename metadata.
- func ls_has_errors(self: LexerState∗) → bool

  Report whether lexing has produced any errors.
- func ls_emit_error(self: LexerState∗, code: string, message: string, line: int, column: int, line_end: int, column_end: int)

  Emit a lexer diagnostic at an explicit source span.
- func ls_queue_token(self: LexerState∗, token: Token)

  Queue a physical token for later emission by `ls_next_token`.
- func ls_has_queued_tokens(self: LexerState∗) → bool

  Return whether previously queued tokens remain to be emitted.
- func ls_take_queued_token(self: LexerState∗) → Token

  Return the next queued token and transfer ownership out of the queue.
- func ls_defer_recoverable_error(self: LexerState∗, code: string, message: string, line: int, column: int, line_end: int, column_end: int)

  Store a recoverable lexer diagnostic for the token currently being scanned.
- func ls_clear_pending_error(self: LexerState∗)

  Clear pending recoverable diagnostic state before scanning a token.
- func ls_queue_pending_errors(self: LexerState∗, start_index: int)

  Queue all pending recoverable diagnostics as unrecoverable lexer-error tokens.
- func ls_queue_pending_recovery(self: LexerState∗, start_index: int, start_line: int, start_column: int, recovery: TokenRecovery) → Token

  Queue pending recoverable diagnostics and a final recoverable wrapper token.
- func ls_queue_terminal_recovery(self: LexerState∗, start_index: int, start_line: int, start_column: int, code: string, message: string, line_end: int, column_end: int, recovery: TokenRecovery) → Token

  Queue pending recoverable diagnostics followed by a terminal recoverable wrapper token.
- func ls_free(self: LexerState∗)

  Free a lexer state and its diagnostics.
- func ls_at_end(self: LexerState∗) → bool

  Check whether the lexer cursor is at the end of the source.
- func ls_peek(self: LexerState∗) → byte

  Return the current source byte without advancing.
- func ls_peek_next(self: LexerState∗) → byte

Return the byte after the current source byte without advancing.

- func ls_advance(self: LexerState*) → byte

  Consume and return the current source byte.
- func tokenize(self: LexerState*) → TokenVector?

  Tokenizes the input source code and returns a vector of tokens.
- func ls_next_token(self: LexerState*) → Token?

  Reads the next token from the input source code.
- func ls_read_byte_literal(self: LexerState*, start_index: int, start_line: int, start_column: int) → Token?

  Reads a byte literal from the input, starting after the opening single quote.
- func ls_read_string_literal(self: LexerState*, start_index: int, start_line: int, start_column: int) → Token?

  Reads a string literal from the input, starting after the opening double quote.
- func ls_read_valid_char_escape(self: LexerState*, single_byte: bool) → EscapedChar?

  Reads a valid escape sequence from the input, starting after the backslash.
- func ls_read_number(self: LexerState*, first_char: byte, is_negative: bool, start_index: int, start_line: int, start_column: int) → Token?

  Reads an integer literal from the input, starting with the first digit character.
- func ls_skip_whitespace_and_comments(self: LexerState*) → Unit?

  Skips over any whitespace characters and comments in the input until it reaches a non-whitespace, non-comment character or the end of the input.
- func ls_skip_invalid_characters(self: LexerState*)

  Skips over the rest of an invalid-character run after the first invalid byte has been consumed.

8.64.1 Function Documentation

8.64.1.1 escape_char_value()

```
func escape_char_value(c:  byte) → byte
```

Decode one simple escape-code character into its byte value.

Parameters

c	Escape-code byte after the backslash.

Returns

Decoded byte value.

8.64.1.2 hex_char_to_int()

```
func hex_char_to_int(c:  byte) → int
```

Convert a hexadecimal digit character to its integer value.

Parameters

c	Hexadecimal digit byte.

Returns

Integer value in the range `0..15`, or `-1` if `c` is not valid.

8.64.1.3 is_escape_char()

```
func is_escape_char(c:  byte) → bool
```

Check whether a byte is a supported escape-code character.

Parameters

c	Byte to inspect.

Returns

`true` if `c` is accepted after a backslash in literals.

8.64.1.4 is_hex_digit()

```
func is_hex_digit(c:  byte) → bool
```

Check whether a byte is a hexadecimal digit.

Parameters

c	Byte to inspect.

Returns

`true` if `c` is in `0..9`, `a..f`, or `A..F`.

8.64.1.5 is_ident_part()

```
func is_ident_part(c:  byte) → bool
```

Check whether a byte can continue an identifier.

Parameters

c	Byte to inspect.

Returns

`true` if `c` is alphanumeric or _.

8.64.1.6 is_ident_start()

```
func is_ident_start(c:  byte) → bool
```

Check whether a byte can start an identifier.

Parameters

c	Byte to inspect.

Returns

`true` if `c` is alphabetic or _.

8.64.1.7 is_octal_digit()

```
func is_octal_digit(c:  byte) → bool
```

Check whether a byte is an octal digit.

Parameters

c	Byte to inspect.

Returns

`true` if `c` is in the range `0..7`.

8.64.1.8 is_printable_ascii()

```
func is_printable_ascii(c:  byte) → bool
```

Check whether a byte is a printable ASCII character (space through tilde).

Parameters

c	Byte to inspect.

Returns

`true` if `c` is a space, tab, carriage return, or newline.

8.64.1.9 ls_advance()

```
func ls_advance(self:  LexerState*) → byte
```

Consume and return the current source byte.

The column advances once per Unicode code point, not per byte: UTF-8 continuation bytes do not bump it, so positions stay aligned with the code-point columns reported by the Stage 1 Python frontend.

Parameters

self	Lexer state to advance.

Returns

Byte that was current before advancing.

8.64.1.10 ls_at_end()

```
func ls_at_end(self: LexerState*) → bool
```

Check whether the lexer cursor is at the end of the source.

Parameters

self	Lexer state to inspect.

Returns

`true` if no more source bytes remain.

8.64.1.11 ls_clear_pending_error()

```
func ls_clear_pending_error(self: LexerState*)
```

Clear pending recoverable diagnostic state before scanning a token.

8.64.1.12 ls_create()

```
func ls_create(source: string, filename: string) → LexerState*
```

Create a lexer state for source text and filename metadata.

Parameters

source	Source text to tokenize.
filename	Source filename used in diagnostics.

Returns

New lexer state.

8.64.1.13 ls_defer_recoverable_error()

```
func ls_defer_recoverable_error(self: LexerState*, code: string, message: string, line:
int, column: int, line_end: int, column_end: int)
```

Store a recoverable lexer diagnostic for the token currently being scanned.

8.64.1.14 ls_emit_error()

```
func ls_emit_error(self: LexerState*, code: string, message: string, line: int, column:
int, line_end: int, column_end: int)
```

Emit a lexer diagnostic at an explicit source span.

Parameters

self	Lexer state collecting diagnostics.
code	Diagnostic code.
message	Diagnostic message.
line	One-based starting line.
column	One-based starting column.
line_end	One-based ending line.
column_end	One-based ending column.

8.64.1.15 ls_free()

```
func ls_free(self: LexerState*)
```

Free a lexer state and its diagnostics.

Parameters

self	Lexer state to free.

8.64.1.16 ls_has_errors()

```
func ls_has_errors(self: LexerState*) → bool
```

Report whether lexing has produced any errors.

Parameters

self	Lexer state to inspect.

Returns

`true` if the lexer's diagnostic collector contains an error.

8.64.1.17 ls_has_queued_tokens()

```
func ls_has_queued_tokens(self: LexerState*) → bool
```

Return whether previously queued tokens remain to be emitted.

Parameters

self	Lexer state to inspect.

Returns

`true` when queued tokens remain.

8.64.1.18 ls_next_token()

```
func ls_next_token(self: LexerState*) → Token?
```

Reads the next token from the input source code.

Returns a Token if successful, or `null` if a lexing error occurs, in which case the error field in the LexerState will be set.

8.64.1.19 ls_peek()

```
func ls_peek(self: LexerState*) → byte
```

Return the current source byte without advancing.

Parameters

self	Lexer state to inspect.

Returns

Current source byte, or `\\0` at end of input.

8.64.1.20 ls_peek_next()

```
func ls_peek_next(self: LexerState*) → byte
```

Return the byte after the current source byte without advancing.

Parameters

self	Lexer state to inspect.

Returns

Next source byte, or `\\0` if there is no next byte.

8.64.1.21 ls_queue_pending_errors()

```
func ls_queue_pending_errors(self:  LexerState*, start_index:  int)
```

Queue all pending recoverable diagnostics as unrecoverable lexer-error tokens.

Parameters

self	Lexer state to drain.
start_index	Physical token start offset.

8.64.1.22 ls_queue_pending_recovery()

```
func ls_queue_pending_recovery(self:  LexerState*, start_index:  int, start_line:  int, start_column:
int, recovery:  TokenRecovery) → Token
```

Queue pending recoverable diagnostics and a final recoverable wrapper token.

Parameters

self	Lexer state to drain.
start_index	Physical token start offset.
start_line	Physical token start line.
start_column	Physical token start column.
recovery	Logical recovery payload for the final wrapper.

Returns

First queued token.

8.64.1.23 ls_queue_terminal_recovery()

```
func ls_queue_terminal_recovery(self:  LexerState*, start_index:  int, start_line:  int, start_column:
int, code:  string, message:  string, line_end:  int, column_end:  int, recovery:  TokenRecovery)
→ Token
```

Queue pending recoverable diagnostics followed by a terminal recoverable wrapper token.

Parameters

self	Lexer state to drain.
start_index	Physical token start offset.
start_line	Physical token start line.
start_column	Physical token start column.
code	Terminal diagnostic code.
message	Terminal diagnostic message.
line_end	Terminal diagnostic end line.
column_end	Terminal diagnostic end column.
recovery	Logical recovery payload for the final wrapper.

Returns

First queued token.

8.64.1.24 ls_queue_token()

```
func ls_queue_token(self: LexerState*, token: Token)
```

Queue a physical token for later emission by `ls_next_token`.

Parameters

self	Lexer state collecting queued tokens.
token	Token to queue.

8.64.1.25 ls_read_byte_literal()

```
func ls_read_byte_literal(self: LexerState*, start_index: int, start_line: int, start_column:
int) → Token?
```

Reads a byte literal from the input, starting after the opening single quote.

Handles escape sequences and validates that the literal represents a single byte. Returns a Token of type TT_BYTE if successful, or `null` if a lexing error occurs, in which case the error field in the LexerState will be set.

8.64.1.26 ls_read_number()

```
func ls_read_number(self: LexerState*, first_char: byte, is_negative: bool, start_index:
int, start_line: int, start_column: int) → Token?
```

Reads an integer literal from the input, starting with the first digit character.

Validates that the literal is well-formed and does not contain invalid characters immediately following it. Also checks for integer overflow and ensures the value fits within a 32-bit signed integer range. Returns a Token of type TT_INT if successful, or `null` if a lexing error occurs, in which case the error field in the LexerState will be set.

8.64.1.27 ls_read_string_literal()

```
func ls_read_string_literal(self: LexerState*, start_index: int, start_line: int, start_column:
int) → Token?
```

Reads a string literal from the input, starting after the opening double quote.

Handles escape sequences and validates that the string is properly terminated. Returns a Token of type TT_↩ STRING if successful, or `null` if a lexing error occurs, in which case the error field in the LexerState will be set.

8.64.1.28 ls_read_valid_char_escape()

```
func ls_read_valid_char_escape(self: LexerState*, single_byte: bool) → EscapedChar?
```

Reads a valid escape sequence from the input, starting after the backslash.

Validates that the escape sequence is well-formed and returns both the original escape string and its corresponding character value. If `single_byte` is true, also validates that the resulting character value can fit in a single byte (0-255). Returns an EscapedChar struct if successful, or `null` if an invalid escape sequence is encountered, in which case the error field in the LexerState will be set.

8.64.1.29 ls_skip_invalid_characters()

```
func ls_skip_invalid_characters(self: LexerState*)
```

Skips over the rest of an invalid-character run after the first invalid byte has been consumed.

The run stops at printable ASCII or whitespace, so the resulting diagnostic span never crosses a line break.

8.64.1.30 ls_skip_whitespace_and_comments()

```
func ls_skip_whitespace_and_comments(self: LexerState*) → Unit?
```

Skips over any whitespace characters and comments in the input until it reaches a non-whitespace, non-comment character or the end of the input.

Handles both line comments and block comments. Returns std.unit::present() if successful, or `null` if an unterminated block comment is encountered, in which case the error field in the LexerState will be set.

8.64.1.31 ls_take_queued_token()

```
func ls_take_queued_token(self: LexerState*) → Token
```

Return the next queued token and transfer ownership out of the queue.

Parameters

self	Lexer state to inspect.

Returns

Next queued token.

8.64.1.32 tokenize()

```
func tokenize(self:  LexerState*) → TokenVector?
```

Tokenizes the input source code and returns a vector of tokens.

If a lexing error occurs, the error field in the LexerState will be set and null will be returned.

8.65 compiler/stage2_l0/src/locals.l0 File Reference

Module: `locals`

```
import driver;
import ast;
import util.diag;
import std.vector;
import std.text;
import std.string;
import util.strings;
import std.hashmap;
```

Classes

- struct LocalSymbol

  Represents the LocalSymbol structure.
- struct Scope

  Represents the Scope structure.
- struct MatchArmScopeEntry

  Represents the MatchArmScopeEntry structure.
- struct ScopeSlot

  Represents the ScopeSlot structure.
- struct FunctionEnv

  Represents the FunctionEnv structure.
- struct LocalsResult

  Represents the LocalsResult structure.

Enumerations

- enum LocalKind

  Defines the LocalKind enumeration.

Functions

- func lc_key(module_name: string, func_name: string) → string

  Build a fully qualified function-scope key.
- func lc_scope_create(parent: Scope∗?) → Scope∗

  Create a local scope with an optional parent scope.
- func lc_scope_define(self: Scope∗, name: string, kind: LocalKind, decl_ptr: void∗, decl_id: int, binding_slot: int)

  Define a local symbol in the current scope if absent.
- func lc_scope_lookup(self: Scope∗?, name: string) → LocalSymbol∗?

  Look up a local symbol through the scope chain.
- func lc_scope_free(self: Scope∗)

  Free a scope and its local-symbol table.
- func lc_func_env_free(self: FunctionEnv∗)

  Free one function-local environment.
- func locals_free(self: LocalsResult∗)

  Free the full locals-resolution result.
- func locals_env_count(self: LocalsResult∗) → int

  Return the number of function environments in the locals result.
- func locals_env_get(self: LocalsResult∗, index: int) → FunctionEnv∗

  Return a function environment by index.
- func locals_env_find(self: LocalsResult∗, module_name: string, func_name: string) → FunctionEnv∗?

  Find a function environment by module and function name.
- func lc_func_env_get_block_scope(self: FunctionEnv∗, stmt_id: StmtId) → Scope∗?

  Return the scope associated with a block statement.
- func lc_func_env_get_match_arm_scope(self: FunctionEnv∗, arm: MatchArm∗) → Scope∗?

  Return the scope associated with a match arm.
- func lc_scope_track(env: FunctionEnv∗, scope: Scope∗)

  Record a scope so the function environment can free it later.
- func lc_scope_set_block(env: FunctionEnv∗, stmt_id: StmtId, scope: Scope∗)

  Associate a block statement id with a scope.
- func lc_bind_pattern(env: FunctionEnv∗, pattern_id: PatternId, scope: Scope∗)

  Bind pattern variables for a match arm into a scope.
- func lc_visit_block(env: FunctionEnv∗, block_id: StmtId, scope: Scope∗)

  Visits a block during AST traversal.
- func lc_visit_scoped_stmt(env: FunctionEnv∗, stmt_id: StmtId, scope: Scope∗)

  Visit a statement body under an already-created scope.
- func lc_visit_match_arm(env: FunctionEnv∗, arm: MatchArm∗, parent_scope: Scope∗)

  Visits a arm during AST traversal.
- func lc_visit_case_arm(env: FunctionEnv∗, arm: CaseArm∗, parent_scope: Scope∗)

  Visits a arm during AST traversal.
- func lc_visit_case_else(env: FunctionEnv∗, arm: CaseElse∗, parent_scope: Scope∗)

  Visits a else during AST traversal.
- func lc_visit_with(env: FunctionEnv∗, stmt_id: StmtId, stmt: StmtNode∗, parent_scope: Scope∗)

  Visits a with during AST traversal.
- func lc_visit_stmt(env: FunctionEnv∗, stmt_id: StmtId, scope: Scope∗)

  Visits a stmt during AST traversal.
- func lc_make_function_env(module_name: string, unit: DriverUnit∗, decl: FuncDecl∗) → FunctionEnv∗

  Build locals metadata for one non-extern function.
- func locals_resolve(driver_state: DriverState∗, diags: DiagCollector∗) → LocalsResult∗

  Build local-scope environments for every non-extern function.

8.65.1 Enumeration Type Documentation

8.65.1.1 LocalKind

```
enum LocalKind
```

Defines the LocalKind enumeration.

Enumerator

LK_PARAM	
LK_LOCAL	
LK_PATTERN_VAR	

8.65.2 Function Documentation

8.65.2.1 lc_bind_pattern()

```
func lc_bind_pattern(env: FunctionEnv*, pattern_id: PatternId, scope: Scope*)
```

Bind pattern variables for a match arm into a scope.

Parameters

env	Function environment providing the pattern arena.
pattern↩ _id	Pattern to inspect.
scope	Scope to receive bindings.

8.65.2.2 lc_func_env_free()

```
func lc_func_env_free(self: FunctionEnv*)
```

Free one function-local environment.

Parameters

self	Function environment to free.

8.65.2.3 lc_func_env_get_block_scope()

```
func lc_func_env_get_block_scope(self: FunctionEnv*, stmt_id: StmtId) → Scope*?
```

Return the scope associated with a block statement.

Parameters

self	Function environment to inspect.
stmt↩ _id	Block statement id.

Returns

Scope for `stmt_id`, or `null` if none is recorded.

8.65.2.4 lc_func_env_get_match_arm_scope()

```
func lc_func_env_get_match_arm_scope(self:  FunctionEnv*, arm:  MatchArm*) → Scope*?
```

Return the scope associated with a match arm.

Parameters

self	Function environment to inspect.
arm	Match arm to look up.

Returns

Scope for `arm`, or `null` if none is recorded.

8.65.2.5 lc_key()

```
func lc_key(module_name:  string, func_name:  string) → string
```

Build a fully qualified function-scope key.

Parameters

module_name	Defining module name.
func_name	Function name.

Returns

Key in `module::func` form.

8.65.2.6 lc_make_function_env()

```
func lc_make_function_env(module_name:  string, unit:  DriverUnit*, decl:  FuncDecl*) → FunctionEnv*
```

Build locals metadata for one non-extern function.

Parameters

module_name	Defining module name.
unit	Driver unit owning the function.
decl	Function declaration.

Returns

Populated function environment.

8.65.2.7 lc_scope_create()

```
func lc_scope_create(parent:  Scope*?)  → Scope*
```

Create a local scope with an optional parent scope.

Parameters

parent	Parent scope, or `null` for the root scope.

Returns

New scope.

8.65.2.8 lc_scope_define()

```
func lc_scope_define(self:  Scope*, name:  string, kind:  LocalKind, decl_ptr:  void*, decl_id:
int, binding_slot:  int)
```

Define a local symbol in the current scope if absent.

Parameters

self	Scope to update.
name	Local name to bind.
kind	Kind of local binding.
decl_ptr	AST pointer associated with the binding.
decl_id	Owning statement, pattern, or parameter index.
binding_slot	Pattern binder slot, or `-1` for other bindings.

8.65.2.9 lc_scope_free()

```
func lc_scope_free(self:  Scope*)
```

Free a scope and its local-symbol table.

Parameters

self	Scope to free.

8.65.2.10 lc_scope_lookup()

```
func lc_scope_lookup(self:  Scope*?, name:  string) → LocalSymbol*?
```

Look up a local symbol through the scope chain.

Parameters

self	Starting scope, or `null`.
name	Local name to look up.

Returns

Matching symbol, or `null` if not found.

8.65.2.11 lc_scope_set_block()

```
func lc_scope_set_block(env:  FunctionEnv*, stmt_id:  StmtId, scope:  Scope*)
```

Associate a block statement id with a scope.

Parameters

env	Function environment being populated.
stmt↩ _id	Block statement id.
scope	Scope to attach.

8.65.2.12 lc_scope_track()

```
func lc_scope_track(env:  FunctionEnv*, scope:  Scope*)
```

Record a scope so the function environment can free it later.

Parameters

env	Function environment being populated.
scope	Scope to record.

8.65.2.13 lc_visit_block()

```
func lc_visit_block(env:  FunctionEnv*, block_id:  StmtId, scope:  Scope*)
```

Visits a block during AST traversal.

Parameters

env	The current module environment.
block↩ _id	The block id.
scope	The scope.

8.65.2.14 lc_visit_case_arm()

```
func lc_visit_case_arm(env:  FunctionEnv*, arm:  CaseArm*, parent_scope:  Scope*)
```

Visits a arm during AST traversal.

Parameters

env	The current module environment.
arm	The arm.
parent_scope	The parent scope.

8.65.2.15 lc_visit_case_else()

```
func lc_visit_case_else(env:  FunctionEnv*, arm:  CaseElse*, parent_scope:  Scope*)
```

Visits a else during AST traversal.

Parameters

env	The current module environment.
arm	The arm.
parent_scope	The parent scope.

8.65.2.16 lc_visit_match_arm()

```
func lc_visit_match_arm(env:  FunctionEnv*, arm:  MatchArm*, parent_scope:  Scope*)
```

Visits a arm during AST traversal.

Parameters

env	The current module environment.
arm	The arm.
parent_scope	The parent scope.

8.65.2.17 lc_visit_scoped_stmt()

```
func lc_visit_scoped_stmt(env: FunctionEnv*, stmt_id: StmtId, scope: Scope*)
```

Visit a statement body under an already-created scope.

Parameters

env	Current function environment.
stmt_id	Body statement id.
scope	Body scope.

8.65.2.18 lc_visit_stmt()

```
func lc_visit_stmt(env: FunctionEnv*, stmt_id: StmtId, scope: Scope*)
```

Visits a stmt during AST traversal.

Parameters

env	The current module environment.
stmt_id	The stmt id.
scope	The scope.

8.65.2.19 lc_visit_with()

```
func lc_visit_with(env: FunctionEnv*, stmt_id: StmtId, stmt: StmtNode*, parent_scope: Scope*)
```

Visits a with during AST traversal.

Parameters

env	The current module environment.
stmt_id	The `with` statement id.
stmt	The stmt.
parent_scope	The parent scope.

8.65.2.20 locals_env_count()

```
func locals_env_count(self: LocalsResult*) → int
```

Return the number of function environments in the locals result.

Parameters

self	Locals result to inspect.

Returns

Function-environment count.

8.65.2.21 locals_env_find()

```
func locals_env_find(self: LocalsResult*, module_name: string, func_name: string) → FunctionEnv*?
```

Find a function environment by module and function name.

Parameters

self	Locals result to inspect.
module_name	Module name.
func_name	Function name.

Returns

Matching function environment, or `null` if absent.

8.65.2.22 locals_env_get()

```
func locals_env_get(self: LocalsResult*, index: int) → FunctionEnv*
```

Return a function environment by index.

Parameters

self	Locals result to inspect.
index	Zero-based function-environment index.

Returns

Function environment at `index`.

8.65.2.23 locals_free()

```
func locals_free(self: LocalsResult*)
```

Free the full locals-resolution result.

Parameters

self	Locals result to free.

8.65.2.24 locals_resolve()

```
func locals_resolve(driver_state:  DriverState*, diags:  DiagCollector*) → LocalsResult*
```

Build local-scope environments for every non-extern function.

Parameters

driver_state	Parsed driver state.
diags	Diagnostic collector.

Returns

Locals-resolution result.

8.66 compiler/stage2_l0/src/name_resolver.l0 File Reference

Module: `name_resolver`

```
import symbols;
import ast;
import util.diag;
import std.string;
import util.strings;
import std.hashset;
import std.hashmap;
import driver;
import std.vector;
import std.text;
import sys.rt;
```

Classes

- struct ModuleEnv

  Represents the ModuleEnv structure.
- struct NameResolution

  Represents the NameResolution structure.

Functions

- func nr_span_diag_error(diags: DiagCollector∗, code: string, message: string, filename: string, span: Span)

  Emit a name-resolution error for a source span.
- func nr_span_diag_warn(diags: DiagCollector∗, code: string, message: string, filename: string, span: Span)

  Emit a name-resolution warning for a source span.
- func nr_map_symbol_get(self: StringPtrMap∗, name: string) → Symbol∗?

  Return a symbol from a string-pointer map when present.
- func nr_build_top_key(module_name: string, name: string) → string

  Build a fully qualified top-level symbol key.
- func nr_path_is_same_signature(left: TypeRef∗?, right: TypeRef∗?) → bool

  Compare two parsed type references for signature compatibility.
- func nr_extern_signatures_compatible(local_sym: Symbol∗, imported_sym: Symbol∗) → bool

  Check whether two extern function symbols have compatible signatures.
- func nr_symbol_free_map(self: StringPtrMap∗)

  Free a symbol map that owns its symbol values.
- func nr_module_env_free(self: ModuleEnv∗)

  Free a module environment and its owned local symbols.
- func nr_free(self: NameResolution∗)

  Free a complete name-resolution result.
- func nr_env_count(self: NameResolution∗) → int

  Return the number of module environments in the result.
- func nr_env_get(self: NameResolution∗, index: int) → ModuleEnv∗

  Return a module environment by index.
- func nr_env_find(self: NameResolution∗, module_name: string) → ModuleEnv∗?

  Find a module environment by module name.
- func nr_module_env_local(self: ModuleEnv∗, name: string) → Symbol∗?

  Look up a locally defined symbol in a module environment.
- func nr_module_env_imported(self: ModuleEnv∗, name: string) → Symbol∗?

  Look up an imported symbol in a module environment.
- func nr_module_env_all(self: ModuleEnv∗, name: string) → Symbol∗?

  Look up any visible symbol in a module environment.
- func nr_module_env_is_ambiguous(self: ModuleEnv∗, name: string) → bool

  Check whether an imported name is ambiguous in a module environment.
- func nr_module_env_imports_module(self: ModuleEnv∗, module_name: string) → bool

  Check whether a module environment imports a given module.
- func nr_module_env_local_keys(self: ModuleEnv∗) → StringVector∗

  Return the names of locally defined symbols.
- func nr_module_env_imported_keys(self: ModuleEnv∗) → StringVector∗

  Return the names of imported symbols.
- func nr_module_env_all_keys(self: ModuleEnv∗) → StringVector∗

  Return the names of all visible non-ambiguous symbols.
- func nr_define_local(env: ModuleEnv∗, diags: DiagCollector∗, name: string, sym: Symbol∗)

  Add a top-level local symbol to a module environment.
- func nr_collect_locals(env: ModuleEnv∗, diags: DiagCollector∗)

  Collect locally declared top-level symbols for one module.
- func nr_open_imports_one(env: ModuleEnv∗, imported_env: ModuleEnv∗, diags: DiagCollector∗, imp: Import∗)

  Merge one imported module's visible symbols into a module environment.
- func nr_open_imports(nr: NameResolution∗, env: ModuleEnv∗, diags: DiagCollector∗)

  Resolve and open every import for a module environment.

- func nr_make_env(module_ptr: Module∗) → ModuleEnv∗

  Create an empty module environment for a parsed module.
- func nr_resolve(driver_state: DriverState∗, diags: DiagCollector∗) → NameResolution∗

  Build module environments and resolve opened imports.

8.66.1 Function Documentation

8.66.1.1 nr_build_top_key()

```
func nr_build_top_key(module_name:  string, name:  string) → string
```

Build a fully qualified top-level symbol key.

Parameters

module_name	Defining module name.
name	Symbol name.

Returns

Key in `module::name` form.

8.66.1.2 nr_collect_locals()

```
func nr_collect_locals(env:  ModuleEnv∗, diags:  DiagCollector∗)
```

Collect locally declared top-level symbols for one module.

Parameters

env	Module environment being populated.
diags	Diagnostic collector.

8.66.1.3 nr_define_local()

```
func nr_define_local(env:  ModuleEnv∗, diags:  DiagCollector∗, name:  string, sym:  Symbol∗)
```

Add a top-level local symbol to a module environment.

Parameters

env	Module environment being populated.
diags	Diagnostic collector for duplicate-definition errors.
name	Symbol name.
sym	Symbol to insert.

8.66.1.4 nr_env_count()

```
func nr_env_count(self:  NameResolution*) → int
```

Return the number of module environments in the result.

Parameters

self	Name-resolution result to inspect.

Returns

Module-environment count.

8.66.1.5 nr_env_find()

```
func nr_env_find(self:  NameResolution*, module_name:  string) → ModuleEnv*?
```

Find a module environment by module name.

Parameters

self	Name-resolution result to inspect.
module_name	Module name to look up.

Returns

Matching environment, or `null` if absent.

8.66.1.6 nr_env_get()

```
func nr_env_get(self:  NameResolution*, index:  int) → ModuleEnv*
```

Return a module environment by index.

Parameters

self	Name-resolution result to inspect.
index	Zero-based environment index.

Returns

Module environment at `index`.

8.66.1.7 nr_extern_signatures_compatible()

```
func nr_extern_signatures_compatible(local_sym:  Symbol*, imported_sym:  Symbol*) → bool
```

Check whether two extern function symbols have compatible signatures.

Parameters

local_sym	Local symbol.
imported_sym	Imported symbol.

Returns

`true` if both are extern functions with matching signatures.

8.66.1.8 nr_free()

```
func nr_free(self:  NameResolution*)
```

Free a complete name-resolution result.

Parameters

self	Name-resolution result to free.

8.66.1.9 nr_make_env()

```
func nr_make_env(module_ptr:  Module*) → ModuleEnv*
```

Create an empty module environment for a parsed module.

Parameters

module_ptr	Parsed module.

Returns

New module environment for `module_ptr`.

8.66.1.10 nr_map_symbol_get()

```
func nr_map_symbol_get(self:  StringPtrMap*, name:  string) → Symbol*?
```

Return a symbol from a string-pointer map when present.

Parameters

self	Symbol map to inspect.
name	Symbol name to look up.

Returns

Matching symbol, or `null` if absent.

8.66.1.11 nr_module_env_all()

```
func nr_module_env_all(self:  ModuleEnv*, name:  string) → Symbol*?
```

Look up any visible symbol in a module environment.

Parameters

self	Module environment to inspect.
name	Symbol name to look up.

Returns

Visible symbol, or `null` if absent or ambiguous.

8.66.1.12 nr_module_env_all_keys()

```
func nr_module_env_all_keys(self:  ModuleEnv*) → StringVector*
```

Return the names of all visible non-ambiguous symbols.

Parameters

self	Module environment to inspect.

Returns

New vector of visible symbol names.

8.66.1.13 nr_module_env_free()

```
func nr_module_env_free(self:  ModuleEnv*)
```

Free a module environment and its owned local symbols.

Parameters

self	Module environment to free.

8.66.1.14 nr_module_env_imported()

```
func nr_module_env_imported(self:  ModuleEnv*, name:  string) → Symbol*?
```

Look up an imported symbol in a module environment.

Parameters

self	Module environment to inspect.
name	Symbol name to look up.

Returns

Imported symbol, or `null` if absent.

8.66.1.15 nr_module_env_imported_keys()

```
func nr_module_env_imported_keys(self:  ModuleEnv*) → StringVector*
```

Return the names of imported symbols.

Parameters

self	Module environment to inspect.

Returns

New vector of imported symbol names.

8.66.1.16 nr_module_env_imports_module()

```
func nr_module_env_imports_module(self:  ModuleEnv*, module_name:  string) → bool
```

Check whether a module environment imports a given module.

Parameters

self	Module environment to inspect.
module_name	Imported module name to check.

Returns

`true` if `module_name` appears in the import set.

8.66.1.17 nr_module_env_is_ambiguous()

```
func nr_module_env_is_ambiguous(self:  ModuleEnv*, name:  string) → bool
```

Check whether an imported name is ambiguous in a module environment.

Parameters

self	Module environment to inspect.
name	Symbol name to check.

Returns

`true` if multiple imports provide `name`.

8.66.1.18 nr_module_env_local()

```
func nr_module_env_local(self: ModuleEnv*, name: string) → Symbol*?
```

Look up a locally defined symbol in a module environment.

Parameters

self	Module environment to inspect.
name	Symbol name to look up.

Returns

Local symbol, or `null` if absent.

8.66.1.19 nr_module_env_local_keys()

```
func nr_module_env_local_keys(self: ModuleEnv*) → StringVector*
```

Return the names of locally defined symbols.

Parameters

self	Module environment to inspect.

Returns

New vector of local symbol names.

8.66.1.20 nr_open_imports()

```
func nr_open_imports(nr: NameResolution*, env: ModuleEnv*, diags: DiagCollector*)
```

Resolve and open every import for a module environment.

Parameters

nr	Full name-resolution state.
env	Module environment being populated.
diags	Diagnostic collector.

8.66.1.21 nr_open_imports_one()

```
func nr_open_imports_one(env: ModuleEnv*, imported_env: ModuleEnv*, diags: DiagCollector*,
imp: Import*)
```

Merge one imported module's visible symbols into a module environment.

Parameters

env	Importing module environment.
imported_env	Imported module environment.
diags	Diagnostic collector.
imp	Import declaration being processed.

8.66.1.22 nr_path_is_same_signature()

```
func nr_path_is_same_signature(left: TypeRef*?, right: TypeRef*?) → bool
```

Compare two parsed type references for signature compatibility.

Parameters

left	First type reference.
right	Second type reference.

Returns

`true` if the two type references describe the same type syntax.

8.66.1.23 nr_resolve()

```
func nr_resolve(driver_state: DriverState*, diags: DiagCollector*) → NameResolution*
```

Build module environments and resolve opened imports.

Parameters

driver_state	Parsed driver state.
diags	Diagnostic collector.

Returns

Name-resolution result containing per-module environments.

8.66.1.24 nr_span_diag_error()

```
func nr_span_diag_error(diags: DiagCollector*, code: string, message: string, filename:
string, span: Span)
```

Emit a name-resolution error for a source span.

Parameters

diags	Diagnostic collector.
code	Diagnostic code.
message	Diagnostic message.
filename	Source filename.
span	Source span to highlight.

8.66.1.25 nr_span_diag_warn()

```
func nr_span_diag_warn(diags: DiagCollector*, code: string, message: string, filename:
string, span: Span)
```

Emit a name-resolution warning for a source span.

Parameters

diags	Diagnostic collector.
code	Diagnostic code.
message	Diagnostic message.
filename	Source filename.
span	Source span to highlight.

8.66.1.26 nr_symbol_free_map()

```
func nr_symbol_free_map(self: StringPtrMap*)
```

Free a symbol map that owns its symbol values.

Parameters

self	Symbol map to free.

8.67 compiler/stage2_l0/src/parser.l0 File Reference

Module: `parser`

```
import parser.decl;
import util.diag;
import parser.shared;
import ast;
import tokens;
import lexer;
```

Classes

- struct ParseResult

  Represents the ParseResult structure.

Functions

- func parse_result_free(self: ParseResult∗)

  Free a public parse result and all owned AST arenas.
- func parse_has_errors(self: ParseResult∗) → bool

  Report whether parsing produced any errors.
- func parse_diag_count(self: ParseResult∗) → int

  Return the number of diagnostics produced during parsing.
- func parse_diag_get(self: ParseResult∗, index: int) → Diagnostic∗

  Return a parse diagnostic by index.
- func impl_result_to_public(self: ImplParseResult∗) → ParseResult∗

  Convert the internal parser result to the public parse-result type.
- func parse_module_tokens(tokens: TokenVector, filename: string) → ParseResult∗

  Parse a module from an existing token stream.
- func parse_module_tokens_owned(tokens: TokenVector, filename: string) → ParseResult∗

  Parse a module from an existing token stream and free the token buffer.
- func parse_lex_error_result(ls: LexerState∗, filename: string) → ParseResult∗

  Convert lexer diagnostics into a public parse result and free the lexer state.
- func parse_module_source(source: string, filename: string) → ParseResult∗

  Lex and parse a module from source text.

8.67.1 Function Documentation

8.67.1.1 impl_result_to_public()

```
func impl_result_to_public(self:  ImplParseResult∗) → ParseResult∗
```

Convert the internal parser result to the public parse-result type.

Parameters

self	Internal parse result to consume.

Returns

Public parse result owning the same AST arenas and diagnostics.

8.67.1.2 parse_diag_count()

```
func parse_diag_count(self:  ParseResult*) → int
```

Return the number of diagnostics produced during parsing.

Parameters

self	Parse result to inspect.

Returns

Total diagnostic count.

8.67.1.3 parse_diag_get()

```
func parse_diag_get(self:  ParseResult*, index:  int) → Diagnostic*
```

Return a parse diagnostic by index.

Parameters

self	Parse result to inspect.
index	Zero-based diagnostic index.

Returns

Diagnostic at `index`.

8.67.1.4 parse_has_errors()

```
func parse_has_errors(self:  ParseResult*) → bool
```

Report whether parsing produced any errors.

Parameters

self	Parse result to inspect.

Returns

`true` if the parse diagnostics contain an error.

8.67.1.5 parse_lex_error_result()

```
func parse_lex_error_result(ls:  LexerState*, filename:  string) → ParseResult*
```

Convert lexer diagnostics into a public parse result and free the lexer state.

Parameters

ls	Lexer state holding diagnostics.
filename	Source filename used in diagnostics.

Returns

Parse result containing the copied lexer diagnostics.

8.67.1.6 parse_module_source()

```
func parse_module_source(source:  string, filename:  string) → ParseResult*
```

Lex and parse a module from source text.

Parameters

source	Source text to parse.
filename	Source filename used in diagnostics.

Returns

Parse result for the module.

8.67.1.7 parse_module_tokens()

```
func parse_module_tokens(tokens:  TokenVector, filename:  string) → ParseResult*
```

Parse a module from an existing token stream.

Parameters

tokens	Token stream to parse.
filename	Source filename used in diagnostics.

Returns

Parse result for the module.

8.67.1.8 parse_module_tokens_owned()

```
func parse_module_tokens_owned(tokens: TokenVector, filename: string) → ParseResult*
```

Parse a module from an existing token stream and free the token buffer.

Parameters

tokens	Token stream to parse.
filename	Source filename used in diagnostics.

Returns

Parse result for the module.

8.67.1.9 parse_result_free()

```
func parse_result_free(self: ParseResult*)
```

Free a public parse result and all owned AST arenas.

Parameters

self	Parse result to free.

8.68 compiler/stage2_l0/src/parser/decl.l0 File Reference

Module: `parser.decl`

```
import parser.stmt;
import parser.expr;
import parser.shared;
import ast;
import lexer;
import tokens;
import util.diag;
import std.vector;
import std.text;
```

Functions

- func ps_make_empty_block_stmt(self: ParserState*, span: Span) → StmtId

  Create an empty block statement node in the statement arena.
- func ps_finish_function(self: ParserState*, start: Span, name_tok: Token, params: VectorBase*, is_extern: bool, ret_type: TypeRef*) → FuncDecl*?

  Finish parsing a function declaration after its signature.

- func ps_parse_function(self: ParserState*, is_extern: bool) → FuncDecl*?

  Parse a function declaration.
- func ps_parse_extern_func(self: ParserState*) → FuncDecl*?

  Parse an extern function declaration.
- func ps_parse_struct(self: ParserState*) → StructDecl*?

  Parse a struct declaration.
- func ps_parse_enum(self: ParserState*) → EnumDecl*?

  Parse an enum declaration.
- func ps_parse_type_alias(self: ParserState*) → TypeAliasDecl*?

  Parse a type-alias declaration.
- func ps_parse_top_level_let(self: ParserState*) → LetDecl*?

  Parse a top-level `let` declaration.
- func ps_parse_top_level_decl(self: ParserState*) → TopLevelDecl*?

  Parse one top-level declaration.
- func ps_parse_module(self: ParserState*) → Module*?

  Parse a module declaration and its contents.
- func impl_parse_module_tokens(tokens: TokenVector, filename: string) → ImplParseResult*

  Parse a module from a token stream and keep the internal arenas.
- func ps_lex_error_result(ls: LexerState*, filename: string) → ImplParseResult*

  Build a parser result from lexer diagnostics when tokenization fails.
- func ps_tokens_result(ls: LexerState*, toks: TokenVector, filename: string) → ImplParseResult*

  Parse tokenized source and clean up lexer/token resources.
- func impl_parse_module_source(source: string, filename: string) → ImplParseResult*

  Lex and parse a module from source text, preserving internal arenas.

8.68.1 Function Documentation

8.68.1.1 impl_parse_module_source()

```
func impl_parse_module_source(source:  string, filename:  string) → ImplParseResult*
```

Lex and parse a module from source text, preserving internal arenas.

Parameters

source	Source text to parse.
filename	Source filename used in diagnostics.

Returns

Internal parse result.

8.68.1.2 impl_parse_module_tokens()

```
func impl_parse_module_tokens(tokens:  TokenVector, filename:  string) → ImplParseResult*
```

Parse a module from a token stream and keep the internal arenas.

Parameters

tokens	Token stream to parse.
filename	Source filename used in diagnostics.

Returns

Internal parse result.

8.68.1.3 ps_finish_function()

```
func ps_finish_function(self: ParserState*, start: Span, name_tok: Token, params: VectorBase*,
is_extern: bool, ret_type: TypeRef*) → FuncDecl*?
```

Finish parsing a function declaration after its signature.

Parameters

self	Parser state to advance.
start	Starting span for the declaration.
name_tok	Function name token.
params	Parsed parameter vector.
is_extern	Whether this is an extern declaration.
ret_type	Parsed return type.

Returns

Completed function declaration, or `null` on error.

8.68.1.4 ps_lex_error_result()

```
func ps_lex_error_result(ls: LexerState*, filename: string) → ImplParseResult*
```

Build a parser result from lexer diagnostics when tokenization fails.

Parameters

ls	Lexer state holding diagnostics.
filename	Source filename used for fallback diagnostics.

Returns

Internal parse result containing copied lexer diagnostics.

8.68.1.5 ps_make_empty_block_stmt()

```
func ps_make_empty_block_stmt(self: ParserState*, span: Span) → StmtId
```

Create an empty block statement node in the statement arena.

Parameters

self	Parser state providing the statement arena.
span	Source span for the synthetic block.

Returns

Statement id for the created block.

8.68.1.6 ps_parse_enum()

```
func ps_parse_enum(self: ParserState*) → EnumDecl*?
```

Parse an enum declaration.

Parameters

self	Parser state to advance.

Returns

Parsed enum declaration, or `null` on error.

8.68.1.7 ps_parse_extern_func()

```
func ps_parse_extern_func(self: ParserState*) → FuncDecl*?
```

Parse an extern function declaration.

Parameters

self	Parser state to advance.

Returns

Parsed extern function declaration, or `null` on error.

8.68.1.8 ps_parse_function()

```
func ps_parse_function(self: ParserState*, is_extern: bool) → FuncDecl*?
```

Parse a function declaration.

Parameters

self	Parser state to advance.
is_extern	Whether the declaration started with `extern`.

Returns

Parsed function declaration, or `null` on error.

8.68.1.9 ps_parse_module()

```
func ps_parse_module(self:  ParserState*) → Module*?
```

Parse a module declaration and its contents.

Parameters

self	Parser state to advance.

Returns

Parsed module, or `null` on error.

8.68.1.10 ps_parse_struct()

```
func ps_parse_struct(self:  ParserState*) → StructDecl*?
```

Parse a struct declaration.

Parameters

self	Parser state to advance.

Returns

Parsed struct declaration, or `null` on error.

8.68.1.11 ps_parse_top_level_decl()

```
func ps_parse_top_level_decl(self:  ParserState*) → TopLevelDecl*?
```

Parse one top-level declaration.

Parameters

self	Parser state to advance.

Returns

Parsed top-level declaration, or `null` on error.

8.68.1.12 ps_parse_top_level_let()

```
func ps_parse_top_level_let(self:  ParserState*) → LetDecl*?
```

Parse a top-level `let` declaration.

Parameters

self	Parser state to advance.

Returns

Parsed top-level `let`, or `null` on error.

8.68.1.13 ps_parse_type_alias()

```
func ps_parse_type_alias(self:  ParserState*) → TypeAliasDecl*?
```

Parse a type-alias declaration.

Parameters

self	Parser state to advance.

Returns

Parsed type-alias declaration, or `null` on error.

8.68.1.14 ps_tokens_result()

```
func ps_tokens_result(ls:  LexerState*, toks:  TokenVector, filename:  string) → ImplParseResult*
```

Parse tokenized source and clean up lexer/token resources.

Parameters

ls	Lexer state to free after parsing.
toks	Token stream to parse and then free.
filename	Source filename used in diagnostics.

Returns

Internal parse result.

8.69 compiler/stage2_l0/src/parser/expr.l0 File Reference

Module: `parser.expr`

```
import parser.shared;
import ast;
import tokens;
import std.text;
import std.vector;
import std.string;
```

Functions

- func ps_is_builtin_type_name(self: ParserState*) → bool
  Check whether the current token is a builtin type name.
- func ps_is_unambiguous_type_start(self: ParserState*) → bool
  Check whether the current token sequence can only start a type.
- func ps_parse_call_argument(self: ParserState*) → ExprId?
  Parse one call argument.
- func ps_parse_expr(self: ParserState*) → ExprId?
  Parse an expression.
- func ps_parse_or_expr(self: ParserState*) → ExprId?
  Parse a logical-or expression.
- func ps_parse_and_expr(self: ParserState*) → ExprId?
  Parse a logical-and expression.
- func ps_parse_equality_expr(self: ParserState*) → ExprId?
  Parse an equality expression.
- func ps_parse_rel_expr(self: ParserState*) → ExprId?
  Parse a relational expression.
- func ps_parse_add_expr(self: ParserState*) → ExprId?
  Parse additive expressions.
- func ps_parse_mul_expr(self: ParserState*) → ExprId?
  Parse a multiplicative expression.
- func ps_parse_unary_expr(self: ParserState*) → ExprId?
  Parse a unary expression.
- func ps_parse_cast_expr(self: ParserState*) → ExprId?
  Parse a cast expression.
- func ps_parse_postfix_expr(self: ParserState*) → ExprId?
  Parse a postfix expression.
- func ps_parse_primary_expr(self: ParserState*) → ExprId?
  Parse a primary expression.

8.69.1 Function Documentation

8.69.1.1 ps_is_builtin_type_name()

```
func ps_is_builtin_type_name(self: ParserState*) → bool
```

Check whether the current token is a builtin type name.

Parameters

self	Parser state to inspect.

Returns

`true` if the current token names a builtin type.

8.69.1.2 ps_is_unambiguous_type_start()

```
func ps_is_unambiguous_type_start(self:  ParserState*) → bool
```

Check whether the current token sequence can only start a type.

Parameters

self	Parser state to inspect.

Returns

`true` if the lookahead unambiguously starts a type reference.

8.69.1.3 ps_parse_add_expr()

```
func ps_parse_add_expr(self:  ParserState*) → ExprId?
```

Parse additive expressions.

Parameters

self	Parser state to advance.

Returns

The parsed additive expression, or `null` on error.

8.69.1.4 ps_parse_and_expr()

```
func ps_parse_and_expr(self:  ParserState*) → ExprId?
```

Parse a logical-and expression.

Parameters

self	Parser state to advance.

Returns

Parsed logical-and expression, or `null` on error.

8.69.1.5 ps_parse_call_argument()

```
func ps_parse_call_argument(self:  ParserState*) → ExprId?
```

Parse one call argument.

Parameters

self	Parser state to advance.

Returns

Parsed argument expression, or `null` on error.

8.69.1.6 ps_parse_cast_expr()

```
func ps_parse_cast_expr(self:  ParserState*) → ExprId?
```

Parse a cast expression.

Parameters

self	Parser state to advance.

Returns

Parsed cast expression, or `null` on error.

8.69.1.7 ps_parse_equality_expr()

```
func ps_parse_equality_expr(self:  ParserState*) → ExprId?
```

Parse an equality expression.

Parameters

self	Parser state to advance.

Returns

Parsed equality expression, or `null` on error.

8.69.1.8 ps_parse_expr()

```
func ps_parse_expr(self:  ParserState*) → ExprId?
```

Parse an expression.

Parameters

self	Parser state to advance.

Returns

Parsed expression, or `null` on error.

8.69.1.9 ps_parse_mul_expr()

```
func ps_parse_mul_expr(self:  ParserState*) → ExprId?
```

Parse a multiplicative expression.

Parameters

self	Parser state to advance.

Returns

Parsed multiplicative expression, or `null` on error.

8.69.1.10 ps_parse_or_expr()

```
func ps_parse_or_expr(self:  ParserState*) → ExprId?
```

Parse a logical-or expression.

Parameters

self	Parser state to advance.

Returns

Parsed logical-or expression, or `null` on error.

8.69.1.11 ps_parse_postfix_expr()

```
func ps_parse_postfix_expr(self:  ParserState*) → ExprId?
```

Parse a postfix expression.

Parameters

self	Parser state to advance.

Returns

Parsed postfix expression, or `null` on error.

8.69.1.12 ps_parse_primary_expr()

```
func ps_parse_primary_expr(self:  ParserState*) → ExprId?
```

Parse a primary expression.

Parameters

self	Parser state to advance.

Returns

Parsed primary expression, or `null` on error.

8.69.1.13 ps_parse_rel_expr()

```
func ps_parse_rel_expr(self:  ParserState*) → ExprId?
```

Parse a relational expression.

Parameters

self	Parser state to advance.

Returns

Parsed relational expression, or `null` on error.

8.69.1.14 ps_parse_unary_expr()

```
func ps_parse_unary_expr(self:  ParserState*) → ExprId?
```

Parse a unary expression.

Parameters

self	Parser state to advance.

Returns

Parsed unary expression, or `null` on error.

8.70 compiler/stage2_l0/src/parser/shared.l0 File Reference

Module: `parser.shared`

```
import tokens;
import util.intset;
import util.diag;
import ast;
import std.vector;
import std.text;
import std.unit;
import std.string;
```

Classes

- struct ImplParseResult

  Represents the ImplParseResult structure.
- struct ParserState

  Represents the ParserState structure.
- struct QualifiedNameResult

  Represents the QualifiedNameResult structure.

Functions

- func ps_create(tokens: TokenVector, filename: string) → ParserState∗

  Create a parser state for a token stream and source file.
- func ps_destroy(self: ParserState∗)

  Release a parser state and its owned data.
- func ps_emit_error(self: ParserState∗, code: string, message: string, tok: Token)

  Emit a parser diagnostic anchored to a token.
- func ps_has_error(self: ParserState∗) → bool

  Report whether the parser has recorded any errors.
- func tok_is(tok: Token, kind_ord: int) → bool

  Check whether a token has the given kind ordinal.
- func tok_is_ident(tok: Token) → bool

  Check whether a token is an identifier.
- func tok_is_int(tok: Token) → bool

  Check whether a token is an integer literal.
- func tok_is_byte(tok: Token) → bool

  Check whether a token is a byte literal.
- func tok_is_string(tok: Token) → bool

  Check whether a token is a string literal.
- func tok_is_future_extension(tok: Token) → bool

  Check whether a token is reserved for a future language extension.
- func ps_peek(self: ParserState∗) → Token

Return the current lookahead token.

- func ps_skip_lexer_errors(self: ParserState*)

  Emit and step over any TT_LEXER_ERROR tokens at the cursor.
- func ps_logical_token_at_cursor(self: ParserState*) → Token

  Return the logical current token and skip unrecoverable lexer wrappers.
- func ps_peek_logical_ahead(self: ParserState*, offset: int) → Token

  Return a logical lookahead token without moving the parser cursor.
- func ps_emit_remaining_lexer_errors(self: ParserState*)

  Emit deferred lexer diagnostics that parsing never reached.
- func ps_last(self: ParserState*) → Token

  Return the most recently consumed token.
- func ps_at_end(self: ParserState*) → bool

  Check whether the parser is positioned at EOF.
- func ps_advance(self: ParserState*) → Token

  Consume and return the current token.
- func ps_check(self: ParserState*, kind_ord: int) → bool

  Check whether the current token has the given kind ordinal.
- func ps_match(self: ParserState*, kind_ord: int) → bool

  Consume the current token if it has the given kind ordinal.
- func ps_match_any2(self: ParserState*, k1: int, k2: int) → bool

  Consume the current token if it matches either of two kinds.
- func ps_match_any3(self: ParserState*, k1: int, k2: int, k3: int) → bool

  Consume the current token if it matches any of three kinds.
- func ps_match_any4(self: ParserState*, k1: int, k2: int, k3: int, k4: int) → bool

  Consume the current token if it matches any of four kinds.
- func ps_check_ident(self: ParserState*) → bool

  Check whether the current token is an identifier.
- func ps_match_ident(self: ParserState*) → bool

  Consume the current token if it is an identifier.
- func ps_check_int(self: ParserState*) → bool

  Check whether the current token is an integer literal.
- func ps_match_int(self: ParserState*) → bool

  Consume the current token if it is an integer literal.
- func ps_check_byte(self: ParserState*) → bool

  Check whether the current token is a byte literal.
- func ps_match_byte(self: ParserState*) → bool

  Consume the current token if it is a byte literal.
- func ps_check_string(self: ParserState*) → bool

  Check whether the current token is a string literal.
- func ps_match_string(self: ParserState*) → bool

  Consume the current token if it is a string literal.
- func ps_is_top_level_start(self: ParserState*) → bool

  Check whether the current token can start a top-level declaration.
- func ps_sync_top_level(self: ParserState*)

  Recover after a parse error by advancing to the next top-level boundary.
- func ps_at_stmt_start(self: ParserState*) → bool

  Report whether the current token can start a statement.
- func ps_sync_stmt(self: ParserState*)

  Recover after a failed statement by advancing to the next statement boundary.
- func ps_expect(self: ParserState*, kind_ord: int, code: string, msg: string) → Token?

  Require the current token to have a specific kind.

- func ps_expect_ident(self: ParserState*, code: string, msg: string) → Token?

  Require the current token to be an identifier.
- func token_ident_text(tok: Token) → string?

  Extract identifier text from a token.
- func token_int_text(tok: Token) → string?

  Extract integer literal text from a token.
- func token_int_value(tok: Token) → int?

  Extract the parsed integer value from a token.
- func token_byte_text(tok: Token) → string?

  Extract byte literal text from a token.
- func token_byte_value(tok: Token) → byte?

  Extract byte literal numeric value from a token.
- func token_string_text(tok: Token) → string?

  Extract string literal text from a token.
- func token_string_value(tok: Token) → string?

  Extract decoded string-literal bytes from a token.
- func token_text_for_width(tok: Token) → string

  Return the source-like text used to measure a token's display width.
- func token_width(tok: Token) → int

  Compute the display width of a token for diagnostics.
- func ps_span_start(self: ParserState*) → Span

  Capture a zero-width span at the current token position.
- func ps_extend_span(self: ParserState*, start: Span) → Span

  Extend a starting span to the end of the last consumed token.
- func ps_expect_variable_name(self: ParserState*, code: string, msg: string) → Token?

  Require an identifier that is legal as a variable name.
- func parse_dotted_name_rest(self: ParserState*, first: string) → string?

  Parse the remaining components of a dotted name.
- func qn_result_free_all(self: QualifiedNameResult*)

  Free a qualified-name parse result and its owned vectors.
- func ps_try_parse_qualified_name(self: ParserState*) → QualifiedNameResult*?

  Parse a qualified name when the lookahead matches `Module.Path::  Name`.
- func ps_parse_type(self: ParserState*) → TypeRef*?

  Parse a type reference.
- func ps_check_reserved_binary_op(self: ParserState*) → Unit?

  Reject reserved binary operators at the current parse position.

8.70.1 Function Documentation

8.70.1.1 parse_dotted_name_rest()

```
func parse_dotted_name_rest(self:  ParserState*, first:  string) → string?
```

Parse the remaining components of a dotted name.

Parameters

self	Parser state to advance.
first	Already parsed first identifier component.

Returns

Full dotted name, or `null` on parse error.

8.70.1.2 ps_advance()

```
func ps_advance(self:  ParserState*) → Token
```

Consume and return the current token.

Parameters

self	Parser state to advance.

Returns

Token that was current before advancing.

8.70.1.3 ps_at_end()

```
func ps_at_end(self:  ParserState*) → bool
```

Check whether the parser is positioned at EOF.

Parameters

self	Parser state to inspect.

Returns

`true` if the current token is `TT_EOF`.

8.70.1.4 ps_at_stmt_start()

```
func ps_at_stmt_start(self:  ParserState*) → bool
```

Report whether the current token can start a statement.

Parameters

self	Parser state to inspect.

Returns

True if the current token is a statement-start keyword.

8.70.1.5 ps_check()

```
func ps_check(self:  ParserState*, kind_ord:  int) → bool
```

Check whether the current token has the given kind ordinal.

Parameters

self	Parser state to inspect.
kind_ord	Token kind ordinal to compare against.

Returns

`true` if the current token matches `kind_ord`.

8.70.1.6 ps_check_byte()

```
func ps_check_byte(self:  ParserState*) → bool
```

Check whether the current token is a byte literal.

Parameters

self	Parser state to inspect.

Returns

`true` if the current token is `TT_BYTE`.

8.70.1.7 ps_check_ident()

```
func ps_check_ident(self:  ParserState*) → bool
```

Check whether the current token is an identifier.

Parameters

self	Parser state to inspect.

Returns

`true` if the current token is `TT_IDENT`.

8.70.1.8 ps_check_int()

```
func ps_check_int(self:  ParserState*) → bool
```

Check whether the current token is an integer literal.

Parameters

self	Parser state to inspect.

Returns

`true` if the current token is `TT_INT`.

8.70.1.9 ps_check_reserved_binary_op()

```
func ps_check_reserved_binary_op(self: ParserState*) → Unit?
```

Reject reserved binary operators at the current parse position.

Parameters

self	Parser state to inspect.

Returns

`present()` if the token is allowed here, or `null` after emitting an error.

8.70.1.10 ps_check_string()

```
func ps_check_string(self: ParserState*) → bool
```

Check whether the current token is a string literal.

Parameters

self	Parser state to inspect.

Returns

`true` if the current token is `TT_STRING`.

8.70.1.11 ps_create()

```
func ps_create(tokens: TokenVector, filename: string) → ParserState*
```

Create a parser state for a token stream and source file.

Parameters

tokens	Token stream to parse.
filename	Source filename used in diagnostics.

Returns

A new parser state with fresh arenas and diagnostics.

8.70.1.12 ps_destroy()

```
func ps_destroy(self:  ParserState*)
```

Release a parser state and its owned data.

Parameters

self	Parser state to free.

8.70.1.13 ps_emit_error()

```
func ps_emit_error(self:  ParserState*, code:  string, message:  string, tok:  Token)
```

Emit a parser diagnostic anchored to a token.

Parameters

self	Parser state collecting diagnostics.
code	Diagnostic code.
message	Diagnostic message.
tok	Token whose span should be highlighted.

8.70.1.14 ps_emit_remaining_lexer_errors()

```
func ps_emit_remaining_lexer_errors(self:  ParserState*)
```

Emit deferred lexer diagnostics that parsing never reached.

Parameters

self	Parser state whose physical token stream should be scanned.

8.70.1.15 ps_expect()

```
func ps_expect(self:  ParserState*, kind_ord:  int, code:  string, msg:  string) → Token?
```

Require the current token to have a specific kind.

Parameters

self	Parser state to inspect.

Parameters

kind_ord	Required token kind ordinal.
code	Diagnostic code to emit on mismatch.
msg	Expectation message prefix used in the diagnostic.

Returns

Consumed token, or `null` after emitting an error if the token does not match.

8.70.1.16 ps_expect_ident()

```
func ps_expect_ident(self: ParserState*, code: string, msg: string) → Token?
```

Require the current token to be an identifier.

Parameters

self	Parser state to inspect.
code	Diagnostic code to emit on mismatch.
msg	Expectation message prefix used in the diagnostic.

Returns

Consumed identifier token, or `null` after emitting an error.

8.70.1.17 ps_expect_variable_name()

```
func ps_expect_variable_name(self: ParserState*, code: string, msg: string) → Token?
```

Require an identifier that is legal as a variable name.

Parameters

self	Parser state to inspect.
code	Diagnostic code to emit on mismatch.
msg	Expectation message prefix used in the diagnostic.

Returns

Variable-name token, or `null` after emitting an error.

8.70.1.18 ps_extend_span()

```
func ps_extend_span(self: ParserState*, start: Span) → Span
```

Extend a starting span to the end of the last consumed token.

Parameters

self	Parser state providing the current position.
start	Starting span captured before parsing the construct.

Returns

Span from `start` through the last consumed token.

8.70.1.19 ps_has_error()

```
func ps_has_error(self:  ParserState*) → bool
```

Report whether the parser has recorded any errors.

Parameters

self	Parser state to inspect.

Returns

`true` if parser diagnostics contain an error.

8.70.1.20 ps_is_top_level_start()

```
func ps_is_top_level_start(self:  ParserState*) → bool
```

Check whether the current token can start a top-level declaration.

Parameters

self	Parser state to inspect.

Returns

`true` if the current token starts a declaration accepted at module scope.

8.70.1.21 ps_last()

```
func ps_last(self:  ParserState*) → Token
```

Return the most recently consumed token.

Parameters

self	Parser state to inspect.

Returns

Last consumed token, or the first token if nothing has been consumed yet.

8.70.1.22 ps_logical_token_at_cursor()

```
func ps_logical_token_at_cursor(self: ParserState*) → Token
```

Return the logical current token and skip unrecoverable lexer wrappers.

Parameters

self	Parser state to inspect.

Returns

Logical current token.

8.70.1.23 ps_match()

```
func ps_match(self: ParserState*, kind_ord: int) → bool
```

Consume the current token if it has the given kind ordinal.

Parameters

self	Parser state to advance.
kind_ord	Token kind ordinal to match.

Returns

`true` if a matching token was consumed.

8.70.1.24 ps_match_any2()

```
func ps_match_any2(self: ParserState*, k1: int, k2: int) → bool
```

Consume the current token if it matches either of two kinds.

Parameters

self	Parser state to advance.
k1	First acceptable token kind ordinal.
k2	Second acceptable token kind ordinal.

Returns

`true` if one of the requested token kinds was consumed.

8.70.1.25 ps_match_any3()

```
func ps_match_any3(self:  ParserState*, k1:  int, k2:  int, k3:  int) → bool
```

Consume the current token if it matches any of three kinds.

Parameters

self	Parser state to advance.
k1	First acceptable token kind ordinal.
k2	Second acceptable token kind ordinal.
k3	Third acceptable token kind ordinal.

Returns

`true` if one of the requested token kinds was consumed.

8.70.1.26 ps_match_any4()

```
func ps_match_any4(self:  ParserState*, k1:  int, k2:  int, k3:  int, k4:  int) → bool
```

Consume the current token if it matches any of four kinds.

Parameters

self	Parser state to advance.
k1	First acceptable token kind ordinal.
k2	Second acceptable token kind ordinal.
k3	Third acceptable token kind ordinal.
k4	Fourth acceptable token kind ordinal.

Returns

`true` if one of the requested token kinds was consumed.

8.70.1.27 ps_match_byte()

```
func ps_match_byte(self:  ParserState*) → bool
```

Consume the current token if it is a byte literal.

Parameters

self	Parser state to advance.

Returns

`true` if a byte literal token was consumed.

8.70.1.28 ps_match_ident()

```
func ps_match_ident(self: ParserState*) → bool
```

Consume the current token if it is an identifier.

Parameters

self	Parser state to advance.

Returns

`true` if an identifier token was consumed.

8.70.1.29 ps_match_int()

```
func ps_match_int(self: ParserState*) → bool
```

Consume the current token if it is an integer literal.

Parameters

self	Parser state to advance.

Returns

`true` if an integer literal token was consumed.

8.70.1.30 ps_match_string()

```
func ps_match_string(self: ParserState*) → bool
```

Consume the current token if it is a string literal.

Parameters

self	Parser state to advance.

Returns

`true` if a string literal token was consumed.

8.70.1.31 ps_parse_type()

```
func ps_parse_type(self:  ParserState*) → TypeRef*?
```

Parse a type reference.

Parameters

self	Parser state to advance.

Returns

Parsed type reference, or `null` on parse error.

8.70.1.32 ps_peek()

```
func ps_peek(self:  ParserState*) → Token
```

Return the current lookahead token.

TT_LEXER_ERROR wrappers are interpreted as logical tokens. Wrappers with recovery are exposed as that token; wrappers without recovery are skipped.

Parameters

self	Parser state to inspect.

Returns

Token at the current parse position.

8.70.1.33 ps_peek_logical_ahead()

```
func ps_peek_logical_ahead(self:  ParserState*, offset:  int) → Token
```

Return a logical lookahead token without moving the parser cursor.

Parameters

self	Parser state to inspect.
offset	Logical token offset from the current cursor.

Returns

Logical lookahead token.

8.70.1.34 ps_skip_lexer_errors()

```
func ps_skip_lexer_errors(self:  ParserState*)
```

Emit and step over any TT_LEXER_ERROR tokens at the cursor.

Parameters

self	Parser state to advance.

8.70.1.35 ps_span_start()

```
func ps_span_start(self:  ParserState*) → Span
```

Capture a zero-width span at the current token position.

Parameters

self	Parser state to inspect.

Returns

Span starting and ending at the current token location.

8.70.1.36 ps_sync_stmt()

```
func ps_sync_stmt(self:  ParserState*)
```

Recover after a failed statement by advancing to the next statement boundary.

Skips to and consumes the next `;`, or stops before a `}`, a statement-start keyword, a top-level start, or end of input. This keeps a single in-body statement error from unwinding to the top-level loop.

Parameters

self	Parser state to resynchronize.

8.70.1.37 ps_sync_top_level()

```
func ps_sync_top_level(self:  ParserState*)
```

Recover after a parse error by advancing to the next top-level boundary.

Parameters

self	Parser state to resynchronize.

8.70.1.38 ps_try_parse_qualified_name()

```
func ps_try_parse_qualified_name(self: ParserState*) → QualifiedNameResult*?
```

Parse a qualified name when the lookahead matches `Module.Path::Name`.

Parameters

self	Parser state to inspect.

Returns

Parsed qualified-name components, or `null` if the lookahead is not a qualified name.

8.70.1.39 qn_result_free_all()

```
func qn_result_free_all(self: QualifiedNameResult*)
```

Free a qualified-name parse result and its owned vectors.

Parameters

self	Qualified-name result to free.

8.70.1.40 tok_is()

```
func tok_is(tok: Token, kind_ord: int) → bool
```

Check whether a token has the given kind ordinal.

Parameters

tok	Token to inspect.
kind_ord	Token kind ordinal to compare against.

Returns

`true` if `tok` has the requested kind.

8.70.1.41 tok_is_byte()

```
func tok_is_byte(tok: Token) → bool
```

Check whether a token is a byte literal.

Parameters

tok	Token to inspect.

Returns

`true` if `tok` is a `TT_BYTE`.

8.70.1.42 tok_is_future_extension()

```
func tok_is_future_extension(tok:  Token) → bool
```

Check whether a token is reserved for a future language extension.

Parameters

tok	Token to inspect.

Returns

`true` if `tok` is `TT_FUTURE_EXTENSION`.

8.70.1.43 tok_is_ident()

```
func tok_is_ident(tok:  Token) → bool
```

Check whether a token is an identifier.

Parameters

tok	Token to inspect.

Returns

`true` if `tok` is a `TT_IDENT`.

8.70.1.44 tok_is_int()

```
func tok_is_int(tok:  Token) → bool
```

Check whether a token is an integer literal.

Parameters

tok	Token to inspect.

Returns

`true` if `tok` is a `TT_INT`.

8.70.1.45 tok_is_string()

```
func tok_is_string(tok:  Token) → bool
```

Check whether a token is a string literal.

Parameters

tok	Token to inspect.

Returns

`true` if `tok` is a `TT_STRING`.

8.70.1.46 token_byte_text()

```
func token_byte_text(tok:  Token) → string?
```

Extract byte literal text from a token.

Parameters

tok	Token to inspect.

Returns

Byte literal spelling, or `null` if `tok` is not `TT_BYTE`.

8.70.1.47 token_byte_value()

```
func token_byte_value(tok:  Token) → byte?
```

Extract byte literal numeric value from a token.

Parameters

tok	Token to inspect.

Returns

Byte literal value, or `null` if `tok` is not `TT_BYTE`.

8.70.1.48 token_ident_text()

```
func token_ident_text(tok:  Token) → string?
```

Extract identifier text from a token.

Parameters

tok	Token to inspect.

Returns

Identifier spelling, or `null` if `tok` is not `TT_IDENT`.

8.70.1.49 token_int_text()

```
func token_int_text(tok:  Token) → string?
```

Extract integer literal text from a token.

Parameters

tok	Token to inspect.

Returns

Integer literal spelling, or `null` if `tok` is not `TT_INT`.

8.70.1.50 token_int_value()

```
func token_int_value(tok:  Token) → int?
```

Extract the parsed integer value from a token.

Parameters

tok	Token to inspect.

Returns

Parsed integer value, or `null` if `tok` is not `TT_INT`.

8.70.1.51 token_string_text()

```
func token_string_text(tok:  Token) → string?
```

Extract string literal text from a token.

Parameters

tok	Token to inspect.

Returns

String literal spelling, or `null` if `tok` is not `TT_STRING`.

8.70.1.52 token_string_value()

```
func token_string_value(tok:  Token) → string?
```

Extract decoded string-literal bytes from a token.

Parameters

tok	Token to inspect.

Returns

Decoded string bytes, or `null` if `tok` is not `TT_STRING`.

8.70.1.53 token_text_for_width()

```
func token_text_for_width(tok:  Token) → string
```

Return the source-like text used to measure a token's display width.

Parameters

tok	Token to inspect.

Returns

Source spelling or token label used for width calculations.

8.70.1.54 token_width()

```
func token_width(tok:  Token) → int
```

Compute the display width of a token for diagnostics.

Parameters

tok	Token to measure.

Returns

Width of the token text in bytes.

8.71 compiler/stage2_l0/src/parser/stmt.l0 File Reference

Module: `parser.stmt`

```
import parser.shared;
import ast;
import parser.expr;
import tokens;
import std.vector;
import std.text;
import std.string;
```

Functions

- func ps_parse_pattern(self: ParserState∗) → PatternId?
  Parse a match pattern.
- func ps_parse_block(self: ParserState∗) → StmtId?
  Parse a block statement.
- func ps_parse_break_stmt(self: ParserState∗) → StmtId?
  Parse a `break` statement.
- func ps_parse_continue_stmt(self: ParserState∗) → StmtId?
  Parse a `continue` statement.
- func ps_parse_return_stmt(self: ParserState∗) → StmtId?
  Parse a `return` statement.
- func ps_parse_drop_stmt(self: ParserState∗) → StmtId?
  Parse a `drop` statement.
- func ps_parse_let_stmt(self: ParserState∗) → StmtId?
  Parse a local `let` statement.
- func ps_parse_if_stmt(self: ParserState∗) → StmtId?
  Parse an `if` statement.
- func ps_parse_while_stmt(self: ParserState∗) → StmtId?
  Parse a `while` statement.
- func ps_parse_for_stmt(self: ParserState∗) → StmtId?
  Parse a `for` statement.
- func ps_parse_for_update_stmt(self: ParserState∗) → StmtId?
  Parse a statement permitted in a `for` update clause.
- func ps_parse_match_stmt(self: ParserState∗) → StmtId?
  Parse a `match` statement.
- func ps_parse_with_item(self: ParserState∗) → WithItem∗?
  Parse one item inside a `with` header.

- func ps_parse_with_stmt(self: ParserState*) → StmtId?

  Parse a `with` statement.
- func ps_parse_case_literal(self: ParserState*) → ExprId?

  Parse a literal allowed in a `case` arm.
- func ps_at_case_arm_start(self: ParserState*) → bool

  Report whether the current token can start a `case` arm.
- func ps_sync_case_invalid_arm(self: ParserState*)

  Skip one malformed or forbidden `case` arm.
- func ps_parse_case_stmt(self: ParserState*) → StmtId?

  Parse a `case` statement.
- func ps_parse_simple_stmt(self: ParserState*) → StmtId?

  Parse a statement that does not consume its trailing semicolon.
- func ps_parse_stmt(self: ParserState*) → StmtId?

  Parse one statement.

8.71.1 Function Documentation

8.71.1.1 ps_at_case_arm_start()

```
func ps_at_case_arm_start(self:  ParserState*) → bool
```

Report whether the current token can start a `case` arm.

Parameters

self	Parser state to inspect.

Returns

True if the current token is a likely arm boundary.

8.71.1.2 ps_parse_block()

```
func ps_parse_block(self:  ParserState*) → StmtId?
```

Parse a block statement.

Parameters

self	Parser state to advance.

Returns

Parsed block statement id, or `null` on error.

8.71.1.3 ps_parse_break_stmt()

```
func ps_parse_break_stmt(self:  ParserState*) → StmtId?
```

Parse a `break` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `break` statement id, or `null` on error.

8.71.1.4 ps_parse_case_literal()

```
func ps_parse_case_literal(self:  ParserState*) → ExprId?
```

Parse a literal allowed in a `case` arm.

Parameters

self	Parser state to advance.

Returns

Parsed literal expression id, or `null` on error.

8.71.1.5 ps_parse_case_stmt()

```
func ps_parse_case_stmt(self:  ParserState*) → StmtId?
```

Parse a `case` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `case` statement id, or `null` on error.

8.71.1.6 ps_parse_continue_stmt()

```
func ps_parse_continue_stmt(self:  ParserState*) → StmtId?
```

Parse a `continue` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `continue` statement id, or `null` on error.

8.71.1.7 ps_parse_drop_stmt()

```
func ps_parse_drop_stmt(self:  ParserState*) → StmtId?
```

Parse a `drop` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `drop` statement id, or `null` on error.

8.71.1.8 ps_parse_for_stmt()

```
func ps_parse_for_stmt(self:  ParserState*) → StmtId?
```

Parse a `for` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `for` statement id, or `null` on error.

8.71.1.9 ps_parse_for_update_stmt()

```
func ps_parse_for_update_stmt(self:  ParserState*) → StmtId?
```

Parse a statement permitted in a `for` update clause.

Parameters

self	Parser state to advance.

Returns

Parsed update statement id, or `null` on error.

8.71.1.10 ps_parse_if_stmt()

```
func ps_parse_if_stmt(self: ParserState*) → StmtId?
```

Parse an `if` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `if` statement id, or `null` on error.

8.71.1.11 ps_parse_let_stmt()

```
func ps_parse_let_stmt(self: ParserState*) → StmtId?
```

Parse a local `let` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `let` statement id, or `null` on error.

8.71.1.12 ps_parse_match_stmt()

```
func ps_parse_match_stmt(self: ParserState*) → StmtId?
```

Parse a `match` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `match` statement id, or `null` on error.

8.71.1.13 ps_parse_pattern()

```
func ps_parse_pattern(self: ParserState*) → PatternId?
```

Parse a match pattern.

Parameters

self	Parser state to advance.

Returns

Parsed pattern, or `null` on error.

8.71.1.14 ps_parse_return_stmt()

```
func ps_parse_return_stmt(self:  ParserState*) → StmtId?
```

Parse a `return` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `return` statement id, or `null` on error.

8.71.1.15 ps_parse_simple_stmt()

```
func ps_parse_simple_stmt(self:  ParserState*) → StmtId?
```

Parse a statement that does not consume its trailing semicolon.

Parameters

self	Parser state to advance.

Returns

Parsed simple statement id, or `null` on error.

8.71.1.16 ps_parse_stmt()

```
func ps_parse_stmt(self:  ParserState*) → StmtId?
```

Parse one statement.

Parameters

self	Parser state to advance.

Returns

Parsed statement id, or `null` on error.

8.71.1.17 ps_parse_while_stmt()

```
func ps_parse_while_stmt(self:  ParserState*) → StmtId?
```

Parse a `while` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `while` statement id, or `null` on error.

8.71.1.18 ps_parse_with_item()

```
func ps_parse_with_item(self:  ParserState*) → WithItem*?
```

Parse one item inside a `with` header.

Parameters

self	Parser state to advance.

Returns

Parsed `with` item, or `null` on error.

8.71.1.19 ps_parse_with_stmt()

```
func ps_parse_with_stmt(self:  ParserState*) → StmtId?
```

Parse a `with` statement.

Parameters

self	Parser state to advance.

Returns

Parsed `with` statement id, or `null` on error.

8.71.1.20 ps_sync_case_invalid_arm()

```
func ps_sync_case_invalid_arm(self:  ParserState*)
```

Skip one malformed or forbidden `case` arm.

Starts at the offending arm token, consumes at least one token, and stops before the next likely arm or the enclosing `case` close.

Parameters

self	Parser state to resynchronize.

8.72 compiler/stage2_l0/src/scope_context.l0 File Reference

Module: `scope_context`

```
import types;
import ast;
import std.vector;
import std.string;
```

Classes

- struct ScopeVar

  Represents one declared variable tracked by the backend.
- struct ScopeContext

  Represents backend cleanup state for a single lexical scope.

Functions

- func sc_create(parent: ScopeContext*?) → ScopeContext*

  Create a backend scope context.
- func sc_var_free(self: ScopeVar*)

  Free a tracked variable record.
- func sc_inline_cleanup_free_opt(items: VectorBase*?)

  Free an optional inline-cleanup vector.
- func sc_free(self: ScopeContext*)

  Free one scope context and its owned containers.
- func sc_add_declared(self: ScopeContext*, name: string, var_type: Type*)

  Add a declared variable to a scope.
- func sc_add_owned(self: ScopeContext*, name: string, var_type: Type*)

  Add an owned variable to a scope and record it as declared.
- func sc_remove_owned(self: ScopeContext*, name: string)

  Remove a variable from the owned set.
- func sc_lookup_declared_type(self: ScopeContext*?, name: string) → Type*?

  Return the closest declared variable type from a scope chain.
- func sc_has_inline_cleanup(self: ScopeContext*) → bool

  Return whether a scope currently has inline cleanup items.
- func sc_push_inline_cleanup(self: ScopeContext*, stmt_id: StmtId)

  Lazily create and append an inline cleanup statement id.

8.72.1 Function Documentation

8.72.1.1 sc_add_declared()

```
func sc_add_declared(self:  ScopeContext*, name:  string, var_type:  Type*)
```

Add a declared variable to a scope.

Parameters

self	Scope to update.
name	Mangled C variable name.
var_type	Resolved L0 type.

8.72.1.2 sc_add_owned()

```
func sc_add_owned(self:  ScopeContext*, name:  string, var_type:  Type*)
```

Add an owned variable to a scope and record it as declared.

Parameters

self	Scope to update.
name	Mangled C variable name.
var_type	Resolved L0 type.

8.72.1.3 sc_create()

```
func sc_create(parent:  ScopeContext*?)  → ScopeContext*
```

Create a backend scope context.

Parameters

parent	Parent scope, or `null` for the function root.

Returns

New scope context.

8.72.1.4 sc_free()

```
func sc_free(self:  ScopeContext*)
```

Free one scope context and its owned containers.

Parameters

self	Scope context to free.

8.72.1.5 sc_has_inline_cleanup()

```
func sc_has_inline_cleanup(self:  ScopeContext*) → bool
```

Return whether a scope currently has inline cleanup items.

Parameters

self	Scope to inspect.

Returns

`true` if inline cleanup statements are recorded.

8.72.1.6 sc_inline_cleanup_free_opt()

```
func sc_inline_cleanup_free_opt(items:  VectorBase*?)
```

Free an optional inline-cleanup vector.

Parameters

items	Optional cleanup vector to free.

8.72.1.7 sc_lookup_declared_type()

```
func sc_lookup_declared_type(self:  ScopeContext*?, name:  string) → Type*?
```

Return the closest declared variable type from a scope chain.

Parameters

self	Starting scope, or `null`.
name	Mangled C variable name to find.

Returns

Borrowed type pointer, or `null` if absent.

8.72.1.8 sc_push_inline_cleanup()

```
func sc_push_inline_cleanup(self:  ScopeContext*, stmt_id:  StmtId)
```

Lazily create and append an inline cleanup statement id.

Parameters

self	Scope to update.
stmt↩ _id	Cleanup statement id to append.

8.72.1.9 sc_remove_owned()

```
func sc_remove_owned(self:  ScopeContext*, name:  string)
```

Remove a variable from the owned set.

Parameters

self	Scope to update.
name	Mangled C variable name to stop cleaning up.

8.72.1.10 sc_var_free()

```
func sc_var_free(self:  ScopeVar*)
```

Free a tracked variable record.

Parameters

self	Record to free.

8.73 compiler/stage2_l0/src/sem_context.l0 File Reference

Module: `sem_context`

```
import signatures;
import types;
import locals;
import ast;
import util.diag;
import driver;
import std.vector;
import std.text;
import name_resolver;
import std.hashmap;
```

Classes

- struct AnalysisResult

  Represents the AnalysisResult structure.

Enumerations

- enum VarRefResolution

  Defines the VarRefResolution enumeration.

Functions

- func analysis_owned_type_map_put(map: StringPtrMap∗, key: string, ty: Type∗)

  Replace one owned type-map entry with a clone of `ty`*.*
- func analysis_result_free(self: AnalysisResult∗)

  Free an analysis result and all owned semantic tables.
- func analysis_has_errors(self: AnalysisResult∗) → bool

  Report whether analysis produced any errors.
- func analysis_diag_count(self: AnalysisResult∗) → int

  Return the number of diagnostics produced by analysis.
- func analysis_diag_get(self: AnalysisResult∗, index: int) → Diagnostic∗

  Return a diagnostic from the analysis result.
- func analysis_source_names(self: AnalysisResult∗) → StringVector∗

  Return source filenames captured during driver analysis.
- func analysis_source_texts(self: AnalysisResult∗) → StringVector∗

  Return source texts captured during driver analysis.
- func analysis_name_resolution(self: AnalysisResult∗) → NameResolution∗?

  Return the name-resolution tables, if available.
- func analysis_signature_tables(self: AnalysisResult∗) → SignatureTables∗?

  Return the signature tables, if available.
- func analysis_locals(self: AnalysisResult∗) → LocalsResult∗?

  Return locals-resolution results, if available.
- func analysis_expr_key(module_name: string, expr_id: ExprId) → string

  Build the storage key used for per-expression analysis tables.
- func analysis_get_expr_type(self: AnalysisResult∗, module_name: string, expr_id: ExprId) → Type∗?

  Return a cloned resolved expression type when present.
- func analysis_get_intrinsic_target(self: AnalysisResult∗, module_name: string, expr_id: ExprId) → Type∗?

  Return a cloned intrinsic target type when present.
- func analysis_get_var_ref_resolution(self: AnalysisResult∗, module_name: string, expr_id: ExprId) → int?

  Return the stored variable-resolution kind when present.
- func analysis_is_arc_type(ty: Type∗) → bool

  Report whether a type is directly ARC-managed.
- func analysis_has_arc_data(self: AnalysisResult∗, ty: Type∗) → bool

  Report whether a type transitively contains ARC-managed data.

8.73.1 Enumeration Type Documentation

8.73.1.1 VarRefResolution

```
enum VarRefResolution
```

Defines the VarRefResolution enumeration.

Enumerator

VAR_RES_LOCAL	
VAR_RES_MODULE	

8.73.2 Function Documentation

8.73.2.1 analysis_diag_count()

```
func analysis_diag_count(self:  AnalysisResult*) → int
```

Return the number of diagnostics produced by analysis.

Parameters

self	Analysis result to inspect.

Returns

Total diagnostic count.

8.73.2.2 analysis_diag_get()

```
func analysis_diag_get(self:  AnalysisResult*, index:  int) → Diagnostic*
```

Return a diagnostic from the analysis result.

Parameters

self	Analysis result to inspect.
index	Zero-based diagnostic index.

Returns

Diagnostic at `index`.

8.73.2.3 analysis_expr_key()

```
func analysis_expr_key(module_name:  string, expr_id:  ExprId) → string
```

Build the storage key used for per-expression analysis tables.

Parameters

module_name	Owning module name.
expr_id	Expression id.

Returns

Key in `module::id` form.

8.73.2.4 analysis_get_expr_type()

```
func analysis_get_expr_type(self: AnalysisResult*, module_name: string, expr_id: ExprId) →
Type*?
```

Return a cloned resolved expression type when present.

Parameters

self	Analysis result to inspect.
module_name	Owning module name.
expr_id	Expression id.

Returns

Cloned expression type, or `null` when absent.

8.73.2.5 analysis_get_intrinsic_target()

```
func analysis_get_intrinsic_target(self: AnalysisResult*, module_name: string, expr_id:
ExprId) → Type*?
```

Return a cloned intrinsic target type when present.

Parameters

self	Analysis result to inspect.
module_name	Owning module name.
expr_id	Expression id.

Returns

Cloned target type, or `null` when absent.

8.73.2.6 analysis_get_var_ref_resolution()

```
func analysis_get_var_ref_resolution(self: AnalysisResult*, module_name: string, expr_id:
ExprId) → int?
```

Return the stored variable-resolution kind when present.

Parameters

self	Analysis result to inspect.
module_name	Owning module name.
expr_id	Expression id.

Returns

Stored resolution kind as an `int`, or `null` when absent.

8.73.2.7 analysis_has_arc_data()

```
func analysis_has_arc_data(self:  AnalysisResult*, ty:  Type*) → bool
```

Report whether a type transitively contains ARC-managed data.

Parameters

self	Analysis result providing resolved struct/enum metadata.
ty	Type to inspect.

Returns

`true` when values of `ty` need ARC cleanup.

8.73.2.8 analysis_has_errors()

```
func analysis_has_errors(self:  AnalysisResult*) → bool
```

Report whether analysis produced any errors.

Parameters

self	Analysis result to inspect.

Returns

`true` if the diagnostic collector contains an error.

8.73.2.9 analysis_is_arc_type()

```
func analysis_is_arc_type(ty:  Type*) → bool
```

Report whether a type is directly ARC-managed.

Parameters

ty	Type to inspect.

Returns

`true` when `ty` is a `string`.

8.73.2.10 analysis_locals()

```
func analysis_locals(self:  AnalysisResult*) → LocalsResult*?
```

Return locals-resolution results, if available.

Parameters

self	Analysis result to inspect.

Returns

Locals result, or `null` if analysis stopped earlier.

8.73.2.11 analysis_name_resolution()

```
func analysis_name_resolution(self:  AnalysisResult*) → NameResolution*?
```

Return the name-resolution tables, if available.

Parameters

self	Analysis result to inspect.

Returns

Name-resolution result, or `null` if analysis stopped earlier.

8.73.2.12 analysis_owned_type_map_put()

```
func analysis_owned_type_map_put(map:  StringPtrMap*, key:  string, ty:  Type*)
```

Replace one owned type-map entry with a clone of `ty`.

Analysis type maps own their values. Centralizing replacement here keeps insert and overwrite paths symmetric: an existing value is freed exactly once, while the caller retains ownership of `ty`.

Parameters

map	Owned `Type*` value map.
key	Entry key.
ty	Borrowed type to clone into the map.

8.73.2.13 analysis_result_free()

```
func analysis_result_free(self:  AnalysisResult*)
```

Free an analysis result and all owned semantic tables.

Parameters

self	Analysis result to free.

8.73.2.14 analysis_signature_tables()

```
func analysis_signature_tables(self:  AnalysisResult*) → SignatureTables*?
```

Return the signature tables, if available.

Parameters

self	Analysis result to inspect.

Returns

Signature tables, or `null` if analysis stopped earlier.

8.73.2.15 analysis_source_names()

```
func analysis_source_names(self:  AnalysisResult*) → StringVector*
```

Return source filenames captured during driver analysis.

Parameters

self	Analysis result to inspect.

Returns

Vector of source filenames.

8.73.2.16 analysis_source_texts()

```
func analysis_source_texts(self:  AnalysisResult*) → StringVector*
```

Return source texts captured during driver analysis.

Parameters

self	Analysis result to inspect.

Returns

Vector of source texts aligned with `analysis_source_names`.

8.74 compiler/stage2_l0/src/signatures.l0 File Reference

Module: `signatures`

```
import name_resolver;
import types;
import symbols;
import ast;
import type_resolve;
import driver;
import std.vector;
import std.text;
import util.diag;
import std.string;
import std.hashset;
import std.hashmap;
```

Functions

- func sig_key(module_name: string, name: string) → string

  Build a fully qualified signature-table key.
- func sig_top_decl_name(top: TopLevelDecl∗) → string

  Return the declared name carried by one top-level AST entry.
- func sig_top_decl_span(top: TopLevelDecl∗) → Span

  Return the source span carried by one top-level AST entry.
- func sig_diag_error(diags: DiagCollector∗, code: string, message: string, filename: string, span: Span)

  Emit a signature-resolution error for a source span.
- func sig_type_vec_free(items: VectorBase∗?)

  Free a vector of owned types.
- func sig_set_free_opt(self: StringSet∗?)

  Free an optional string set.
- func sig_type_vec_free_opt(items: VectorBase∗?)

  Free an optional vector of owned types.
- func sig_struct_field_vec_temp_free(items: VectorBase∗?)

  Free a temporary vector of struct-field metadata.
- func sig_enum_variant_vec_temp_free(items: VectorBase∗?)

  Free a temporary vector of enum-variant metadata.
- func sig_lookup_func_type(self: SignatureTables∗, module_name: string, name: string) → Type∗?

  Look up a resolved function type.
- func sig_lookup_struct_info(self: SignatureTables∗, module_name: string, name: string) → StructInfo∗?

  Look up resolved struct metadata.
- func sig_lookup_enum_info(self: SignatureTables∗, module_name: string, name: string) → EnumInfo∗?

  Look up resolved enum metadata.
- func sig_lookup_let_type(self: SignatureTables∗, module_name: string, name: string) → Type∗?

  Look up a resolved top-level `let` type.
- func sig_make_tables() → SignatureTables∗

  Create empty signature tables.
- func sig_resolve_struct(tables: SignatureTables∗, nr: NameResolution∗, env: ModuleEnv∗, decl: StructDecl∗, diags: DiagCollector∗)

  Resolve one struct declaration into stored signature metadata.

- func sig_resolve_enum(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*, decl: EnumDecl*, diags: DiagCollector*)

  Resolve one enum declaration into stored signature metadata.
- func sig_resolve_func(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*, decl: FuncDecl*, diags: DiagCollector*)

  Resolve one function declaration into a stored function type.
- func sig_infer_literal_type(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*, driver_state: DriverState*, expr_id: ExprId) → Type*?

  Infer a top-level literal initializer type without full expression checking.
- func sig_symbol_is_zero_field_enum_variant(sym: Symbol*) → bool

  Report whether one symbol names a zero-field enum variant.
- func sig_resolve_let(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*, driver_state: DriverState*, decl: LetDecl*, diags: DiagCollector*)

  Resolve or infer the type of one top-level `let` binding.
- func sig_extract_dependencies(ty: Type*, out: StringSet*)

  Collect value-type dependencies referenced by a type.
- func sig_cycle_visit(tables: SignatureTables*, key: string, visiting: StringSet*, visited: StringSet*, diags: DiagCollector*) → bool

  Visit one value-type node in the cycle-detection graph.
- func sig_detect_value_type_cycles(tables: SignatureTables*, diags: DiagCollector*)

  Detect value-type cycles across resolved structs and enums.
- func sig_resolve(driver_state: DriverState*, nr: NameResolution*, diags: DiagCollector*) → SignatureTables*

  Resolve top-level type signatures for parsed modules.

8.74.1 Function Documentation

8.74.1.1 sig_cycle_visit()

```
func sig_cycle_visit(tables: SignatureTables*, key: string, visiting: StringSet*, visited:
StringSet*, diags: DiagCollector*) → bool
```

Visit one value-type node in the cycle-detection graph.

Parameters

tables	Signature tables containing resolved struct and enum metadata.
key	Fully qualified type key to visit.
visiting	DFS stack set for cycle detection.
visited	Set of nodes already proven acyclic.
diags	Diagnostic collector for emitted cycle errors.

Returns

`true` if this visit reaches a value-type cycle.

8.74.1.2 sig_detect_value_type_cycles()

```
func sig_detect_value_type_cycles(tables: SignatureTables*, diags: DiagCollector*)
```

Detect value-type cycles across resolved structs and enums.

Parameters

tables	Signature tables to inspect.
diags	Diagnostic collector.

8.74.1.3 sig_diag_error()

```
func sig_diag_error(diags: DiagCollector*, code: string, message: string, filename: string,
span: Span)
```

Emit a signature-resolution error for a source span.

Parameters

diags	Diagnostic collector.
code	Diagnostic code.
message	Diagnostic message.
filename	Source filename.
span	Source span to highlight.

8.74.1.4 sig_enum_variant_vec_temp_free()

```
func sig_enum_variant_vec_temp_free(items: VectorBase*?)
```

Free a temporary vector of enum-variant metadata.

Parameters

items	Optional vector of `EnumVariantInfo*` values to free.

8.74.1.5 sig_extract_dependencies()

```
func sig_extract_dependencies(ty: Type*, out: StringSet*)
```

Collect value-type dependencies referenced by a type.

Parameters

ty	Type to inspect.
out	Set that receives fully qualified dependency keys.

8.74.1.6 sig_infer_literal_type()

```
func sig_infer_literal_type(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*,
driver_state: DriverState*, expr_id: ExprId) → Type*?
```

Infer a top-level literal initializer type without full expression checking.

Parameters

tables	Signature tables available during signature resolution.
nr	Name-resolution data for symbol lookups.
env	Module environment owning the initializer.
driver_state	Driver state providing parsed ASTs.
expr_id	Expression id to inspect.

Returns

Inferred literal type, or `null` when inference is not supported.

8.74.1.7 sig_key()

```
func sig_key(module_name:  string, name:  string) → string
```

Build a fully qualified signature-table key.

Parameters

module_name	Defining module name.
name	Symbol name.

Returns

Key in `module::name` form.

8.74.1.8 sig_lookup_enum_info()

```
func sig_lookup_enum_info(self:  SignatureTables*, module_name:  string, name:  string) →
EnumInfo*?
```

Look up resolved enum metadata.

Parameters

self	Signature tables to inspect.
module_name	Defining module name.
name	Enum name.

Returns

Enum info, or `null` if absent.

8.74.1.9 sig_lookup_func_type()

```
func sig_lookup_func_type(self: SignatureTables*, module_name: string, name: string) →
Type*?
```

Look up a resolved function type.

Parameters

self	Signature tables to inspect.
module_name	Defining module name.
name	Function name.

Returns

Function type, or `null` if absent.

8.74.1.10 sig_lookup_let_type()

```
func sig_lookup_let_type(self: SignatureTables*, module_name: string, name: string) → Type*?
```

Look up a resolved top-level `let` type.

Parameters

self	Signature tables to inspect.
module_name	Defining module name.
name	Binding name.

Returns

Resolved type, or `null` if absent.

8.74.1.11 sig_lookup_struct_info()

```
func sig_lookup_struct_info(self: SignatureTables*, module_name: string, name: string) →
StructInfo*?
```

Look up resolved struct metadata.

Parameters

self	Signature tables to inspect.
module_name	Defining module name.
name	Struct name.

Returns

Struct info, or `null` if absent.

8.74.1.12 sig_make_tables()

```
func sig_make_tables() → SignatureTables*
```

Create empty signature tables.

Returns

New signature tables.

8.74.1.13 sig_resolve()

```
func sig_resolve(driver_state: DriverState*, nr: NameResolution*, diags: DiagCollector*) →
SignatureTables*
```

Resolve top-level type signatures for parsed modules.

Parameters

driver_state	Parsed driver state.
nr	Name-resolution result.
diags	Diagnostic collector.

Returns

Populated signature tables.

8.74.1.14 sig_resolve_enum()

```
func sig_resolve_enum(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*, decl:
EnumDecl*, diags: DiagCollector*)
```

Resolve one enum declaration into stored signature metadata.

Parameters

tables	Signature tables receiving resolved enum info.
nr	Name-resolution data for symbol lookups.
env	Module environment owning the declaration.
decl	Enum declaration to resolve.
diags	Diagnostic collector for resolution failures.

8.74.1.15 sig_resolve_func()

```
func sig_resolve_func(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*, decl:
FuncDecl*, diags: DiagCollector*)
```

Resolve one function declaration into a stored function type.

Parameters

tables	Signature tables receiving resolved function types.
nr	Name-resolution data for symbol lookups.
env	Module environment owning the declaration.
decl	Function declaration to resolve.
diags	Diagnostic collector for resolution failures.

8.74.1.16 sig_resolve_let()

```
func sig_resolve_let(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*, driver_state:
DriverState*, decl: LetDecl*, diags: DiagCollector*)
```

Resolve or infer the type of one top-level `let` binding.

Parameters

tables	Signature tables receiving resolved `let` types.
nr	Name-resolution data for symbol lookups.
env	Module environment owning the declaration.
driver_state	Driver state providing parsed ASTs.
decl	`let` declaration to resolve.
diags	Diagnostic collector for resolution failures.

8.74.1.17 sig_resolve_struct()

```
func sig_resolve_struct(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*,
decl: StructDecl*, diags: DiagCollector*)
```

Resolve one struct declaration into stored signature metadata.

Parameters

tables	Signature tables receiving resolved struct info.
nr	Name-resolution data for symbol lookups.
env	Module environment owning the declaration.
decl	Struct declaration to resolve.
diags	Diagnostic collector for resolution failures.

8.74.1.18 sig_set_free_opt()

```
func sig_set_free_opt(self:  StringSet*?)
```

Free an optional string set.

Parameters

self	Optional string set to free.

8.74.1.19 sig_struct_field_vec_temp_free()

```
func sig_struct_field_vec_temp_free(items:  VectorBase*?)
```

Free a temporary vector of struct-field metadata.

Parameters

items	Optional vector of `StructFieldInfo*` values to free.

8.74.1.20 sig_symbol_is_zero_field_enum_variant()

```
func sig_symbol_is_zero_field_enum_variant(sym:  Symbol*) → bool
```

Report whether one symbol names a zero-field enum variant.

Parameters

sym	Symbol to inspect.

Returns

`true` when `sym` is a zero-field enum variant.

8.74.1.21 sig_top_decl_name()

```
func sig_top_decl_name(top:  TopLevelDecl*) → string
```

Return the declared name carried by one top-level AST entry.

Parameters

top	Top-level declaration to inspect.

Returns

Declaration name.

8.74.1.22 sig_top_decl_span()

```
func sig_top_decl_span(top:  TopLevelDecl*) → Span
```

Return the source span carried by one top-level AST entry.

Parameters

top	Top-level declaration to inspect.

Returns

Declaration span.

8.74.1.23 sig_type_vec_free()

```
func sig_type_vec_free(items:  VectorBase*?)
```

Free a vector of owned types.

Parameters

items	Optional vector of `Type*` values to free.

8.74.1.24 sig_type_vec_free_opt()

```
func sig_type_vec_free_opt(items:  VectorBase*?)
```

Free an optional vector of owned types.

Parameters

items	Optional vector of `Type*` values to free.

8.75 compiler/stage2_l0/src/source_paths.l0 File Reference

Module: `source_paths`

```
import util.strings;
import util.diag;
```

```
import std.vector;
import std.text;
import std.system;
import std.path;
import std.string;
import std.fs;
import sys.rt;
```

Classes

- struct SourceSearchPaths

  Represents the SourceSearchPaths structure.

Functions

- func sp_create() → SourceSearchPaths*

  Create an empty source search-path collection.
- func sp_free(self: SourceSearchPaths*)

  Free a source search-path collection.
- func sp_add_system_root(self: SourceSearchPaths*, root: string)

  Append a system root directory.
- func sp_add_project_root(self: SourceSearchPaths*, root: string)

  Append a project root directory.
- func sp_system_root_count(self: SourceSearchPaths*) → int

  Return the number of configured system roots.
- func sp_project_root_count(self: SourceSearchPaths*) → int

  Return the number of configured project roots.
- func sp_system_root_get(self: SourceSearchPaths*, index: int) → string

  Return a system root by index.
- func sp_project_root_get(self: SourceSearchPaths*, index: int) → string

  Return a project root by index.
- func sp_format_root_list(roots: StringVector*) → string

  Format one root list using the Stage 1 verbose display contract.
- func sp_module_relpath(module_name: string) → string

  Convert a dotted module name to its relative `.l0` path.
- func sp_resolve(self: SourceSearchPaths*, module_name: string) → string?

  Resolve a module name against configured search roots.
- func sp_is_valid_ident_component(component: string) → bool

  Check whether a module-name component is a valid identifier.
- func sp_is_valid_module_name(module_name: string) → bool

  Check whether a dotted module name is valid.
- func sp_apply_default_project_roots(project_roots: StringVector*)

  Ensure a default project root exists.
- func sp_append_env_sys_roots(sys_roots: StringVector*, env_value: string)

  Append system roots from a split environment-style path string.
- func sp_apply_default_sys_roots_env(sys_roots: StringVector*, l0_system: string?, l0_home: string?)

  Populate default system roots from effective environment values when needed.
- func sp_apply_default_sys_roots(sys_roots: StringVector*)

  Populate default system roots from `L0_SYSTEM` when needed.
- func sp_build_from_roots(project_roots: StringVector*, sys_roots: StringVector*) → SourceSearchPaths*

  Build a search-path collection from explicit root vectors.
- func sp_normalize_entry_target(target: string, project_roots: StringVector*, diags: DiagCollector*) → string?

  Normalize a CLI entry target to a module name.

8.75.1 Function Documentation

8.75.1.1 sp_add_project_root()

```
func sp_add_project_root(self:  SourceSearchPaths*, root:  string)
```

Append a project root directory.

Parameters

self	Search paths to update.
root	Project root directory to search.

8.75.1.2 sp_add_system_root()

```
func sp_add_system_root(self:  SourceSearchPaths*, root:  string)
```

Append a system root directory.

Parameters

self	Search paths to update.
root	Root directory to search before project roots.

8.75.1.3 sp_append_env_sys_roots()

```
func sp_append_env_sys_roots(sys_roots:  StringVector*, env_value:  string)
```

Append system roots from a split environment-style path string.

Parameters

sys_roots	System-root vector to update.
env_value	Colon- or semicolon-separated root list.

8.75.1.4 sp_apply_default_project_roots()

```
func sp_apply_default_project_roots(project_roots:  StringVector*)
```

Ensure a default project root exists.

Parameters

project_roots	Project-root vector to update.

8.75.1.5 sp_apply_default_sys_roots()

```
func sp_apply_default_sys_roots(sys_roots:  StringVector*)
```

Populate default system roots from `L0_SYSTEM` when needed.

Parameters

sys_roots	System-root vector to update.

8.75.1.6 sp_apply_default_sys_roots_env()

```
func sp_apply_default_sys_roots_env(sys_roots:  StringVector*, l0_system:  string?, l0_home:
string?)
```

Populate default system roots from effective environment values when needed.

Parameters

sys_roots	System-root vector to update.
l0_system	Effective `L0_SYSTEM` value, if any.
l0_home	Effective `L0_HOME` value, if any.

8.75.1.7 sp_build_from_roots()

```
func sp_build_from_roots(project_roots:  StringVector*, sys_roots:  StringVector*) → SourceSearchPaths*
```

Build a search-path collection from explicit root vectors.

Parameters

project_roots	Project roots to copy.
sys_roots	System roots to copy.

Returns

New search-path collection containing both root sets.

8.75.1.8 sp_create()

```
func sp_create() → SourceSearchPaths*
```

Create an empty source search-path collection.

Returns

New search-path set with separate system and project root vectors.

8.75.1.9 sp_format_root_list()

```
func sp_format_root_list(roots:  StringVector*) → string
```

Format one root list using the Stage 1 verbose display contract.

Parameters

roots	Root vector to format.

Returns

Comma-joined quoted roots, or `<none>` when empty.

8.75.1.10 sp_free()

```
func sp_free(self:  SourceSearchPaths*)
```

Free a source search-path collection.

Parameters

self	Search paths to free.

8.75.1.11 sp_is_valid_ident_component()

```
func sp_is_valid_ident_component(component:  string) → bool
```

Check whether a module-name component is a valid identifier.

Parameters

component	Module-name component to validate.

Returns

`true` if the component matches identifier rules.

8.75.1.12 sp_is_valid_module_name()

```
func sp_is_valid_module_name(module_name:  string) → bool
```

Check whether a dotted module name is valid.

Parameters

module_name	Module name to validate.

Returns

`true` if every component is a valid identifier.

8.75.1.13 sp_module_relpath()

```
func sp_module_relpath(module_name: string) → string
```

Convert a dotted module name to its relative `.l0` path.

Parameters

module_name	Dotted module name such as `foo.bar`.

Returns

Relative source path such as `foo/bar.l0`.

8.75.1.14 sp_normalize_entry_target()

```
func sp_normalize_entry_target(target: string, project_roots: StringVector*, diags: DiagCollector*)
→ string?
```

Normalize a CLI entry target to a module name.

Parameters

target	CLI target, either a module name or a path-like input.
project_roots	Project-root vector to extend when `target` includes a parent path.
diags	Diagnostic collector for validation errors.

Returns

Normalized module name, or `null` if validation fails.

8.75.1.15 sp_project_root_count()

```
func sp_project_root_count(self: SourceSearchPaths*) → int
```

Return the number of configured project roots.

Parameters

self	Search paths to inspect.

Returns

Number of project root directories.

8.75.1.16 sp_project_root_get()

```
func sp_project_root_get(self:  SourceSearchPaths*, index:  int) → string
```

Return a project root by index.

Parameters

self	Search paths to inspect.
index	Zero-based project-root index.

Returns

Project root at `index`.

8.75.1.17 sp_resolve()

```
func sp_resolve(self:  SourceSearchPaths*, module_name:  string) → string?
```

Resolve a module name against configured search roots.

Parameters

self	Search paths to inspect.
module_name	Dotted module name to resolve.

Returns

First matching source path, or `null` if the module is not found.

8.75.1.18 sp_system_root_count()

```
func sp_system_root_count(self:  SourceSearchPaths*) → int
```

Return the number of configured system roots.

Parameters

self	Search paths to inspect.

Returns

Number of system root directories.

8.75.1.19 sp_system_root_get()

```
func sp_system_root_get(self:  SourceSearchPaths*, index:  int) → string
```

Return a system root by index.

Parameters

self	Search paths to inspect.
index	Zero-based system-root index.

Returns

System root at `index`.

8.76 compiler/stage2_l0/src/string_escape.l0 File Reference

Module: `string_escape`

```
import std.text;
import std.string;
```

Functions

- func se_append_octal_digit(cb: CharBuffer*, digit: int)

  Append one octal digit in the range `0..7`.
- func se_append_octal_escape(cb: CharBuffer*, b: byte)

  Append a byte using a fixed-width C octal escape.
- func se_is_direct_c_byte(b: byte) → bool

  Return whether a byte is printable ASCII that can be emitted directly in a C string.
- func se_is_trigraph_suffix(b: byte) → bool

  Return whether a byte can complete a historical C trigraph spelling.
- func se_encode_c_string_bytes(data: string) → string

  Encode decoded string bytes for inclusion in a C string literal body.

8.76.1 Function Documentation

8.76.1.1 se_append_octal_digit()

```
func se_append_octal_digit(cb:  CharBuffer*, digit:  int)
```

Append one octal digit in the range `0..7`.

Parameters

cb	Character buffer to extend.
digit	Octal digit value.

8.76.1.2 se_append_octal_escape()

```
func se_append_octal_escape(cb:  CharBuffer*, b:  byte)
```

Append a byte using a fixed-width C octal escape.

Parameters

cb	Character buffer to extend.
b	Byte value to encode.

8.76.1.3 se_encode_c_string_bytes()

```
func se_encode_c_string_bytes(data:  string) → string
```

Encode decoded string bytes for inclusion in a C string literal body.

Parameters

data	Decoded byte string.

Returns

Escaped bytes suitable for inclusion between double quotes in C.

8.76.1.4 se_is_direct_c_byte()

```
func se_is_direct_c_byte(b:  byte) → bool
```

Return whether a byte is printable ASCII that can be emitted directly in a C string.

Parameters

b	Byte to inspect.

Returns

`true` for safe direct emission.

8.76.1.5 se_is_trigraph_suffix()

```
func se_is_trigraph_suffix(b:  byte) → bool
```

Return whether a byte can complete a historical C trigraph spelling.

Parameters

b	Byte to inspect.

Returns

`true` for the nine trigraph suffix bytes.

8.77 compiler/stage2_l0/src/symbols.l0 File Reference

Module: `symbols`

```
import types;
import ast;
```

Classes

- struct Symbol

  Represents the Symbol structure.

Enumerations

- enum SymbolKind

  Defines the SymbolKind enumeration.

Functions

- func symbol_create(name: string, kind: SymbolKind, module_name: string, owner_name: string, decl_ptr: void∗, span: Span) → Symbol∗

  Create a new symbol with the given properties.
- func symbol_set_type_clone(self: Symbol∗, resolved_type: Type∗)

  Replace a symbol's resolved type with a cloned copy.
- func symbol_kind_name(kind: SymbolKind) → string

  Return the printable name for a symbol kind.
- func symbol_free(self: Symbol∗)

  Free a symbol and any owned resolved type.

8.77.1 Enumeration Type Documentation

8.77.1.1 SymbolKind

```
enum SymbolKind
```

Defines the SymbolKind enumeration.

Enumerator

SYM_FUNC	
SYM_STRUCT	
SYM_ENUM	
SYM_ENUM_VARIANT	
SYM_TYPE_ALIAS	
SYM_LET	

8.77.2 Function Documentation

8.77.2.1 symbol_create()

```
func symbol_create(name: string, kind: SymbolKind, module_name: string, owner_name: string,
decl_ptr: void*, span: Span) → Symbol*
```

Create a new symbol with the given properties.

Parameters

name	Name of the symbol.
kind	Kind of the symbol.
module_name	Name of the module where the symbol is defined.
owner_name	Name of the owner (e.g., function or struct) of the symbol.
decl_ptr	Pointer to the AST node where the symbol is declared.
span	Source code span for error reporting.

Returns

A new Symbol instance with cloned string fields and null resolved type.

8.77.2.2 symbol_free()

```
func symbol_free(self: Symbol*)
```

Free a symbol and any owned resolved type.

Parameters

self	Symbol to free.

8.77.2.3 symbol_kind_name()

```
func symbol_kind_name(kind: SymbolKind) → string
```

Return the printable name for a symbol kind.

Parameters

kind	Symbol kind to format.

Returns

Uppercase display name for `kind`.

8.77.2.4 symbol_set_type_clone()

```
func symbol_set_type_clone(self:  Symbol*, resolved_type:  Type*)
```

Replace a symbol's resolved type with a cloned copy.

Parameters

self	Symbol to update.
resolved_type	Resolved type to clone into the symbol.

8.78 compiler/stage2_l0/src/tokens.l0 File Reference

Module: `tokens`

```
import std.vector;
import std.text;
import sys.rt;
import std.linear_map;
import std.unit;
import std.string;
import std.io;
```

Classes

- struct Token

  Represents the Token structure.
- struct TokenVector

  Represents the TokenVector structure.

Enumerations

- enum TokenRecovery

  Non-recursive logical token payload used by lexer-error wrappers.

- enum TokenType {
  TT_EOF , TT_UNDERSCORE =(text: string) , TT_MODULE =(text: string, value: string) , TT_IMPORT ,
  TT_FUNC , TT_STRUCT , TT_ENUM , TT_TYPE ,
  TT_EXTERN , TT_LET , TT_RETURN , TT_MATCH ,
  TT_CASE , TT_IF , TT_ELSE , TT_WHILE ,
  TT_FOR , TT_BREAK , TT_CONTINUE , TT_TRUE ,
  TT_FALSE , TT_NULL , TT_AS , TT_NEW ,
  TT_DROP , TT_WITH , TT_CLEANUP , TT_LBRACE ,
  TT_RBRACE , TT_LPAREN , TT_RPAREN , TT_LBRACKET ,
  TT_RBRACKET , TT_COMMA , TT_SEMI , TT_COLON ,
  TT_DOUBLE_COLON , TT_ARROW_FUNC , TT_ARROW_MATCH , TT_EQ ,
  TT_PLUS , TT_MINUS , TT_STAR , TT_SLASH ,
  TT_MODULO , TT_LT , TT_GT , TT_LE ,
  TT_GE , TT_EQEQ , TT_NE , TT_ANDAND ,
  TT_OROR , TT_BANG , TT_QUESTION , TT_DOT ,
  TT_AMP , TT_PIPE , TT_CARET , TT_TILDE ,
  TT_LSHIFT , TT_RSHIFT , TT_FUTURE_EXTENSION , TT_LEXER_ERROR =(code: string, message↩
  : string, line_end: int, column_end: int, recovery: TokenRecovery) }

  Defines the TokenType enumeration.

Functions

- func token_to_string(token: Token) → string

  Render a token as human-readable text.
- func token_recovery_to_string(recovery: TokenRecovery) → string

  Render a logical recovery payload.
- func token_recovery_is_some(recovery: TokenRecovery) → bool

  Return whether a lexer-error wrapper has a parser-visible recovery token.
- func token_recovered(wrapper: Token) → Token

  Build the logical token represented by a lexer-error recovery payload.
- func token_logical_predecessor_ends_expression(token: Token, previous: bool) → bool

  Return whether the parser-visible predecessor ends an expression.
- func is_reserved_ident_word(word: string) → bool

  Check whether identifier text is reserved as a builtin type name.
- func is_reserved_keyword(token: Token) → bool

  Check whether a token is reserved as a keyword or future extension.
- func ident_to_token(ident: string, index: int, line: int, col: int) → Token

  Convert identifier text into the appropriate token at a source location.
- func is_expr_ending_token_type(token_type: TokenType) → bool

  Check whether a token type can legally terminate an expression.
- func tv_create() → TokenVector

  Create an empty token vector.
- func tv_push(self: TokenVector, token: Token)

  Append a token to a token vector.
- func tv_get(self: TokenVector, index: int) → Token

  Return a token from a token vector.
- func tv_len(self: TokenVector) → int

  Return the number of tokens in a token vector.
- func token_release_payload(token: Token)

  Release ARC-managed payload strings held by a token.
- func token_recovery_release_payload(recovery: TokenRecovery)

  Release ARC-managed payload strings held by a token recovery payload.
- func tv_free(self: TokenVector)

  Free a token vector and release any token payload strings.

8.78.1 Enumeration Type Documentation

8.78.1.1 TokenRecovery

```
enum TokenRecovery
```

Non-recursive logical token payload used by lexer-error wrappers.

Variants:

- TR_NONE
- TR_INT(text: string, value: int)
- TR_BYTE(text: string, value: byte)
- TR_STRING(text: string, value: string)

8.78.1.2 TokenType

```
enum TokenType
```

Defines the TokenType enumeration.

8.78.2 Function Documentation

8.78.2.1 ident_to_token()

```
func ident_to_token(ident: string, index: int, line: int, col: int) → Token
```

Variants:

- TT_EOF
- TT_IDENT(text: string)
- TT_UNDERSCORE
- TT_INT(text: string, value: int)
- TT_BYTE(text: string, value: byte)
- TT_STRING(text: string, value: string)
- TT_MODULE
- TT_IMPORT
- TT_FUNC
- TT_STRUCT
- TT_ENUM

- TT_TYPE
- TT_EXTERN
- TT_LET
- TT_RETURN
- TT_MATCH
- TT_CASE
- TT_IF
- TT_ELSE
- TT_WHILE
- TT_FOR
- TT_BREAK
- TT_CONTINUE
- TT_TRUE
- TT_FALSE
- TT_NULL
- TT_AS
- TT_NEW
- TT_DROP
- TT_WITH
- TT_CLEANUP
- TT_LBRACE
- TT_RBRACE
- TT_LPAREN
- TT_RPAREN
- TT_LBRACKET
- TT_RBRACKET
- TT_COMMA
- TT_SEMI
- TT_COLON
- TT_DOUBLE_COLON
- TT_ARROW_FUNC
- TT_ARROW_MATCH
- TT_EQ
- TT_PLUS
- TT_MINUS
- TT_STAR

- TT_SLASH
- TT_MODULO
- TT_LT
- TT_GT
- TT_LE
- TT_GE
- TT_EQEQ
- TT_NE
- TT_ANDAND
- TT_OROR
- TT_BANG
- TT_QUESTION
- TT_DOT
- TT_AMP
- TT_PIPE
- TT_CARET
- TT_TILDE
- TT_LSHIFT
- TT_RSHIFT
- TT_FUTURE_EXTENSION
- TT_LEXER_ERROR(code: string, message: string, line_end: int, column_end: int, recovery: TokenRecovery)

Convert identifier text into the appropriate token at a source location.

Parameters

ident	Identifier text as it appeared in source.
index	Zero-based byte offset in the source.
line	One-based source line.
col	One-based source column.

Returns

Keyword, underscore, future-extension, or identifier token for `ident`.

8.78.2.2 is_expr_ending_token_type()

```
func is_expr_ending_token_type(token_type: TokenType) → bool
```

Check whether a token type can legally terminate an expression.

Parameters

token_type	Token type to inspect.

Returns

`true` if the token type can appear at the end of an expression.

8.78.2.3 is_reserved_ident_word()

```
func is_reserved_ident_word(word: string) → bool
```

Check whether identifier text is reserved as a builtin type name.

Parameters

word	Identifier text to inspect.

Returns

`true` if `word` is reserved and cannot be used as a plain identifier.

8.78.2.4 is_reserved_keyword()

```
func is_reserved_keyword(token: Token) → bool
```

Check whether a token is reserved as a keyword or future extension.

Parameters

token	Token to inspect.

Returns

`true` if the token cannot be used as a variable name.

8.78.2.5 token_logical_predecessor_ends_expression()

```
func token_logical_predecessor_ends_expression(token: Token, previous: bool) → bool
```

Return whether the parser-visible predecessor ends an expression.

Lexer-error wrappers with recovery contribute the recovered token context; wrappers without recovery preserve the preceding logical context.

Parameters

token	Newly scanned physical token.
previous	Whether the previous parser-visible token ended an expression.

Returns

Expression-ending context for signed-literal lexing.

8.78.2.6 token_recovered()

```
func token_recovered(wrapper:  Token) → Token
```

Build the logical token represented by a lexer-error recovery payload.

Parameters

wrapper	Physical lexer-error token.

Returns

Logical recovered token, or the wrapper when there is no recovery.

8.78.2.7 token_recovery_is_some()

```
func token_recovery_is_some(recovery:  TokenRecovery) → bool
```

Return whether a lexer-error wrapper has a parser-visible recovery token.

Parameters

recovery	Recovery payload to inspect.

Returns

`true` if the payload represents a logical token.

8.78.2.8 token_recovery_release_payload()

```
func token_recovery_release_payload(recovery:  TokenRecovery)
```

Release ARC-managed payload strings held by a token recovery payload.

Parameters

recovery	Recovery payload whose strings should be released.

8.78.2.9 token_recovery_to_string()

```
func token_recovery_to_string(recovery:  TokenRecovery) → string
```

Render a logical recovery payload.

Parameters

recovery	Recovery payload to format.

Returns

Human-readable recovery token spelling.

8.78.2.10 token_release_payload()

```
func token_release_payload(token:  Token)
```

Release ARC-managed payload strings held by a token.

Parameters

token	Token whose payload should be released.

8.78.2.11 token_to_string()

```
func token_to_string(token:  Token) → string
```

Render a token as human-readable text.

Parameters

token	Token to format.

Returns

Readable token spelling for diagnostics and debugging output.

8.78.2.12 tv_create()

```
func tv_create() → TokenVector
```

Create an empty token vector.

Returns

Token vector with an initial capacity.

8.78.2.13 tv_free()

```
func tv_free(self:  TokenVector)
```

Free a token vector and release any token payload strings.

Parameters

self	Token vector to free.

8.78.2.14 tv_get()

```
func tv_get(self:  TokenVector, index:  int) → Token
```

Return a token from a token vector.

Parameters

self	Token vector to inspect.
index	Zero-based token index.

Returns

Token at `index`.

8.78.2.15 tv_len()

```
func tv_len(self:  TokenVector) → int
```

Return the number of tokens in a token vector.

Parameters

self	Token vector to inspect.

Returns

Token count.

8.78.2.16 tv_push()

```
func tv_push(self:  TokenVector, token:  Token)
```

Append a token to a token vector.

Parameters

self	Token vector to extend.
token	Token value to append.

8.79 compiler/stage2_l0/src/type_resolve.l0 File Reference

Module: `type_resolve`

```
import types;
import driver;
import name_resolver;
import ast;
import util.diag;
import std.vector;
import std.text;
import symbols;
import std.string;
import std.hashset;
```

Functions

- func tr_key(module_name: string, name: string) → string

  Build a fully qualified type-resolution key.
- func tr_module_from_path(path: StringVector∗) → string

  Join a parsed module path into dotted form.
- func tr_type_ref_full_name(tref: TypeRef∗) → string

  Format the fully qualified display name for one parsed type reference.
- func tr_type_ref_simple_name(tref: TypeRef∗) → string

  Format the direct type name suggestion for one parsed type reference.
- func tr_ambiguous_type_modules(nr: NameResolution∗, env: ModuleEnv∗, name: string) → StringVector∗

  Collect imported module names contributing one ambiguous type name.
- func tr_format_ambiguous_modules(modules: StringVector∗) → string

  Format one quoted module list for ambiguous-type diagnostics.
- func tr_format_ambiguous_hints(modules: StringVector∗, name: string) → string

  Format the disambiguation hints for one ambiguous type name.
- func tr_diag_error(diags: DiagCollector∗, code: string, message: string, filename: string, span: Span)

  Emit a type-resolution diagnostic for a source span.
- func tr_symbol_lookup(nr: NameResolution∗, env: ModuleEnv∗, name: string, module_path: StringVector∗?, name_qualifier: StringVector∗?) → Symbol∗?

  Resolve a symbol visible from a module environment.
- func tr_resolve_alias(tables: SignatureTables∗, nr: NameResolution∗, env: ModuleEnv∗, sym: Symbol∗, diags: DiagCollector∗, seen: StringSet∗) → Type∗?

  Resolve a type alias symbol to its final target type.
- func tr_apply_suffix(base: Type∗, tref: TypeRef∗, diags: DiagCollector∗, filename: string) → Type∗?

  Apply pointer and nullable suffixes from a parsed type reference.
- func tr_resolve_type_ref(tables: SignatureTables∗, nr: NameResolution∗, env: ModuleEnv∗, tref: TypeRef∗, diags: DiagCollector∗, seen: StringSet∗) → Type∗?

  Resolve a parsed type reference into a semantic type descriptor.

8.79.1 Function Documentation

8.79.1.1 tr_ambiguous_type_modules()

```
func tr_ambiguous_type_modules(nr: NameResolution*, env: ModuleEnv*, name: string) → StringVector*
```

Collect imported module names contributing one ambiguous type name.

Parameters

nr	Name-resolution result containing all module environments.
env	Current module environment.
name	Ambiguous imported type name.

Returns

Sorted module names that provide the ambiguous type.

8.79.1.2 tr_apply_suffix()

```
func tr_apply_suffix(base: Type*, tref: TypeRef*, diags: DiagCollector*, filename: string)
→ Type*?
```

Apply pointer and nullable suffixes from a parsed type reference.

Parameters

base	Resolved base type.
tref	Parsed type reference supplying suffixes.
diags	Diagnostic collector.
filename	Source filename for diagnostics.

Returns

Suffixed type, or `null` if suffix validation fails.

8.79.1.3 tr_diag_error()

```
func tr_diag_error(diags: DiagCollector*, code: string, message: string, filename: string,
span: Span)
```

Emit a type-resolution diagnostic for a source span.

Parameters

diags	Diagnostic collector.
code	Diagnostic code.
message	Diagnostic message.
filename	Source filename.
span	Source span to highlight.

8.79.1.4 tr_format_ambiguous_hints()

```
func tr_format_ambiguous_hints(modules:  StringVector*, name:  string) → string
```

Format the disambiguation hints for one ambiguous type name.

Parameters

modules	Sorted module names contributing an ambiguity.
name	Ambiguous type name.

Returns

String like ‘'a::T’ or 'b::T'`.

8.79.1.5 tr_format_ambiguous_modules()

```
func tr_format_ambiguous_modules(modules:  StringVector*) → string
```

Format one quoted module list for ambiguous-type diagnostics.

Parameters

modules	Sorted module names contributing an ambiguity.

Returns

String like ‘a’, 'b`.

8.79.1.6 tr_key()

```
func tr_key(module_name:  string, name:  string) → string
```

Build a fully qualified type-resolution key.

Parameters

module_name	Defining module name.
name	Symbol name within the module.

Returns

Key in `module::name` form.

8.79.1.7 tr_module_from_path()

```
func tr_module_from_path(path:  StringVector*) → string
```

Join a parsed module path into dotted form.

Parameters

path	Vector of module-path components.

Returns

Dotted module name.

8.79.1.8 tr_resolve_alias()

```
func tr_resolve_alias(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*, sym:
Symbol*, diags: DiagCollector*, seen: StringSet*) → Type*?
```

Resolve a type alias symbol to its final target type.

Parameters

tables	Signature tables available during type resolution.
nr	Name-resolution data containing module environments.
env	Current module environment for diagnostics.
sym	Alias symbol to resolve.
diags	Diagnostic collector for resolution failures.
seen	Alias keys currently being expanded for cycle detection.

Returns

Resolved aliased type, or `null` on failure.

8.79.1.9 tr_resolve_type_ref()

```
func tr_resolve_type_ref(tables: SignatureTables*, nr: NameResolution*, env: ModuleEnv*,
tref: TypeRef*, diags: DiagCollector*, seen: StringSet*) → Type*?
```

Resolve a parsed type reference into a semantic type descriptor.

Parameters

tables	Signature tables available during type resolution.
nr	Name-resolution data containing module environments.
env	Current module environment for lookup and diagnostics.
tref	Parsed type reference to resolve.
diags	Diagnostic collector for resolution failures.
seen	Alias keys currently being expanded for cycle detection.

Returns

Resolved type, or `null` if resolution fails.

8.79.1.10 tr_symbol_lookup()

```
func tr_symbol_lookup(nr: NameResolution*, env: ModuleEnv*, name: string, module_path:
StringVector*?, name_qualifier: StringVector*?) → Symbol*?
```

Resolve a symbol visible from a module environment.

Parameters

nr	Name-resolution data containing module environments.
env	Current module environment.
name	Unqualified symbol name.
module_path	Optional parsed module path for cross-module lookup.
name_qualifier	Optional enum-variant qualifier.

Returns

Resolved symbol, or `null` if lookup fails.

8.79.1.11 tr_type_ref_full_name()

```
func tr_type_ref_full_name(tref: TypeRef*) → string
```

Format the fully qualified display name for one parsed type reference.

Parameters

tref	Parsed type reference to render.

Returns

Display string including module path and name qualifier segments.

8.79.1.12 tr_type_ref_simple_name()

```
func tr_type_ref_simple_name(tref: TypeRef*) → string
```

Format the direct type name suggestion for one parsed type reference.

Parameters

tref	Parsed type reference to simplify.

Returns

Display string omitting enum-style qualifier segments.

8.80 compiler/stage2_l0/src/types.l0 File Reference

Module: `types`

```
import ast;
import std.vector;
import std.text;
import std.string;
import std.hashmap;
```

Classes

- struct Type

  Represents the Type structure.
- struct StructFieldInfo

  Represents the StructFieldInfo structure.
- struct StructInfo

  Represents the StructInfo structure.
- struct EnumVariantInfo

  Represents the EnumVariantInfo structure.
- struct EnumInfo

  Represents the EnumInfo structure.
- struct SignatureTables

  Represents the SignatureTables structure.

Enumerations

- enum TypeKind

  Defines the TypeKind enumeration.

Functions

- func type_param_vec_free(items: VectorBase∗?)

  Free a vector of owned parameter types.
- func type_new(kind: TypeKind, module_name: string, name: string, inner: Type∗?, params: VectorBase∗?, result: Type∗?) → Type∗

  Create a type descriptor from raw components.
- func type_new_builtin(name: string) → Type∗

  Create a builtin type descriptor.
- func type_builtin_bool() → Type∗

  Create the builtin `bool` type.
- func type_builtin_byte() → Type∗

  Create the builtin `byte` type.
- func type_builtin_int() → Type∗

Create the builtin `int` type.

- func type_builtin_string() → Type∗

  Create the builtin `string` type.

- func type_builtin_void() → Type∗

  Create the builtin `void` type.

- func type_new_null() → Type∗

  Create the special `null` type.

- func type_new_struct(module_name: string, name: string) → Type∗

  Create a struct type reference.

- func type_new_enum(module_name: string, name: string) → Type∗

  Create an enum type reference.

- func type_new_pointer(inner: Type∗) → Type∗

  Wrap a type in a pointer type.

- func type_new_nullable(inner: Type∗) → Type∗

  Wrap a type in a nullable type.

- func type_new_func(params: VectorBase∗, result: Type∗) → Type∗

  Create a function type.

- func type_is_builtin(self: Type∗, name: string) → bool

  Check whether a type is a specific builtin.

- func type_is_builtin_name(name: string) → bool

  Check whether a type name is one of the builtin names.

- func type_clone(self: Type∗) → Type∗

  Deep-clone a type descriptor.

- func type_equals(a: Type∗?, b: Type∗?) → bool

  Compare two types for structural equality.

- func type_format(self: Type∗) → string

  Format a type as source-like text.

- func type_free(self: Type∗)

  Free a type descriptor and any owned child types.

- func struct_info_free(self: StructInfo∗)

  Free resolved struct metadata.

- func enum_info_free(self: EnumInfo∗)

  Free resolved enum metadata.

- func sig_tables_free(self: SignatureTables∗)

  Free signature tables and every owned resolved type/info entry.

8.80.1 Enumeration Type Documentation

8.80.1.1 TypeKind

```
enum TypeKind
```

Defines the TypeKind enumeration.

Enumerator

TY_BUILTIN	
TY_STRUCT	
TY_ENUM	
TY_POINTER	
TY_NULLABLE	
TY_FUNC	
TY_NULL	

8.80.2 Function Documentation

8.80.2.1 enum_info_free()

```
func enum_info_free(self:  EnumInfo*)
```

Free resolved enum metadata.

Parameters

self	Enum info to free.

8.80.2.2 sig_tables_free()

```
func sig_tables_free(self:  SignatureTables*)
```

Free signature tables and every owned resolved type/info entry.

Parameters

self	Signature tables to free.

8.80.2.3 struct_info_free()

```
func struct_info_free(self:  StructInfo*)
```

Free resolved struct metadata.

Parameters

self	Struct info to free.

8.80.2.4 type_builtin_bool()

```
func type_builtin_bool() → Type*
```

Create the builtin `bool` type.

Returns

New `bool` type.

8.80.2.5 type_builtin_byte()

```
func type_builtin_byte() → Type*
```

Create the builtin `byte` type.

Returns

New `byte` type.

8.80.2.6 type_builtin_int()

```
func type_builtin_int() → Type*
```

Create the builtin `int` type.

Returns

New `int` type.

8.80.2.7 type_builtin_string()

```
func type_builtin_string() → Type*
```

Create the builtin `string` type.

Returns

New `string` type.

8.80.2.8 type_builtin_void()

```
func type_builtin_void() → Type*
```

Create the builtin `void` type.

Returns

New `void` type.

8.80.2.9 type_clone()

```
func type_clone(self:  Type*) → Type*
```

Deep-clone a type descriptor.

Parameters

self	Type to clone.

Returns

Deep copy of `self`.

8.80.2.10 type_equals()

```
func type_equals(a:  Type*?, b:  Type*?)  → bool
```

Compare two types for structural equality.

Parameters

a	First type.
b	Second type.

Returns

`true` if the types are equal.

8.80.2.11 type_format()

```
func type_format(self:  Type*) → string
```

Format a type as source-like text.

Parameters

self	Type to format.

Returns

Human-readable type spelling.

8.80.2.12 type_free()

```
func type_free(self:  Type*)
```

Free a type descriptor and any owned child types.

Parameters

self	Type to free.

8.80.2.13 type_is_builtin()

```
func type_is_builtin(self: Type*, name: string) → bool
```

Check whether a type is a specific builtin.

Parameters

self	Type to inspect.
name	Builtin name to compare against.

Returns

`true` if `self` is the named builtin type.

8.80.2.14 type_is_builtin_name()

```
func type_is_builtin_name(name: string) → bool
```

Check whether a type name is one of the builtin names.

Parameters

name	Type name to inspect.

Returns

`true` if `name` names a builtin type.

8.80.2.15 type_new()

```
func type_new(kind: TypeKind, module_name: string, name: string, inner: Type*?, params:
VectorBase*?, result: Type*?) → Type*
```

Create a type descriptor from raw components.

Parameters

kind	Type kind tag.
module_name	Defining module name for nominal types.
name	Type name for builtin and nominal kinds.
inner	Owned inner type for pointer and nullable kinds.
params	Owned parameter vector for function kinds.
result	Owned result type for function kinds.

Returns

New type descriptor.

8.80.2.16 type_new_builtin()

```
func type_new_builtin(name:  string) → Type*
```

Create a builtin type descriptor.

Parameters

name	Builtin type name.

Returns

New builtin type.

8.80.2.17 type_new_enum()

```
func type_new_enum(module_name:  string, name:  string) → Type*
```

Create an enum type reference.

Parameters

module_name	Module defining the enum.
name	Enum name.

Returns

New enum type.

8.80.2.18 type_new_func()

```
func type_new_func(params:  VectorBase*, result:  Type*) → Type*
```

Create a function type.

Parameters

params	Owned vector of parameter types.
result	Result type.

Returns

New function type.

8.80.2.19 type_new_null()

```
func type_new_null() → Type*
```

Create the special `null` type.

Returns

New `null` type.

8.80.2.20 type_new_nullable()

```
func type_new_nullable(inner:  Type*) → Type*
```

Wrap a type in a nullable type.

Parameters

inner	Wrapped type.

Returns

New nullable type.

8.80.2.21 type_new_pointer()

```
func type_new_pointer(inner:  Type*) → Type*
```

Wrap a type in a pointer type.

Parameters

inner	Pointee type.

Returns

New pointer type.

8.80.2.22 type_new_struct()

```
func type_new_struct(module_name:  string, name:  string) → Type*
```

Create a struct type reference.

Parameters

module_name	Module defining the struct.
name	Struct name.

Returns

New struct type.

8.80.2.23 type_param_vec_free()

```
func type_param_vec_free(items:  VectorBase*?)
```

Free a vector of owned parameter types.

Parameters

items	Optional parameter-type vector to free.

8.81 compiler/stage2_l0/src/util/diag.l0 File Reference

Module: `util.diag`

```
import std.vector;
```

Classes

- struct Diagnostic

  Diagnostic represents an error, warning, or note message produced during compilation.
- struct DiagCollector

  Represents the DiagCollector structure.

Enumerations

- enum DiagnosticSeverity

  Defines the DiagnosticSeverity enumeration.

Functions

- type DiagnosticVector = VectorBase

  Create an empty vector of diagnostics.
- func dgv_push(self: DiagnosticVector*, diag: Diagnostic)

  Append a diagnostic to the vector.
- func dgv_get(self: DiagnosticVector*, index: int) → Diagnostic*

  Return a diagnostic from the vector.
- func dgv_length(self: DiagnosticVector*) → int

  Return the number of diagnostics stored in the vector.
- func dgv_free(self: DiagnosticVector*)

  Free a diagnostic vector and release ARC-owned fields in each entry.
- func diag_create() → DiagCollector*

  Create an empty diagnostic collector.
- func diag_error(dc: DiagCollector*, phase: string, code: string, message: string, filename: string, line: int, column: int, line_end: int, column_end: int)

  Record an error diagnostic.
- func diag_warn(dc: DiagCollector*, phase: string, code: string, message: string, filename: string, line: int, column: int, line_end: int, column_end: int)

  Record a warning diagnostic.
- func diag_note(dc: DiagCollector*, phase: string, code: string, message: string, filename: string, line: int, column: int, line_end: int, column_end: int)

  Record an informational note diagnostic.
- func diag_has_errors(dc: DiagCollector*) → bool

  Report whether the collector contains any errors.
- func diag_count(dc: DiagCollector*) → int

  Return the number of collected diagnostics.
- func diag_get(dc: DiagCollector*, index: int) → Diagnostic*

  Return a diagnostic from the collector.
- func diag_copy(dst: DiagCollector*, src: Diagnostic*)

  Copy a diagnostic into another collector.
- func diag_free(dc: DiagCollector*)

  Free a diagnostic collector and its backing storage.

Variables

- type DiagnosticVector = VectorBase

8.81.1 Enumeration Type Documentation

8.81.1.1 DiagnosticSeverity

```
enum DiagnosticSeverity
```

Defines the DiagnosticSeverity enumeration.

Enumerator

Error	
Warning	
Note	

8.81.2 Function Documentation

8.81.2.1 dgv_create()

```
func dgv_create() → DiagnosticVector*
```

Create an empty vector of diagnostics.

Returns

Diagnostic vector with an initial small capacity.

8.81.2.2 dgv_free()

```
func dgv_free(self: DiagnosticVector*)
```

Free a diagnostic vector and release ARC-owned fields in each entry.

Parameters

self	Diagnostic vector to free.

8.81.2.3 dgv_get()

```
func dgv_get(self: DiagnosticVector*, index: int) → Diagnostic*
```

Return a diagnostic from the vector.

Parameters

self	Diagnostic vector to inspect.
index	Zero-based diagnostic index.

Returns

Diagnostic at `index`.

8.81.2.4 dgv_length()

```
func dgv_length(self: DiagnosticVector*) → int
```

Return the number of diagnostics stored in the vector.

Parameters

self	Diagnostic vector to inspect.

Returns

Total diagnostic count.

8.81.2.5 dgv_push()

```
func dgv_push(self:  DiagnosticVector*, diag:  Diagnostic)
```

Append a diagnostic to the vector.

Parameters

self	Diagnostic vector to extend.
diag	Diagnostic value to copy into the vector.

8.81.2.6 diag_copy()

```
func diag_copy(dst:  DiagCollector*, src:  Diagnostic*)
```

Copy a diagnostic into another collector.

Parameters

dst	Destination collector.
src	Diagnostic to copy.

8.81.2.7 diag_count()

```
func diag_count(dc:  DiagCollector*) → int
```

Return the number of collected diagnostics.

Parameters

dc	Diagnostic collector to inspect.

Returns

Total diagnostic count.

8.81.2.8 diag_create()

```
func diag_create() → DiagCollector*
```

Create an empty diagnostic collector.

Returns

New collector with no diagnostics.

8.81.2.9 diag_error()

```
func diag_error(dc: DiagCollector*, phase: string, code: string, message: string, filename:
string, line: int, column: int, line_end: int, column_end: int)
```

Record an error diagnostic.

Parameters

dc	Collector receiving the diagnostic.
phase	Compiler phase name.
code	Diagnostic code.
message	Diagnostic message.
filename	Source filename, or empty when unavailable.
line	Starting line number.
column	Starting column number.
line_end	Ending line number.
column_end	Ending column number.

8.81.2.10 diag_free()

```
func diag_free(dc: DiagCollector*)
```

Free a diagnostic collector and its backing storage.

Parameters

dc	Diagnostic collector to free.

8.81.2.11 diag_get()

```
func diag_get(dc: DiagCollector*, index: int) → Diagnostic*
```

Return a diagnostic from the collector.

Parameters

dc	Diagnostic collector to inspect.
index	Zero-based diagnostic index.

Returns

Diagnostic at `index`.

8.81.2.12 diag_has_errors()

```
func diag_has_errors(dc: DiagCollector*) → bool
```

Report whether the collector contains any errors.

Parameters

dc	Diagnostic collector to inspect.

Returns

`true` if at least one error diagnostic has been recorded.

8.81.2.13 diag_note()

```
func diag_note(dc: DiagCollector*, phase: string, code: string, message: string, filename:
string, line: int, column: int, line_end: int, column_end: int)
```

Record an informational note diagnostic.

Parameters

dc	Collector receiving the diagnostic.
phase	Compiler phase name.
code	Diagnostic code.
message	Diagnostic message.
filename	Source filename, or empty when unavailable.
line	Starting line number.
column	Starting column number.
line_end	Ending line number.
column_end	Ending column number.

8.81.2.14 diag_warn()

```
func diag_warn(dc: DiagCollector*, phase: string, code: string, message: string, filename:
string, line: int, column: int, line_end: int, column_end: int)
```

Record a warning diagnostic.

Parameters

dc	Collector receiving the diagnostic.

Parameters

phase	Compiler phase name.
code	Diagnostic code.
message	Diagnostic message.
filename	Source filename, or empty when unavailable.
line	Starting line number.
column	Starting column number.
line_end	Ending line number.
column_end	Ending column number.

8.81.3 Variable Documentation

8.81.3.1 DiagnosticVector

```
type DiagnosticVector = VectorBase
```

8.82 compiler/stage2_l0/src/util/intset.l0 File Reference

Module: `util.intset`

```
import std.array;
import std.assert;
import sys.memory;
```

Classes

- struct IntSet

  IntSet is an open-addressing hash set of integers with linear probing.

Functions

- func _iset_slot(key: int, cap: int) → int

  Compute a non-negative slot index from a key and a power-of-2 capacity.
- func iset_create() → IntSet∗

  Creates a new IntSet with default capacity.
- func iset_create_with_capacity(min_cap: int) → IntSet∗

  Creates a new IntSet with at least the specified capacity.
- func _iset_alloc(cap: int) → IntSet∗

  Internal allocator for IntSet.
- func _iset_get_state(self: IntSet∗, i: int) → byte

  Returns the state of the specified slot.
- func _iset_set_state(self: IntSet∗, i: int, s: byte)

  Sets the state of the specified slot.

- func _iset_get_key(self: IntSet*, i: int) → int

  Returns the key of the specified slot.
- func _iset_set_key(self: IntSet*, i: int, key: int)

  Sets the key of the specified slot.
- func _iset_find(self: IntSet*, key: int) → int

  Find the slot index of an existing key.
- func _iset_find_insert(self: IntSet*, key: int) → int

  Find the first non-occupied slot for insertion.
- func _iset_needs_grow(self: IntSet*) → bool

  Checks if the set needs to grow.
- func _iset_rehash(self: IntSet*)

  Rehashes the set to a larger capacity.
- func iset_add(self: IntSet*, key: int) → bool

  Add an integer to the set.
- func iset_has(self: IntSet*, key: int) → bool

  Check whether an integer is in the set.
- func iset_remove(self: IntSet*, key: int) → bool

  Remove an integer from the set.
- func iset_size(self: IntSet*) → int

  Returns the number of entries in the set.
- func iset_capacity(self: IntSet*) → int

  Returns the current capacity of the set.
- func iset_clear(self: IntSet*)

  Remove all entries from the set.
- func iset_free(self: IntSet*)

  Free the set and all backing storage.

Variables

- let ISET_EMPTY: byte = 0
- let ISET_OCCUPIED: byte = 1
- let ISET_TOMBSTONE: byte = 2
- let ISET_DEFAULT_CAP: int = 16

8.82.1 Function Documentation

8.82.1.1 _iset_alloc()

```
func _iset_alloc(cap:  int) → IntSet*
```

Internal allocator for IntSet.

Parameters

cap	The capacity to allocate.

Returns

A pointer to the newly allocated IntSet.

8.82.1.2 _iset_find()

```
func _iset_find(self:  IntSet*, key:  int) → int
```

Find the slot index of an existing key.

Parameters

self	The pointer to the IntSet.
key	The key to find.

Returns

The slot index if found, or -1 if not in the set.

8.82.1.3 _iset_find_insert()

```
func _iset_find_insert(self:  IntSet*, key:  int) → int
```

Find the first non-occupied slot for insertion.

Parameters

self	The pointer to the IntSet.
key	The key to insert.

Returns

The available slot index.

8.82.1.4 _iset_get_key()

```
func _iset_get_key(self:  IntSet*, i:  int) → int
```

Returns the key of the specified slot.

Parameters

self	The pointer to the IntSet.
i	The slot index.

Returns

The key value of the slot.

8.82.1.5 _iset_get_state()

```
func _iset_get_state(self:  IntSet*, i:  int) → byte
```

Returns the state of the specified slot.

Parameters

self	The pointer to the IntSet.
i	The slot index.

Returns

The state byte of the slot.

8.82.1.6 _iset_needs_grow()

```
func _iset_needs_grow(self:  IntSet*) → bool
```

Checks if the set needs to grow.

Parameters

self	The pointer to the IntSet.

Returns

True if growth is needed, false otherwise.

8.82.1.7 _iset_rehash()

```
func _iset_rehash(self:  IntSet*)
```

Rehashes the set to a larger capacity.

Parameters

self	The pointer to the IntSet.

8.82.1.8 _iset_set_key()

```
func _iset_set_key(self:  IntSet*, i:  int, key:  int)
```

Sets the key of the specified slot.

Parameters

self	The pointer to the IntSet.
i	The slot index.
key	The key value to set.

8.82.1.9 _iset_set_state()

```
func _iset_set_state(self:  IntSet*, i:  int, s:  byte)
```

Sets the state of the specified slot.

Parameters

self	The pointer to the IntSet.
i	The slot index.
s	The state byte to set.

8.82.1.10 _iset_slot()

```
func _iset_slot(key:  int, cap:  int) → int
```

Compute a non-negative slot index from a key and a power-of-2 capacity.

Parameters

key	The key value.
cap	The capacity of the set.

Returns

The calculated slot index.

8.82.1.11 iset_add()

```
func iset_add(self:  IntSet*, key:  int) → bool
```

Add an integer to the set.

Returns true if the integer was newly added, false if already present.

Parameters

self	The pointer to the IntSet.
key	The key to add.

Returns

True if the integer was newly added, false otherwise.

8.82.1.12 iset_capacity()

```
func iset_capacity(self:  IntSet*) → int
```

Returns the current capacity of the set.

Parameters

self	The pointer to the IntSet.

Returns

The capacity.

8.82.1.13 iset_clear()

```
func iset_clear(self:  IntSet*)
```

Remove all entries from the set.

Parameters

self	The pointer to the IntSet.

8.82.1.14 iset_create()

```
func iset_create() → IntSet*
```

Creates a new IntSet with default capacity.

Returns

A pointer to the newly created IntSet.

8.82.1.15 iset_create_with_capacity()

```
func iset_create_with_capacity(min_cap:  int) → IntSet*
```

Creates a new IntSet with at least the specified capacity.

Parameters

min_cap	The minimum capacity for the set.

Returns

A pointer to the newly created IntSet.

8.82.1.16 iset_free()

```
func iset_free(self:  IntSet*)
```

Free the set and all backing storage.

Parameters

self	The pointer to the IntSet to free.

8.82.1.17 iset_has()

```
func iset_has(self:  IntSet*, key:  int) → bool
```

Check whether an integer is in the set.

Parameters

self	The pointer to the IntSet.
key	The key to check.

Returns

True if the integer is in the set, false otherwise.

8.82.1.18 iset_remove()

```
func iset_remove(self:  IntSet*, key:  int) → bool
```

Remove an integer from the set.

Returns true if it was found and removed.

Parameters

self	The pointer to the IntSet.
key	The key to remove.

Returns

True if found and removed, false otherwise.

8.82.1.19 iset_size()

```
func iset_size(self:  IntSet*) → int
```

Returns the number of entries in the set.

Parameters

self	The pointer to the IntSet.

Returns

The count of elements.

8.82.2 Variable Documentation

8.82.2.1 ISET_DEFAULT_CAP

```
let ISET_DEFAULT_CAP: int = 16
```

8.82.2.2 ISET_EMPTY

```
let ISET_EMPTY: byte = 0
```

8.82.2.3 ISET_OCCUPIED

```
let ISET_OCCUPIED: byte = 1
```

8.82.2.4 ISET_TOMBSTONE

```
let ISET_TOMBSTONE: byte = 2
```

8.83 compiler/stage2_l0/src/util/log.l0 File Reference

Module: `util.log`

```
import std.text;
import std.string;
import std.io;
```

Classes

- struct LogConfig

  Represents the LogConfig structure.

Enumerations

- enum LogLevel

  Defines the LogLevel enumeration.

Functions

- func log_config_create(level: LogLevel, rich: bool) → LogConfig∗

  Create a logging configuration.
- func log_level_from_verbosity(verbosity: int) → LogLevel

  Map a CLI verbosity count to a log level.
- func log_level_name(level: LogLevel) → string

  Return the printable name for a log level.
- func log_msg(cfg: LogConfig∗, level: LogLevel, message: string)

  Emit a log message if the configuration allows the given level.
- func log_error(cfg: LogConfig∗, message: string)

  Log an error message.
- func log_warning(cfg: LogConfig∗, message: string)

  Log a warning message.
- func log_info(cfg: LogConfig∗, message: string)

  Log an informational message.
- func log_debug(cfg: LogConfig∗, message: string)

  Log a debug message.
- func log_format_stage(stage: string, module_name: string) → string

  Format a stage progress message.
- func log_stage(cfg: LogConfig∗, stage: string, module_name: string)

  Log a stage progress message at info level.

8.83.1 Enumeration Type Documentation

8.83.1.1 LogLevel

```
enum LogLevel
```

Defines the LogLevel enumeration.

Enumerator

LL_SILENT	
LL_ERROR	
LL_WARNING	
LL_INFO	
LL_DEBUG	

8.83.2 Function Documentation

8.83.2.1 log_config_create()

```
func log_config_create(level:  LogLevel, rich:  bool) → LogConfig*
```

Create a logging configuration.

Parameters

level	Minimum log level to emit.
rich	Whether to prefix messages with level labels.

Returns

A new logging configuration.

8.83.2.2 log_debug()

```
func log_debug(cfg:  LogConfig*, message:  string)
```

Log a debug message.

Parameters

cfg	Logging configuration.
message	Message text to emit.

8.83.2.3 log_error()

```
func log_error(cfg:  LogConfig*, message:  string)
```

Log an error message.

Parameters

cfg	Logging configuration.
message	Message text to emit.

8.83.2.4 log_format_stage()

```
func log_format_stage(stage:  string, module_name:  string) → string
```

Format a stage progress message.

Parameters

stage	Human-readable stage label.
module_name	Module name to include when available.

Returns

Formatted stage message.

8.83.2.5 log_info()

```
func log_info(cfg:  LogConfig*, message:  string)
```

Log an informational message.

Parameters

cfg	Logging configuration.
message	Message text to emit.

8.83.2.6 log_level_from_verbosity()

```
func log_level_from_verbosity(verbosity:  int) → LogLevel
```

Map a CLI verbosity count to a log level.

Parameters

verbosity	Verbosity count from command-line flags.

Returns

Log level implied by `verbosity`.

8.83.2.7 log_level_name()

```
func log_level_name(level:  LogLevel) → string
```

Return the printable name for a log level.

Parameters

level	Log level to format.

Returns

Uppercase display name for `level`.

8.83.2.8 log_msg()

```
func log_msg(cfg:  LogConfig*, level:  LogLevel, message:  string)
```

Emit a log message if the configuration allows the given level.

Parameters

cfg	Logging configuration.
level	Severity level of the message.
message	Message text to emit.

8.83.2.9 log_stage()

```
func log_stage(cfg:  LogConfig*, stage:  string, module_name:  string)
```

Log a stage progress message at info level.

Parameters

cfg	Logging configuration.
stage	Human-readable stage label.
module_name	Module name to include when available.

8.83.2.10 log_warning()

```
func log_warning(cfg:  LogConfig*, message:  string)
```

Log a warning message.

Parameters

cfg	Logging configuration.
message	Message text to emit.

8.84 compiler/shared/l0/stdlib/std/path.l0 File Reference

Module: `std.path`

```
import std.text;
import std.string;
```

Functions

- func is_sep(c: byte) → bool

  Check whether a byte is a path separator.
- func is_absolute(path: string) → bool

  Check whether a path is absolute.
- func _path_find_last_sep(path: string) → int

  Find the last path separator in a string.
- func _path_ext_with_dot(ext: string) → string

  Normalize an extension so callers may pass either `txt` or `.txt`.
- func has_parent(path: string) → bool

  Check whether a path has a parent component.
- func basename(path: string) → string

  Return the last component of a path.
- func parent(path: string) → string

  Return the parent directory of a path.
- func stem(path: string) → string

  Return a path basename without its final extension when present.
- func join(root: string, rel: string) → string

  Join a root path and relative path with one separator.
- func has_extension(path: string, ext: string) → bool

  Check whether a path basename ends with the requested extension.

8.84.1 Function Documentation

8.84.1.1 _path_ext_with_dot()

```
func _path_ext_with_dot(ext:  string) → string
```

Normalize an extension so callers may pass either `txt` or `.txt`.

Parameters

ext	Extension text to normalize.

Returns

Extension with a leading `.` when non-empty.

8.84.1.2 _path_find_last_sep()

```
func _path_find_last_sep(path:  string) → int
```

Find the last path separator in a string.

Parameters

path	Path string to inspect.

Returns

Index of the last separator, or `-1` if none is present.

8.84.1.3 basename()

```
func basename(path: string) → string
```

Return the last component of a path.

Parameters

path	Path string to inspect.

Returns

Basename with trailing separators ignored.

8.84.1.4 has_extension()

```
func has_extension(path: string, ext: string) → bool
```

Check whether a path basename ends with the requested extension.

Parameters

path	Path string to inspect.
ext	Extension to match, with or without leading `.`

Returns

`true` if `path` has the requested final extension.

8.84.1.5 has_parent()

```
func has_parent(path: string) → bool
```

Check whether a path has a parent component.

Parameters

path	Path string to inspect.

Returns

`true` if the path contains a separator.

8.84.1.6 is_absolute()

```
func is_absolute(path:  string) → bool
```

Check whether a path is absolute.

Parameters

path	Path string to inspect.

Returns

`true` for POSIX absolute paths and Windows drive-rooted paths.

8.84.1.7 is_sep()

```
func is_sep(c:  byte) → bool
```

Check whether a byte is a path separator.

Parameters

c	Byte to inspect.

Returns

`true` if `c` is `/` or `\\`.

8.84.1.8 join()

```
func join(root:  string, rel:  string) → string
```

Join a root path and relative path with one separator.

Parameters

root	Root directory path.
rel	Relative path to append.

Returns

Combined path.

8.84.1.9 parent()

```
func parent(path:  string) → string
```

Return the parent directory of a path.

Parameters

path	Path string to inspect.

Returns

Parent directory, or `.` when the path has no parent.

8.84.1.10 stem()

```
func stem(path:  string) → string
```

Return a path basename without its final extension when present.

Parameters

path	Path string to inspect.

Returns

Basename stem.

8.85 compiler/stage2_l0/src/util/path.l0 File Reference

Module: `util.path`

```
import std.string;
import std.path;
```

Functions

- func up_is_sep(c: byte) → bool
  Check whether a byte is a path separator.
- func up_is_absolute(path: string) → bool
  Check whether a path is absolute.
- func up_find_last_sep(path: string) → int
  Find the last path separator in a string.
- func up_has_parent(path: string) → bool
  Check whether a path has a parent component.
- func up_basename(path: string) → string
  Return the last component of a path.
- func up_parent(path: string) → string
  Return the parent directory of a path.
- func up_is_l0_file(path: string) → bool
  Check whether a path ends with `.l0`.
- func up_stem(path: string) → string
  Return a path basename without the `.l0` suffix when present.
- func up_join(root: string, rel: string) → string
  Join a root path and relative path with one separator.

8.85.1 Function Documentation

8.85.1.1 up_basename()

```
func up_basename(path:  string) → string
```

Return the last component of a path.

Parameters

path	Path string to inspect.

Returns

Basename with trailing separators ignored.

8.85.1.2 up_find_last_sep()

```
func up_find_last_sep(path:  string) → int
```

Find the last path separator in a string.

Parameters

path	Path string to inspect.

Returns

Index of the last separator, or `-1` if none is present.

8.85.1.3 up_has_parent()

```
func up_has_parent(path:  string) → bool
```

Check whether a path has a parent component.

Parameters

path	Path string to inspect.

Returns

`true` if the path contains a separator.

8.85.1.4 up_is_absolute()

```
func up_is_absolute(path:  string) → bool
```

Check whether a path is absolute.

Parameters

path	Path string to inspect.

Returns

`true` for POSIX absolute paths and Windows drive-rooted paths.

8.85.1.5 up_is_l0_file()

```
func up_is_l0_file(path:  string) → bool
```

Check whether a path ends with `.l0`.

Parameters

path	Path string to inspect.

Returns

`true` if `path` has the `.l0` suffix.

8.85.1.6 up_is_sep()

```
func up_is_sep(c:  byte) → bool
```

Check whether a byte is a path separator.

Parameters

c	Byte to inspect.

Returns

`true` if `c` is `/` or `\\`.

8.85.1.7 up_join()

```
func up_join(root:  string, rel:  string) → string
```

Join a root path and relative path with one separator.

Parameters

root	Root directory path.
rel	Relative path to append.

Returns

Combined path.

8.85.1.8 up_parent()

```
func up_parent(path:  string) → string
```

Return the parent directory of a path.

Parameters

path	Path string to inspect.

Returns

Parent directory, or `.` when the path has no parent.

8.85.1.9 up_stem()

```
func up_stem(path:  string) → string
```

Return a path basename without the `.l0` suffix when present.

Parameters

path	Path string to inspect.

Returns

Basename stem.

8.86 compiler/stage2_l0/src/util/strings.l0 File Reference

Module: `util.strings`

```
import std.vector;
import std.text;
import std.string;
import sys.rt;
```

Functions

- func us_sv_clone(src: StringVector*) → StringVector*

  Clone a vector of strings.
- func us_split_char(s: string, sep: byte, keep_empty: bool) → StringVector*

  Split a string on one separator byte.
- func us_split_chars2_non_empty(s: string, sep1: byte, sep2: byte) → StringVector*

  Split a string on either of two separator bytes, dropping empty fields.
- func us_split_is_space_non_empty(s: string) → StringVector*

  Split a string on space characters, dropping empty fields.
- func us_strip_utf8_bom(s: string) → string

  Strip a leading UTF-8 BOM from a string when present.
- func us_is_utf8_continuation(c: byte) → bool

  Return whether a byte is a UTF-8 continuation byte.
- func us_is_valid_utf8(s: string) → bool

  Validate that a string contains only well-formed UTF-8 byte sequences.

8.86.1 Function Documentation

8.86.1.1 us_is_utf8_continuation()

```
func us_is_utf8_continuation(c:  byte) → bool
```

Return whether a byte is a UTF-8 continuation byte.

Parameters

c	Byte to inspect.

Returns

`true` for bytes in the `0x80..0xBF` range.

8.86.1.2 us_is_valid_utf8()

```
func us_is_valid_utf8(s:  string) → bool
```

Validate that a string contains only well-formed UTF-8 byte sequences.

The validator rejects overlong forms, surrogate code points, truncated sequences, and byte sequences above `U+10FFFF`.

Parameters

s	String to validate.

Returns

`true` when `s` is well-formed UTF-8.

8.86.1.3 us_split_char()

```
func us_split_char(s:  string, sep:  byte, keep_empty:  bool) → StringVector*
```

Split a string on one separator byte.

Parameters

s	String to split.
sep	Separator byte.
keep_empty	Whether empty fields should be preserved.

Returns

Vector of split parts.

8.86.1.4 us_split_chars2_non_empty()

```
func us_split_chars2_non_empty(s:  string, sep1:  byte, sep2:  byte) → StringVector*
```

Split a string on either of two separator bytes, dropping empty fields.

Parameters

s	String to split.
sep1	First separator byte.
sep2	Second separator byte.

Returns

Vector of non-empty split parts.

8.86.1.5 us_split_is_space_non_empty()

```
func us_split_is_space_non_empty(s:  string) → StringVector*
```

Split a string on space characters, dropping empty fields.

Parameters

s	String to split.

Returns

Vector of non-empty split parts.

See also

std.string::is_space for the definition of space characters.

8.86.1.6 us_strip_utf8_bom()

```
func us_strip_utf8_bom(s:  string) → string
```

Strip a leading UTF-8 BOM from a string when present.

Parameters

s	String to normalize.

Returns

`s` without a leading BOM.

8.86.1.7 us_sv_clone()

```
func us_sv_clone(src:  StringVector*) → StringVector*
```

Clone a vector of strings.

Parameters

src	Source vector to copy.

Returns

New vector containing the same strings in the same order.

8.87 docs-mainpage-latex.md File Reference

Index

_CLI_CANONICAL_SHORT_VALUE_OPTIONS
    l0c, 60
_CLI_LONG_VALUE_OPTIONS
    l0c, 60
_CLI_NAMESPACED_SHORT_VALUE_OPTIONS
    l0c, 60
_CLI_OTHER_SHORT_VALUE_OPTIONS
    l0c, 61
_CLI_VALUE_OPTIONS
    l0c, 61
_EXPR_ENDING_TOKENS
    l0_lexer, 33
_HEX_CHARS
    l0_string_escape, 40
_L0_TAG_ENUM
    l0_runtime.h, 671
_L0_TAG_OPT
    l0_runtime.h, 671
_L0_TAG_PTR
    l0_runtime.h, 671
_L0_TAG_STRUCT
    l0_runtime.h, 671
_NULL_TYPE
    l0_types, 43
_OCT_CHARS
    l0_string_escape, 40
_RESERVED_BINARY_OPS
    l0_parser.Parser, 389
_RT_ACCESS_READ
    l0_runtime.h, 672
_RT_ACCESS_UNTRACKED_OK
    l0_runtime.h, 672
_RT_ACCESS_WRITE
    l0_runtime.h, 672
_RT_ALIGNOF
    l0_runtime.h, 672
_RT_ALLOC_INIT_CAP
    l0_runtime.h, 672
_RT_ALLOC_LIVE
    l0_runtime.h, 672
_RT_ALLOC_POOLED
    l0_runtime.h, 672
_RT_ALLOC_QUARANTINED
    l0_runtime.h, 672
_RT_ALLOC_TOMBSTONE
    l0_runtime.h, 672
_RT_MEM_ARC
    l0_runtime.h, 672
_RT_MEM_FOREIGN
    l0_runtime.h, 673
_RT_MEM_NEW
    l0_runtime.h, 673
_RT_MEM_RAW
    l0_runtime.h, 673
_RT_MEM_SENTINEL
    l0_runtime.h, 720
_RT_MEM_STATIC
    l0_runtime.h, 673
_RT_QUARANTINE_MAX_BYTES
    l0_runtime.h, 673
_RT_QUARANTINE_MAX_COUNT
    l0_runtime.h, 673
_RT_REC_POOL_CHUNK
    l0_runtime.h, 673
_RT_TRACE_ARC
    l0_runtime.h, 673
_RT_TRACE_ARC_LOC
    l0_runtime.h, 673
_RT_TRACE_MEM
    l0_runtime.h, 674
_RT_TRACE_MEM_LOC
    l0_runtime.h, 674
_SIMPLE_ESCAPES
    l0_string_escape, 40
_TRIGRAPH_THIRD_BYTES
    l0_string_escape, 41
__contains__
    l0_compilation.CompilationUnit, 212
__init__
    check_trace_log._MessageCollector, 65
    l0_driver.L0Driver, 307
    l0_driver.SourceEncodingError, 423
    l0_internal_error.InternalCompilerError, 293
    l0_lexer.Lexer, 317
    l0_locals.LocalScopeResolver, 327
    l0_name_resolver.NameResolver, 359
    l0_parser.Parser, 377
    l0_signatures.SignatureResolver, 416
    l0c._TemporarySourceWriteError, 72
__post_init__
    l0_backend.Backend, 89
    l0_expr_types.ExpressionTypeChecker, 249
__repr__
    l0_lexer.Token, 460
_active_line_directive
    l0_c_emitter.CEmitter, 199
_add_all_modules_arg
    l0c, 45

_add_codegen_arg
l0c, 46
_add_runtime_args
l0c, 46
_add_target_args
l0c, 46
_advance
l0_lexer.Lexer, 317
l0_parser.Parser, 378
_alignof_expr_for_type
l0_backend.Backend, 89
_alive_scopes
l0_expr_types.ExpressionTypeChecker, 262
_at_case_arm_start
l0_parser.Parser, 378
_at_end
l0_lexer.Lexer, 317
l0_parser.Parser, 378
_binary_equality
l0_expr_types.ExpressionTypeChecker, 249
_binary_expect_both_bool
l0_expr_types.ExpressionTypeChecker, 249
_binary_expect_both_int
l0_expr_types.ExpressionTypeChecker, 250
_bind_pattern_vars
l0_locals.LocalScopeResolver, 328
_breakable_loop_depth
l0_expr_types.ExpressionTypeChecker, 262
_build_arg_parser
check_trace_log, 22
_build_function_env
l0_locals.LocalScopeResolver, 328
_build_type_dependency_graph
l0_backend.Backend, 89
_can_assign
l0_expr_types.ExpressionTypeChecker, 250
_case_literal_info
l0_expr_types.ExpressionTypeChecker, 250
_check
l0_parser.Parser, 378
_check_block
l0_expr_types.ExpressionTypeChecker, 250
_check_dead_liveness_block
l0_expr_types.ExpressionTypeChecker, 251
_check_entry_main_for_build
l0c, 46
_check_expr_liveness
l0_expr_types.ExpressionTypeChecker, 251
_check_function
l0_expr_types.ExpressionTypeChecker, 251
_check_loop_iteration
l0_expr_types.ExpressionTypeChecker, 251
_check_reserved_binary_op
l0_parser.Parser, 378
_check_return
l0_expr_types.ExpressionTypeChecker, 251
_check_stmt
l0_expr_types.ExpressionTypeChecker, 252
_check_update_from_backedge_states
l0_expr_types.ExpressionTypeChecker, 252
_cleanup_header_ref_guard_stack
l0_expr_types.ExpressionTypeChecker, 262
_clone_alive_scopes
l0_expr_types.ExpressionTypeChecker, 252
_collect_invalid_characters
l0_lexer.Lexer, 317
_collect_locals
l0_name_resolver.NameResolver, 360
_collect_opt_wrappers_from_type
l0_c_emitter.CEmitter, 159
_collect_reassigned_arc_params
l0_backend.Backend, 89
_compile_generated_c
l0c, 47
_compiler_flag_family
l0c, 47
_convert_expr_with_expected_type
l0_backend.Backend, 90
_current_func_env
l0_expr_types.ExpressionTypeChecker, 262
_current_func_result
l0_backend.Backend, 123
_current_func_type
l0_expr_types.ExpressionTypeChecker, 262
_current_scope
l0_backend.Backend, 123
_declare
l0_locals.LocalScopeResolver, 328
_declare_local
l0_expr_types.ExpressionTypeChecker, 252
_decode_escaped_bytes
l0_expr_types.ExpressionTypeChecker, 253
_default_executable_name
l0c, 47
_define_local
l0_name_resolver.NameResolver, 360
_describe_lvalue
l0_expr_types.ExpressionTypeChecker, 253
_detect_value_type_cycles
l0_signatures.SignatureResolver, 417
_diagnostic
l0_expr_types.ExpressionTypeChecker, 253
_dump_tokens_for_file
l0c, 48
_emit
l0_signatures.SignatureResolver, 417
_emit_bare_variant_static_initializer
l0_backend.Backend, 90
_emit_binary_op
l0_backend.Backend, 90
_emit_block
l0_backend.Backend, 91
_emit_block_sequence
l0_backend.Backend, 91
_emit_case
l0_backend.Backend, 91

_emit_case_literal
l0_backend.Backend, 92
_emit_checked_pointer_expr
l0_backend.Backend, 92
_emit_cleanup_at_scope_exit
l0_backend.Backend, 92
_emit_cleanup_by_type
l0_c_emitter.CEmitter, 159
_emit_cleanup_for_loop_exit
l0_backend.Backend, 93
_emit_cleanup_for_return
l0_backend.Backend, 93
_emit_condition_branch
l0_backend.Backend, 93
_emit_condition_expr
l0_backend.Backend, 94
_emit_condition_value
l0_backend.Backend, 94
_emit_const_constructor
l0_backend.Backend, 94
_emit_const_struct_constructor
l0_backend.Backend, 95
_emit_const_variant_constructor
l0_backend.Backend, 95
_emit_copy_expr_with_retains
l0_backend.Backend, 96
_emit_diagnostic
l0c, 48
_emit_drop
l0_backend.Backend, 96
_emit_enum_cleanup
l0_backend.Backend, 96
_emit_enum_cleanup_switch
l0_c_emitter.CEmitter, 159
_emit_enum_value_cleanup
l0_c_emitter.CEmitter, 160
_emit_expr
l0_backend.Backend, 97
_emit_expr_with_expected_type
l0_backend.Backend, 97
_emit_field_cleanup
l0_c_emitter.CEmitter, 160
_emit_for
l0_backend.Backend, 98
_emit_function_declaration
l0_backend.Backend, 98
_emit_function_declarations
l0_backend.Backend, 98
_emit_function_definition
l0_backend.Backend, 98
_emit_function_definitions
l0_backend.Backend, 99
_emit_if_else
l0_backend.Backend, 99
_emit_inline_with_header_item
l0_backend.Backend, 99
_emit_let
l0_backend.Backend, 99
_emit_let_declaration
l0_backend.Backend, 100
_emit_let_declarations
l0_backend.Backend, 100
_emit_let_initializer
l0_backend.Backend, 100, 123
_emit_lexer_error
l0_parser.Parser, 378
_emit_line_directive
l0_backend.Backend, 101
_emit_lvalue_with_caching
l0_backend.Backend, 101
_emit_main_wrapper_if_needed
l0_backend.Backend, 102
_emit_match
l0_backend.Backend, 102
_emit_new_expr
l0_backend.Backend, 102
_emit_optional_wrapper
l0_c_emitter.CEmitter, 160
_emit_ord_intrinsic
l0_backend.Backend, 102
_emit_owned_expr_with_expected_type
l0_backend.Backend, 103
_emit_pattern_bindings
l0_backend.Backend, 103
_emit_pointer_index_lvalue
l0_backend.Backend, 104
_emit_reassignment
l0_backend.Backend, 104
_emit_remaining_lexer_errors
l0_parser.Parser, 378
_emit_retain_for_copied_value
l0_backend.Backend, 104
_emit_return
l0_backend.Backend, 105
_emit_sizeof_intrinsic
l0_backend.Backend, 105
_emit_stmt
l0_backend.Backend, 106
_emit_struct_cleanup
l0_backend.Backend, 106
_emit_struct_constructor
l0_backend.Backend, 106
_emit_temporary_source_cleanup_failure
l0c, 48
_emit_unwrap
l0_backend.Backend, 107
_emit_value_cleanup
l0_backend.Backend, 107
_emit_variant_constructor
l0_backend.Backend, 108
_emit_verbose_compiler_identity
l0c, 48
_emit_while
l0_backend.Backend, 108
_emit_with
l0_backend.Backend, 108

_emit_with_cleanup_from_scope
l0_backend.Backend, 109
_emit_with_cleanup_header_let_assign
l0_backend.Backend, 109
_emit_with_cleanup_header_let_predecl
l0_backend.Backend, 109
_enum_has_arc_data
l0_c_emitter.CEmitter, 160
_enum_type_for_variant_symbol
l0_signatures.SignatureResolver, 417
_error
l0_expr_types.ExpressionTypeChecker, 253
l0_lexer.Lexer, 317
l0_parser.Parser, 379
_error_bail
l0_parser.Parser, 379
_error_unexpected
l0_parser.Parser, 379
_expect
l0_parser.Parser, 380
_expect_expr_type
l0_backend.Backend, 110
_expect_semicolon
l0_parser.Parser, 380
_expect_variable_name
l0_parser.Parser, 380
_expr_contains_try
l0_expr_types.ExpressionTypeChecker, 253
_extend_span
l0_parser.Parser, 380
_extern_signatures_compatible
l0_name_resolver.NameResolver, 360
_extract_value_type_dependencies
l0_backend.Backend, 110
l0_signatures.SignatureResolver, 417
_find_cc
l0c, 49
_find_cycle_details
l0_backend.Backend, 111
_find_enum_decl
l0_backend.Backend, 111
_find_struct_decl
l0_backend.Backend, 112
_format_span
l0_ast_printer, 27
_format_token_dump_text
l0c, 49
_fresh_label
l0_backend.Backend, 112
_fs_from_rt
fs.l0, 515
_gen_if_else_branch
l0_backend.Backend, 112
_get_const_int_for_explicit_cast
l0_expr_types.ExpressionTypeChecker, 253
_get_dotted_module_name
l0_parser.Parser, 380
_get_enum_info
l0_c_emitter.CEmitter, 161
_get_module_names
l0c, 49
_get_optimize_flag
l0c, 49
_get_struct_info
l0_c_emitter.CEmitter, 161
_has_side_effects
l0_backend.Backend, 113
_hm_slot
hashmap.l0, 521
_hs_slot
hashset.l0, 540
_infer_binary
l0_expr_types.ExpressionTypeChecker, 254
_infer_call
l0_expr_types.ExpressionTypeChecker, 254
_infer_cast
l0_expr_types.ExpressionTypeChecker, 254
_infer_expr
l0_expr_types.ExpressionTypeChecker, 254
_infer_field_access
l0_expr_types.ExpressionTypeChecker, 255
_infer_index
l0_expr_types.ExpressionTypeChecker, 255
_infer_literal_type
l0_signatures.SignatureResolver, 417
_infer_new
l0_expr_types.ExpressionTypeChecker, 255
_infer_ord_intrinsic
l0_expr_types.ExpressionTypeChecker, 255
_infer_sizeof_intrinsic
l0_expr_types.ExpressionTypeChecker, 255
_infer_struct_constructor
l0_expr_types.ExpressionTypeChecker, 255
_infer_try
l0_expr_types.ExpressionTypeChecker, 256
_infer_type_expr
l0_expr_types.ExpressionTypeChecker, 256
_infer_unary
l0_expr_types.ExpressionTypeChecker, 256
_infer_var_ref
l0_expr_types.ExpressionTypeChecker, 256
_infer_variant_constructor
l0_expr_types.ExpressionTypeChecker, 256
_init_env_defaults
l0c, 50
_int_type_size
l0_backend.Backend, 113
_io_check_byte_range
io.l0, 561
_is_ascii_ident_part
l0_lexer, 32
_is_ascii_ident_start
l0_lexer, 32
_is_binary_op_enabled
l0_backend.Backend, 114
_is_bool

l0_expr_types.ExpressionTypeChecker, 256
_is_builtin_type_name
l0_parser.Parser, 381
_is_const_null_for_explicit_cast
l0_expr_types.ExpressionTypeChecker, 257
_is_early_inner
l0_c_emitter.CEmitter, 162
_is_extern_function
l0_backend.Backend, 114
_is_guarded_cleanup_header_ref
l0_expr_types.ExpressionTypeChecker, 257
_is_imported
l0_resolve, 37
_is_int_assignable
l0_backend.Backend, 114
l0_expr_types.ExpressionTypeChecker, 257
_is_niche_nullable
l0_c_emitter.CEmitter, 162
_is_nullable_or_ptr
l0_expr_types.ExpressionTypeChecker, 257
_is_place_expr
l0_backend.Backend, 115
_is_string
l0_expr_types.ExpressionTypeChecker, 257
_is_unambiguous_type_start
l0_parser.Parser, 381
_is_unwrap_cast_from_place
l0_backend.Backend, 115
_is_valid_module_name
l0c, 50
_is_void
l0_expr_types.ExpressionTypeChecker, 257
_is_windows_host
l0c, 50
_iset_alloc
intset.l0, 1102
_iset_find
intset.l0, 1102
_iset_find_insert
intset.l0, 1103
_iset_get_key
intset.l0, 1103
_iset_get_state
intset.l0, 1103
_iset_needs_grow
intset.l0, 1104
_iset_rehash
intset.l0, 1104
_iset_set_key
intset.l0, 1104
_iset_set_state
intset.l0, 1105
_iset_slot
intset.l0, 1105
_islm_entry_by_index
linear_map.l0, 576
_islm_find_by_key
linear_map.l0, 576
_iter_body_stmts
l0_backend.Backend, 115
_iter_events
check_trace_log, 22
_iter_variant_cleanup_fields
l0_c_emitter.CEmitter, 162
_l0_base_opt, 63
has_value, 63
_l0_h_string, 63
bytes, 64
len, 64
refcount, 64
_l0_sh_tag_bool
l0_runtime.h, 719
_l0_sh_tag_byte
l0_runtime.h, 719
_l0_sh_tag_data
l0_runtime.h, 719
_l0_sh_tag_int
l0_runtime.h, 719
_l0_sh_tag_string
l0_runtime.h, 719
_label_counter
l0_backend.Backend, 123
_last
l0_parser.Parser, 381
_last_token
l0_parser.Parser, 389
_lexer_error_token
l0_lexer.Lexer, 317
_line_strip_trailing_cr
text.l0, 607
_liveness_diagnostics_only
l0_expr_types.ExpressionTypeChecker, 262
_lm_entry_by_hk_key
linear_map.l0, 577
_lm_entry_by_hv_value
linear_map.l0, 577
_lm_entry_by_index
linear_map.l0, 577
_lm_grow
linear_map.l0, 578
_lm_key_by_entry
linear_map.l0, 578
_lm_new_entry
linear_map.l0, 578
_lm_set_entry
linear_map.l0, 579
_lm_value_by_entry
linear_map.l0, 579
_load_file_lines
l0c, 50
_load_single_file
l0_driver.L0Driver, 308
_local_scopes
l0_expr_types.ExpressionTypeChecker, 262
_logical_predecessor_ends_expression
l0_lexer, 32

_logical_token_at_cursor
l0_parser.Parser, 381
_lookahead_is_type_suffix
l0_parser.Parser, 381
_lookup_alive
l0_expr_types.ExpressionTypeChecker, 258
_lookup_local
l0_expr_types.ExpressionTypeChecker, 258
_lookup_local_scope_index
l0_expr_types.ExpressionTypeChecker, 258
_lookup_local_var_type
l0_backend.Backend, 116
_lookup_owned_local_name
l0_backend.Backend, 116
_lookup_symbol
l0_backend.Backend, 116
_loop_cleanup_scope_stack
l0_backend.Backend, 123
_loop_flow_capture_stack
l0_expr_types.ExpressionTypeChecker, 262
_loop_label_stack
l0_backend.Backend, 123
_loop_liveness_fixed_point
l0_expr_types.ExpressionTypeChecker, 258
_make_param_alive_scope
l0_expr_types.ExpressionTypeChecker, 258
_make_param_scope
l0_expr_types.ExpressionTypeChecker, 258
_match
l0_parser.Parser, 382
_materialize_arc_temp
l0_backend.Backend, 117
_meet_alive_scopes
l0_expr_types.ExpressionTypeChecker, 259
_meet_alive_state
l0_expr_types.ExpressionTypeChecker, 259
_needs_arc_temp
l0_backend.Backend, 117
_next_stmt_unreachable
l0_backend.Backend, 123, 124
l0_expr_types.ExpressionTypeChecker, 262
_next_token
l0_lexer.Lexer, 318
_normalize_cli_argv
l0c, 51
_open_imports
l0_name_resolver.NameResolver, 360
_opt_emitted
l0_c_emitter.CEmitter, 199
_opt_key_for_type
l0_c_emitter.CEmitter, 162
_opt_wrapper_name_for_inner
l0_c_emitter.CEmitter, 163
_opt_wrappers
l0_c_emitter.CEmitter, 199
_output_flags
l0c, 51
_parse_add_expr
l0_parser.Parser, 382
_parse_and_expr
l0_parser.Parser, 382
_parse_block
l0_parser.Parser, 382
_parse_break_stmt
l0_parser.Parser, 382
_parse_call_argument
l0_parser.Parser, 382
_parse_case_literal
l0_parser.Parser, 383
_parse_case_stmt
l0_parser.Parser, 383
_parse_cast_expr
l0_parser.Parser, 383
_parse_continue_stmt
l0_parser.Parser, 383
_parse_drop_stmt
l0_parser.Parser, 383
_parse_enum
l0_parser.Parser, 383
_parse_equality_expr
l0_parser.Parser, 384
_parse_expr
l0_parser.Parser, 384
_parse_extern_func
l0_parser.Parser, 384
_parse_for_stmt
l0_parser.Parser, 384
_parse_for_update_stmt
l0_parser.Parser, 384
_parse_function
l0_parser.Parser, 384
_parse_if_stmt
l0_parser.Parser, 384
_parse_let_stmt
l0_parser.Parser, 385
_parse_match_stmt
l0_parser.Parser, 385
_parse_mul_expr
l0_parser.Parser, 385
_parse_or_expr
l0_parser.Parser, 385
_parse_pattern
l0_parser.Parser, 385
_parse_postfix_expr
l0_parser.Parser, 385
_parse_primary_expr
l0_parser.Parser, 385
_parse_rel_expr
l0_parser.Parser, 386
_parse_return_stmt
l0_parser.Parser, 386
_parse_simple_stmt
l0_parser.Parser, 386
_parse_source
l0_driver.L0Driver, 308
_parse_stmt

l0_parser.Parser, 386
_parse_struct
l0_parser.Parser, 386
_parse_top_level_decl
l0_parser.Parser, 386
_parse_top_level_let
l0_parser.Parser, 386
_parse_type
l0_parser.Parser, 387
_parse_type_alias
l0_parser.Parser, 387
_parse_unary_expr
l0_parser.Parser, 387
_parse_while_stmt
l0_parser.Parser, 387
_parse_with_item
l0_parser.Parser, 387
_parse_with_stmt
l0_parser.Parser, 387
_path_ext_with_dot
path.l0, 1113
_path_find_last_sep
path.l0, 1113
_peek
l0_lexer.Lexer, 318
l0_parser.Parser, 387
_peek_next
l0_lexer.Lexer, 318
_pointer_type_or_none
l0_backend.Backend, 118
_pop_scope
l0_backend.Backend, 118
l0_expr_types.ExpressionTypeChecker, 259
_prev_ends_expression
l0_lexer.Lexer, 320
_print_report
check_trace_log, 22
_push_scope
l0_backend.Backend, 118
l0_expr_types.ExpressionTypeChecker, 259
_read_byte_literal
l0_lexer.Lexer, 318
_read_number
l0_lexer.Lexer, 318
_read_string_literal
l0_lexer.Lexer, 318
_read_valid_char_escape
l0_lexer.Lexer, 319
_register_inline_with_cleanup
l0_backend.Backend, 118
_reject_name_qualifier
l0_expr_types.ExpressionTypeChecker, 259
_remove_temporary_c_source
l0c, 51
_resolve_enum
l0_signatures.SignatureResolver, 417
_resolve_func
l0_signatures.SignatureResolver, 418
_resolve_let
l0_signatures.SignatureResolver, 418
_resolve_let_type
l0_backend.Backend, 118
_resolve_module_signatures
l0_signatures.SignatureResolver, 418
_resolve_struct
l0_signatures.SignatureResolver, 418
_resolve_type_alias
l0_signatures.SignatureResolver, 418
_resolve_type_alias_symbol
l0_signatures.SignatureResolver, 418
_resolve_type_ref
l0_backend.Backend, 119
l0_expr_types.ExpressionTypeChecker, 259
l0_signatures.SignatureResolver, 419
_return_paths
l0_expr_types.ExpressionTypeChecker, 262
_rt_alloc_hash
l0_runtime.h, 677
_rt_alloc_next_generation
l0_runtime.h, 719
_rt_alloc_obj
l0_runtime.h, 677
_rt_alloc_record, 67
base, 67
cold_index, 67
generation, 67
hot_pad, 68
l0_runtime.h, 675
mem_kind, 68
q_next, 68
read_only, 68
size, 68
state, 68
tree_left, 68
tree_prio, 68
tree_right, 68
_rt_alloc_record_cold, 69
align, 69
alloc_file, 69
alloc_line, 69
drop_file, 69
drop_line, 69
l0_runtime.h, 675
reserved, 69
type_id, 70
_rt_alloc_record_size_check
l0_runtime.h, 675
_rt_alloc_string
l0_runtime.h, 677
_rt_alloc_table
l0_runtime.h, 719
_rt_alloc_table_cap
l0_runtime.h, 719
_rt_alloc_table_cnt
l0_runtime.h, 720
_rt_alloc_table_insert

l0_runtime.h, 678
_rt_alloc_table_lookup
l0_runtime.h, 678
_rt_alloc_table_rehash
l0_runtime.h, 678
_rt_alloc_table_remove
l0_runtime.h, 678
_rt_alloc_table_tombstones
l0_runtime.h, 720
_rt_alloc_tree_find_containing
l0_runtime.h, 678
_rt_alloc_tree_glb
l0_runtime.h, 678
_rt_alloc_tree_insert
l0_runtime.h, 678
_rt_alloc_tree_lub
l0_runtime.h, 679
_rt_alloc_tree_remove
l0_runtime.h, 679
_rt_alloc_tree_root
l0_runtime.h, 720
_rt_argc
l0_runtime.h, 720
_rt_argv
l0_runtime.h, 720
_rt_check_index_ptr_site
l0_runtime.h, 679
_rt_check_index_ptr_site_slow
l0_runtime.h, 679
_rt_check_ptr_align
l0_runtime.h, 680
_rt_check_ptr_site
l0_runtime.h, 680
_rt_check_ptr_site_slow
l0_runtime.h, 681
_rt_check_record_writeable
l0_runtime.h, 681
_rt_cold_chunks
l0_runtime.h, 720
_rt_drop_begin_impl
l0_runtime.h, 681
_rt_drop_finish_impl
l0_runtime.h, 681
_rt_evict_quarantine
l0_runtime.h, 681
_rt_fmix32
l0_runtime.h, 681
_rt_fold_u64_to_u32_fmix
l0_runtime.h, 682
_rt_free_string
l0_runtime.h, 682
_rt_hash_bool
l0_runtime.h, 682
_rt_hash_byte
l0_runtime.h, 683
_rt_hash_data
l0_runtime.h, 683
_rt_hash_int
l0_runtime.h, 683
_rt_hash_string
l0_runtime.h, 684
_rt_hash_tag8
l0_runtime.h, 684
_rt_iadd
l0_runtime.h, 685
_rt_idiv
l0_runtime.h, 685
_rt_imod
l0_runtime.h, 685
_rt_imul
l0_runtime.h, 686
_rt_index_access_mode
l0_runtime.h, 686
_rt_index_allows_untracked
l0_runtime.h, 686
_rt_index_element_size
l0_runtime.h, 686
_rt_index_offset
l0_runtime.h, 686
_rt_init_args
l0_runtime.h, 686
_rt_init_heap_string
l0_runtime.h, 687
_rt_isub
l0_runtime.h, 687
_rt_l0_string_from_const_literal
l0_runtime.h, 688
_rt_narrow_l0_byte
l0_runtime.h, 688
_rt_new_l0_string
l0_runtime.h, 688
_rt_panic
l0_runtime.h, 689
_rt_panic_fmt
l0_runtime.h, 689
_rt_panic_invalid_access
l0_runtime.h, 689
_rt_panic_invalid_drop
l0_runtime.h, 689
_rt_panic_invalid_release
l0_runtime.h, 689
_rt_pid_to_l0_int
l0_runtime.h, 690
_rt_print
l0_runtime.h, 690
_rt_promote_new_alloc
l0_runtime.h, 690
_rt_ptr_is_aligned
l0_runtime.h, 690
_rt_ptr_site, 70
generation, 71
l0_runtime.h, 676
owner, 71
_rt_ptr_site_store
l0_runtime.h, 690
_rt_quarantine_alloc_record

l0_runtime.h, 691
_rt_quarantine_bytes
l0_runtime.h, 720
_rt_quarantine_count
l0_runtime.h, 720
_rt_quarantine_head
l0_runtime.h, 720
_rt_quarantine_tail
l0_runtime.h, 721
_rt_range_contains
l0_runtime.h, 691
_rt_realloc_string
l0_runtime.h, 691
_rt_realloc_tracked_alloc
l0_runtime.h, 691
_rt_rec_cold
l0_runtime.h, 673
_rt_rec_free_list
l0_runtime.h, 721
_rt_rec_new
l0_runtime.h, 691
_rt_rec_pool_chunks
l0_runtime.h, 721
_rt_rec_recycle
l0_runtime.h, 692
_rt_release_tracked_alloc
l0_runtime.h, 692
_rt_required_access_mode
l0_runtime.h, 692
_rt_required_align
l0_runtime.h, 692
_rt_required_size
l0_runtime.h, 692
_rt_sh_key
l0_runtime.h, 721
_rt_siphash_key_t
l0_runtime.h, 676
_rt_siphash_tag8_t
l0_runtime.h, 676
_rt_stream_write_some
l0_runtime.h, 692
_rt_string_bytes
l0_runtime.h, 693
_rt_time_to_l0_int_nsec
l0_runtime.h, 693
_rt_time_to_l0_int_sec
l0_runtime.h, 693
_rt_time_write_parts
l0_runtime.h, 693
_rt_trace_invalid_drop
l0_runtime.h, 694
_rt_track_alloc_record
l0_runtime.h, 694
_rt_track_alloc_record_kind
l0_runtime.h, 694
_rt_track_arc_bytes
l0_runtime.h, 694
_rt_track_static_bytes
l0_runtime.h, 694
_rt_tree_insert_at
l0_runtime.h, 695
_rt_tree_merge
l0_runtime.h, 695
_rt_tree_prio_for
l0_runtime.h, 695
_rt_tree_remove_at
l0_runtime.h, 695
_rt_untrack_arc_alloc
l0_runtime.h, 695
_rt_validate_derived_ptr
l0_runtime.h, 695
_run_analysis
l0c, 52
_runtime_include_flags
l0c, 52
_runtime_library_flags
l0c, 52
_scan_cli_presentation_options
l0c, 52
_scope_chain_has_cleanup
l0_backend.Backend, 119
_set_alive
l0_expr_types.ExpressionTypeChecker, 260
_should_materialize_arc_temp
l0_backend.Backend, 119
_sim_alloc
hashmap.l0, 521
_sim_find
hashmap.l0, 522
_sim_find_insert
hashmap.l0, 522
_sim_get_hash
hashmap.l0, 522
_sim_get_key
hashmap.l0, 523
_sim_get_state
hashmap.l0, 523
_sim_get_value
hashmap.l0, 523
_sim_needs_grow
hashmap.l0, 524
_sim_rehash
hashmap.l0, 524
_sim_release_key
hashmap.l0, 524
_sim_set_hash
hashmap.l0, 524
_sim_set_key_retain
hashmap.l0, 525
_sim_set_state
hashmap.l0, 525
_sim_set_value
hashmap.l0, 525
_sizeof_expr_for_type
l0_backend.Backend, 120
_skip_lexer_errors

l0_parser.Parser, 388
_skip_ws_and_comments
l0_lexer.Lexer, 319
_span_start
l0_parser.Parser, 388
_split_c_options
l0c, 53
_split_cli_and_program_args
l0c, 53
_spm_alloc
hashmap.l0, 526
_spm_find
hashmap.l0, 526
_spm_find_insert
hashmap.l0, 526
_spm_get_hash
hashmap.l0, 526
_spm_get_key
hashmap.l0, 527
_spm_get_state
hashmap.l0, 527
_spm_get_value
hashmap.l0, 527
_spm_needs_grow
hashmap.l0, 528
_spm_rehash
hashmap.l0, 528
_spm_release_key
hashmap.l0, 528
_spm_set_hash
hashmap.l0, 528
_spm_set_key_retain
hashmap.l0, 529
_spm_set_state
hashmap.l0, 529
_spm_set_value
hashmap.l0, 529
_ss_alloc
hashset.l0, 540
_ss_find
hashset.l0, 542
_ss_find_insert
hashset.l0, 542
_ss_get_hash
hashset.l0, 542
_ss_get_key
hashset.l0, 543
_ss_get_state
hashset.l0, 543
_ss_needs_grow
hashset.l0, 543
_ss_rehash
hashset.l0, 544
_ss_release_key
hashset.l0, 544
_ss_set_hash
hashset.l0, 544
_ss_set_key_retain
hashset.l0, 544
_ss_set_state
hashset.l0, 546
_sslm_entry_by_index
linear_map.l0, 579
_sslm_find_by_key
linear_map.l0, 580
_stmt_contains_try
l0_expr_types.ExpressionTypeChecker, 260
_store_sizeof_target
l0_expr_types.ExpressionTypeChecker, 260
_store_var_resolution
l0_expr_types.ExpressionTypeChecker, 260
_string_token_to_c_bytes_and_len
l0_c_emitter.CEmitter, 163
_suppress_diagnostics
l0_expr_types.ExpressionTypeChecker, 262
_suppress_liveness_diagnostics
l0_expr_types.ExpressionTypeChecker, 263
_sv_less
vector.l0, 626
_switch_depth
l0_backend.Backend, 124
_symbol_is_zero_arg_enum_variant
l0_signatures.SignatureResolver, 419
_sync_case_invalid_arm
l0_parser.Parser, 388
_sync_stmt
l0_parser.Parser, 388
_sync_top_level
l0_parser.Parser, 388
_tmp_counter
l0_c_emitter.CEmitter, 199
_topological_sort
l0_backend.Backend, 120
_try_emit_constructor
l0_backend.Backend, 120
_try_emit_intrinsic
l0_backend.Backend, 121
_try_infer_intrinsic
l0_expr_types.ExpressionTypeChecker, 260
_try_parse_qualified_name
l0_parser.Parser, 388
_try_resolve_type_name
l0_expr_types.ExpressionTypeChecker, 260
_types_equal
l0_backend.Backend, 121
l0_expr_types.ExpressionTypeChecker, 261
_unwrap_opt
l0_runtime.h, 695
_unwrap_ptr
l0_runtime.h, 696
_validate_events
check_trace_log, 23
_validate_match_variant_pattern
l0_expr_types.ExpressionTypeChecker, 261
_validate_mode_scoped_flags
l0c, 53

_validate_reserved_canonical_flags
l0c, 54
_validate_runtime_library_path
l0c, 54
_validated_temporary_directory
l0c, 54
_visit_block
l0_locals.LocalScopeResolver, 328
_visit_case_arm
l0_locals.LocalScopeResolver, 328
_visit_case_else
l0_locals.LocalScopeResolver, 329
_visit_match_arm
l0_locals.LocalScopeResolver, 329
_visit_stmt
l0_locals.LocalScopeResolver, 329
_visit_with_stmt
l0_locals.LocalScopeResolver, 329
_warn
l0_expr_types.ExpressionTypeChecker, 261
_wrap_new_diagnostics
l0_lexer.Lexer, 319
_write_temporary_c_source
l0c, 55

abort
system.l0, 603
abs
integer.l0, 551
active_line_directive
CEmitter, 150
add_declared
l0_scope_context.ScopeContext, 409
add_owned
l0_scope_context.ScopeContext, 410
add_project_root
l0_paths.SourceSearchPaths, 424
add_system_root
l0_paths.SourceSearchPaths, 424
align
_rt_alloc_record_cold, 69
align_down
integer.l0, 552
align_up
integer.l0, 552
alive_scopes
ExpressionTypeChecker, 243
all
l0_symbols.ModuleEnv, 354
ModuleEnv, 356
all_modules
CliOptions, 203
alloc_file
_rt_alloc_record_cold, 69
alloc_line
_rt_alloc_record_cold, 69
ambiguous_imports
l0_symbols.ModuleEnv, 354
ModuleEnv, 356
ambiguous_modules
l0_resolve.SymbolResolution, 459
l0_resolve.TypeResolution, 488
AMBIGUOUS_SYMBOL
l0_resolve.ResolveErrorKind, 401
AMBIGUOUS_TYPE
l0_resolve.TypeResolveErrorKind, 490
AMP
l0_lexer.TokenKind, 466
analysis
CEmitter, 150
l0_backend.Backend, 124
l0_c_emitter.CEmitter, 199
l0_expr_types.ExpressionTypeChecker, 263
analysis.l0
analysis_analyze_entry, 774
analysis_copy_driver_diags, 774
analysis_signature_summary, 774
analysis_analyze_entry
analysis.l0, 774
analysis_copy_driver_diags
analysis.l0, 774
analysis_diag_count
sem_context.l0, 1048
analysis_diag_get
sem_context.l0, 1048
analysis_expr_key
sem_context.l0, 1048
analysis_get_expr_type
sem_context.l0, 1049
analysis_get_intrinsic_target
sem_context.l0, 1049
analysis_get_var_ref_resolution
sem_context.l0, 1049
analysis_has_arc_data
sem_context.l0, 1050
analysis_has_errors
sem_context.l0, 1050
analysis_is_arc_type
sem_context.l0, 1050
analysis_locals
sem_context.l0, 1050
analysis_name_resolution
sem_context.l0, 1051
analysis_owned_type_map_put
sem_context.l0, 1051
analysis_result_free
sem_context.l0, 1051
analysis_signature_summary
analysis.l0, 774
analysis_signature_tables
sem_context.l0, 1053
analysis_source_names
sem_context.l0, 1053
analysis_source_texts
sem_context.l0, 1053
AnalysisResult, 72
diags, 73

driver_state, 73
expr_types, 73
intrinsic_targets, 73
locals_result, 74
names, 74
signatures, 74
var_ref_resolution, 74
analyze
l0_driver.L0Driver, 308
ANDAND
l0_lexer.TokenKind, 466
ap_bool_str
ast_printer.l0, 794
ap_full_name
ast_printer.l0, 794
ap_indent_str
ast_printer.l0, 794
ap_module_prefix
ast_printer.l0, 795
ap_print_enum_decl
ast_printer.l0, 795
ap_print_enum_variant
ast_printer.l0, 795
ap_print_expr
ast_printer.l0, 795
ap_print_field_decl
ast_printer.l0, 796
ap_print_func_decl
ast_printer.l0, 796
ap_print_import
ast_printer.l0, 796
ap_print_label
ast_printer.l0, 796
ap_print_let_decl
ast_printer.l0, 797
ap_print_line
ast_printer.l0, 797
ap_print_module
ast_printer.l0, 797
ap_print_param
ast_printer.l0, 798
ap_print_pattern
ast_printer.l0, 798
ap_print_stmt
ast_printer.l0, 798
ap_print_struct_decl
ast_printer.l0, 798
ap_print_top_level_decl
ast_printer.l0, 799
ap_print_type_alias_decl
ast_printer.l0, 799
ap_print_type_ref
ast_printer.l0, 799
ap_quote_string
ast_printer.l0, 799
ap_span_suffix
ast_printer.l0, 800
append
check_trace_log._MessageCollector, 65
argc
system.l0, 603
args
l0_ast.CallExpr, 137
l0_ast.NewExpr, 363
argv
system.l0, 604
arm
MatchArmScopeEntry, 348
arm_scopes
FunctionEnv, 279
arms
l0_ast.CaseStmt, 143
l0_ast.MatchStmt, 349
arr
VectorBase, 498
arr_check
array.l0, 510
arr_create
array.l0, 510
arr_free
array.l0, 510
arr_get
array.l0, 511
arr_resize
array.l0, 511
arr_zap
array.l0, 511
array
l0_ast.IndexExpr, 292
array.l0
arr_check, 510
arr_create, 510
arr_free, 510
arr_get, 511
arr_resize, 511
arr_zap, 511
ba_capacity, 511
ba_create, 512
ba_free, 512
ba_get, 512
ba_set, 512
ba_zap, 514
ArrayBase, 78
capacity, 78
data, 78
element_size, 78
ARROW_FUNC
l0_lexer.TokenKind, 466
ARROW_MATCH
l0_lexer.TokenKind, 466
AS
l0_lexer.TokenKind, 466
assert
assert.l0, 514
assert.l0
assert, 514

ASSIGN_ACCEPT
expr_types.l0, 938
ASSIGN_REJECT
expr_types.l0, 938
ASSIGN_REPORTED
expr_types.l0, 938
AssignCheckResult
expr_types.l0, 938
ast.l0
case_arm_vec_free, 780
enum_decl_free, 780
enum_variant_free, 780
enum_variant_vec_free, 780
EX_BINARY, 778
EX_BOOL, 778
EX_BYTE, 778
EX_CALL, 778
EX_CAST, 778
EX_FIELD, 778
EX_INDEX, 778
EX_INT, 778
EX_NEW, 778
EX_NULL, 778
EX_PAREN, 778
EX_STRING, 778
EX_TRY, 778
EX_TYPE, 778
EX_UNARY, 778
EX_VAR_REF, 778
expr_arena_add, 780
expr_arena_create, 781
expr_arena_free, 781
expr_arena_get, 781
expr_node_create, 782
expr_node_free, 782
ExprId, 792
ExprKind, 778
field_decl_free, 782
field_decl_vec_free, 782
func_decl_free, 783
id_vec_create, 783
id_vec_free, 783
id_vec_get, 783
id_vec_push, 784
import_vec_free, 784
invalid_expr_id, 784
invalid_pattern_id, 784
invalid_stmt_id, 784
let_decl_free, 785
match_arm_vec_free, 785
module_free, 785
param_free, 785
param_vec_free, 786
pattern_arena_add, 786
pattern_arena_create, 786
pattern_arena_free, 786
pattern_arena_get, 787
pattern_node_create, 787
pattern_node_free, 787
PatternId, 792
PatternKind, 779
PT_VARIANT, 779
PT_WILDCARD, 779
ptr_vec_create, 787
ptr_vec_get, 788
ptr_vec_push, 788
span_here, 788
ST_ASSIGN, 779
ST_BLOCK, 779
ST_BREAK, 779
ST_CASE, 779
ST_CONTINUE, 779
ST_DROP, 779
ST_EXPR, 779
ST_FOR, 779
ST_IF, 779
ST_LET, 779
ST_MATCH, 779
ST_RETURN, 779
ST_WHILE, 779
ST_WITH, 779
stmt_arena_add, 789
stmt_arena_create, 789
stmt_arena_free, 789
stmt_arena_get, 790
stmt_node_create, 790
stmt_node_free, 790
StmtId, 792
StmtKind, 779
struct_decl_free, 790
sv_free_opt, 791
TD_ENUM, 779
TD_FUNC, 779
TD_LET, 779
TD_STRUCT, 779
TD_TYPE_ALIAS, 779
top_level_decl_free, 791
top_level_decl_vec_free, 791
TopLevelDeclKind, 779
type_alias_decl_free, 791
type_ref_free, 791
with_item_vec_free, 792
ast_printer.l0
ap_bool_str, 794
ap_full_name, 794
ap_indent_str, 794
ap_module_prefix, 795
ap_print_enum_decl, 795
ap_print_enum_variant, 795
ap_print_expr, 795
ap_print_field_decl, 796
ap_print_func_decl, 796
ap_print_import, 796
ap_print_label, 796
ap_print_let_decl, 797
ap_print_line, 797

ap_print_module, 797
ap_print_param, 798
ap_print_pattern, 798
ap_print_stmt, 798
ap_print_struct_decl, 798
ap_print_top_level_decl, 799
ap_print_type_alias_decl, 799
ap_print_type_ref, 799
ap_quote_string, 799
ap_span_suffix, 800

ba_capacity
array.l0, 511
ba_create
array.l0, 512
ba_free
array.l0, 512
ba_get
array.l0, 512
ba_set
array.l0, 512
ba_zap
array.l0, 514
backedge_states
LoopFixedPointFlow, 339
LoopIterationFlow, 342
Backend, 81
cfg, 82
current_env, 82
current_func_result, 82
current_loop_cleanup_scopes, 82
current_loop_labels, 82
current_module, 82
current_scope, 82
current_unit, 82
emitter, 82
label_counter, 82
next_stmt_unreachable, 83
opts, 83
result, 83
scope_counter, 83
scope_stack, 83
backend.l0
backend_generate, 805
be_alignof_expr_for_type, 805
be_body_reassigns_param, 806
be_collect_type_dependencies, 806
be_collect_type_keys, 806
be_collect_value_type_dependencies, 807
be_convert_expr_with_expected_type, 807
be_create, 807
be_current_env, 808
be_current_expr_arena, 808
be_current_stmt_arena, 808
be_current_unit, 808
be_dependency_sets_free, 809
be_emit_bare_variant_static_initializer, 809
be_emit_binary_op, 809
be_emit_block_sequence, 810
be_emit_block_stmt, 810
be_emit_call, 810
be_emit_case_literal, 811
be_emit_case_stmt, 811
be_emit_checked_pointer_expr, 811
be_emit_cleanup_at_scope_exit, 812
be_emit_cleanup_for_loop_exit, 812
be_emit_cleanup_for_return, 812
be_emit_condition_branch, 812
be_emit_condition_expr, 813
be_emit_condition_value, 813
be_emit_const_constructor, 813
be_emit_constructor_call, 814
be_emit_copy_expr_with_retains, 814
be_emit_drop_stmt, 814
be_emit_enum_cleanup, 815
be_emit_expr, 815
be_emit_expr_with_expected_type, 815
be_emit_function_declarations, 816
be_emit_function_definitions, 816
be_emit_if_branch, 816
be_emit_inline_with_header_item, 816
be_emit_let_declarations, 817
be_emit_let_initializer, 817
be_emit_line_directive, 817
be_emit_lvalue, 818
be_emit_main_wrapper_if_needed, 818
be_emit_match_stmt, 818
be_emit_new_expr, 818
be_emit_owned_expr_with_expected_type, 819
be_emit_pattern_bindings, 819
be_emit_pointer_index_lvalue, 819
be_emit_retain_for_copied_value, 820
be_emit_return_stmt_with_inline_cleanup, 820
be_emit_static_ord_initializer, 820
be_emit_stmt, 821
be_emit_struct_cleanup, 821
be_emit_type_definitions, 821
be_emit_value_cleanup, 821
be_emit_with_cleanup_from_scope, 822
be_emit_with_cleanup_header_let_assign, 822
be_emit_with_cleanup_header_let_predecl, 822
be_emit_with_stmt, 823
be_expr_type, 823
be_find_enum_decl, 823
be_find_env, 823
be_find_struct_decl, 824
be_find_unit, 824
be_find_variant_decl, 824
be_free, 825
be_fresh_label, 825
be_has_side_effects, 825
be_ice, 826
be_ice_string, 826
be_int_type_size, 826
be_is_int_assignable, 827
be_is_place_expr, 827
be_is_static_let_initializer, 827

be_is_unwrap_cast_from_place, 828
be_lookup_local_var_type, 828
be_lookup_owned_local_name, 828
be_lookup_symbol, 829
be_materialize_arc_temp, 829
be_needs_arc_temp, 829
be_pointer_type_or_null, 830
be_pop_loop_cleanup_scopes, 830
be_pop_loop_labels, 830
be_pop_scope, 830
be_push_loop_cleanup_scopes, 831
be_push_loop_labels, 831
be_push_scope, 831
be_register_inline_with_cleanup, 831
be_resolve_let_type, 832
be_resolve_type_ref, 832
be_scope_chain_has_cleanup, 832
be_should_materialize_arc_temp, 833
be_sizeof_expr_for_type, 833
be_static_variant_symbol, 833
be_type_key, 834
be_type_key_module, 834
be_type_key_name, 834
be_type_ptr_map_temp_free, 834
be_type_ptr_vec_temp_free, 835
type_free_opt, 835
backend_generate
backend.l0, 805
BANG
l0_lexer.TokenKind, 466
base
_rt_alloc_record, 67
IntStringLinearMap, 298
StringStringLinearMap, 443
basename
path.l0, 1114
bd_append_c_option_words
build_driver.l0, 838
bd_build_to_exe
build_driver.l0, 838
bd_cmd_build
build_driver.l0, 838
bd_cmd_run
build_driver.l0, 839
bd_collect_c_option_words
build_driver.l0, 839
bd_compiler_flag_family
build_driver.l0, 839
bd_default_exe_name
build_driver.l0, 839
bd_emit_write_error
build_driver.l0, 840
bd_exe_suffix
build_driver.l0, 840
bd_find_c_compiler
build_driver.l0, 840
bd_find_last_sep
build_driver.l0, 840
bd_format_word_list
build_driver.l0, 841
bd_get_optimize_flag
build_driver.l0, 841
bd_has_debug_flag
build_driver.l0, 841
bd_has_explicit_opt_flag
build_driver.l0, 842
bd_is_windows
build_driver.l0, 842
bd_join_display_words
build_driver.l0, 842
bd_join_shell_words
build_driver.l0, 842
bd_log_effective_roots
build_driver.l0, 843
bd_merge_c_option_words
build_driver.l0, 843
bd_null_device
build_driver.l0, 843
bd_prepare_build_result
build_driver.l0, 843
bd_prepare_input
build_driver.l0, 844
bd_prepared_input_free
build_driver.l0, 844
bd_prepared_input_free_opt
build_driver.l0, 844
bd_push_c_compile_command_words
build_driver.l0, 844
bd_push_output_flags
build_driver.l0, 845
bd_push_standard_flags
build_driver.l0, 845
bd_replay_compile_output
build_driver.l0, 845
bd_runtime_include_dir
build_driver.l0, 846
bd_runtime_lib_dir
build_driver.l0, 846
bd_shell_quote
build_driver.l0, 846
bd_shell_quote_posix
build_driver.l0, 847
bd_shell_quote_windows
build_driver.l0, 847
bd_validate_entry_main
build_driver.l0, 847
bd_validate_runtime_lib
build_driver.l0, 848
bd_windows_exec_token
build_driver.l0, 848
bd_windows_token_is_plain
build_driver.l0, 848
bd_with_c_suffix
build_driver.l0, 849
be_alignof_expr_for_type
backend.l0, 805

be_body_reassigns_param
backend.l0, 806
be_collect_type_dependencies
backend.l0, 806
be_collect_type_keys
backend.l0, 806
be_collect_value_type_dependencies
backend.l0, 807
be_convert_expr_with_expected_type
backend.l0, 807
be_create
backend.l0, 807
be_current_env
backend.l0, 808
be_current_expr_arena
backend.l0, 808
be_current_stmt_arena
backend.l0, 808
be_current_unit
backend.l0, 808
be_dependency_sets_free
backend.l0, 809
be_emit_bare_variant_static_initializer
backend.l0, 809
be_emit_binary_op
backend.l0, 809
be_emit_block_sequence
backend.l0, 810
be_emit_block_stmt
backend.l0, 810
be_emit_call
backend.l0, 810
be_emit_case_literal
backend.l0, 811
be_emit_case_stmt
backend.l0, 811
be_emit_checked_pointer_expr
backend.l0, 811
be_emit_cleanup_at_scope_exit
backend.l0, 812
be_emit_cleanup_for_loop_exit
backend.l0, 812
be_emit_cleanup_for_return
backend.l0, 812
be_emit_condition_branch
backend.l0, 812
be_emit_condition_expr
backend.l0, 813
be_emit_condition_value
backend.l0, 813
be_emit_const_constructor
backend.l0, 813
be_emit_constructor_call
backend.l0, 814
be_emit_copy_expr_with_retains
backend.l0, 814
be_emit_drop_stmt
backend.l0, 814
be_emit_enum_cleanup
backend.l0, 815
be_emit_expr
backend.l0, 815
be_emit_expr_with_expected_type
backend.l0, 815
be_emit_function_declarations
backend.l0, 816
be_emit_function_definitions
backend.l0, 816
be_emit_if_branch
backend.l0, 816
be_emit_inline_with_header_item
backend.l0, 816
be_emit_let_declarations
backend.l0, 817
be_emit_let_initializer
backend.l0, 817
be_emit_line_directive
backend.l0, 817
be_emit_lvalue
backend.l0, 818
be_emit_main_wrapper_if_needed
backend.l0, 818
be_emit_match_stmt
backend.l0, 818
be_emit_new_expr
backend.l0, 818
be_emit_owned_expr_with_expected_type
backend.l0, 819
be_emit_pattern_bindings
backend.l0, 819
be_emit_pointer_index_lvalue
backend.l0, 819
be_emit_retain_for_copied_value
backend.l0, 820
be_emit_return_stmt_with_inline_cleanup
backend.l0, 820
be_emit_static_ord_initializer
backend.l0, 820
be_emit_stmt
backend.l0, 821
be_emit_struct_cleanup
backend.l0, 821
be_emit_type_definitions
backend.l0, 821
be_emit_value_cleanup
backend.l0, 821
be_emit_with_cleanup_from_scope
backend.l0, 822
be_emit_with_cleanup_header_let_assign
backend.l0, 822
be_emit_with_cleanup_header_let_predecl
backend.l0, 822
be_emit_with_stmt
backend.l0, 823
be_expr_type
backend.l0, 823

be_find_enum_decl
backend.l0, 823
be_find_env
backend.l0, 823
be_find_struct_decl
backend.l0, 824
be_find_unit
backend.l0, 824
be_find_variant_decl
backend.l0, 824
be_free
backend.l0, 825
be_fresh_label
backend.l0, 825
be_has_side_effects
backend.l0, 825
be_ice
backend.l0, 826
be_ice_string
backend.l0, 826
be_int_type_size
backend.l0, 826
be_is_int_assignable
backend.l0, 827
be_is_place_expr
backend.l0, 827
be_is_static_let_initializer
backend.l0, 827
be_is_unwrap_cast_from_place
backend.l0, 828
be_lookup_local_var_type
backend.l0, 828
be_lookup_owned_local_name
backend.l0, 828
be_lookup_symbol
backend.l0, 829
be_materialize_arc_temp
backend.l0, 829
be_needs_arc_temp
backend.l0, 829
be_pointer_type_or_null
backend.l0, 830
be_pop_loop_cleanup_scopes
backend.l0, 830
be_pop_loop_labels
backend.l0, 830
be_pop_scope
backend.l0, 830
be_push_loop_cleanup_scopes
backend.l0, 831
be_push_loop_labels
backend.l0, 831
be_push_scope
backend.l0, 831
be_register_inline_with_cleanup
backend.l0, 831
be_resolve_let_type
backend.l0, 832
be_resolve_type_ref
backend.l0, 832
be_scope_chain_has_cleanup
backend.l0, 832
be_should_materialize_arc_temp
backend.l0, 833
be_sizeof_expr_for_type
backend.l0, 833
be_static_variant_symbol
backend.l0, 833
be_type_key
backend.l0, 834
be_type_key_module
backend.l0, 834
be_type_key_name
backend.l0, 834
be_type_ptr_map_temp_free
backend.l0, 834
be_type_ptr_vec_temp_free
backend.l0, 835
binding_slot
LocalSymbol, 333
block_scopes
FunctionEnv, 279
body
CaseArm, 138
CaseElse, 140
FuncDecl, 276
l0_ast.CaseArm, 139
l0_ast.CaseElse, 142
l0_ast.ForStmt, 275
l0_ast.FuncDecl, 278
l0_ast.MatchArm, 346
l0_ast.WhileStmt, 501
l0_ast.WithStmt, 507
MatchArm, 347
body_flow
LoopIterationFlow, 342
bool_to_string
text.l0, 607
bool_type
l0_expr_types.ExpressionTypeChecker, 263
bool_value
ExprNode, 265
BREAK
l0_lexer.TokenKind, 466
break_label
LoopLabels, 343
break_scope
LoopCleanupScopes, 337
break_scope_id
LoopCleanupScopes, 337
break_states
l0_expr_types.LoopFlowCapture, 340
LoopFlowCapture, 341
build_compilation_context
l0c, 55
build_compilation_unit

l0_driver.L0Driver, 309
build_driver.l0
bd_append_c_option_words, 838
bd_build_to_exe, 838
bd_cmd_build, 838
bd_cmd_run, 839
bd_collect_c_option_words, 839
bd_compiler_flag_family, 839
bd_default_exe_name, 839
bd_emit_write_error, 840
bd_exe_suffix, 840
bd_find_c_compiler, 840
bd_find_last_sep, 840
bd_format_word_list, 841
bd_get_optimize_flag, 841
bd_has_debug_flag, 841
bd_has_explicit_opt_flag, 842
bd_is_windows, 842
bd_join_display_words, 842
bd_join_shell_words, 842
bd_log_effective_roots, 843
bd_merge_c_option_words, 843
bd_null_device, 843
bd_prepare_build_result, 843
bd_prepare_input, 844
bd_prepared_input_free, 844
bd_prepared_input_free_opt, 844
bd_push_c_compile_command_words, 844
bd_push_output_flags, 845
bd_push_standard_flags, 845
bd_replay_compile_output, 845
bd_runtime_include_dir, 846
bd_runtime_lib_dir, 846
bd_shell_quote, 846
bd_shell_quote_posix, 847
bd_shell_quote_windows, 847
bd_validate_entry_main, 847
bd_validate_runtime_lib, 848
bd_windows_exec_token, 848
bd_windows_token_is_plain, 848
bd_with_c_suffix, 849
build_info.l0
build_info_build_id, 850
build_info_build_time, 850
build_info_commit, 850
build_info_compiler, 850
build_info_has_embedded_version, 850
build_info_host, 851
build_info_identity_with_version, 851
build_info_release_version, 851
build_info_version_text, 851
build_info_build_id
build_info.l0, 850
build_info_build_time
build_info.l0, 850
build_info_commit
build_info.l0, 850
build_info_compiler
build_info.l0, 850
build_info_has_embedded_version
build_info.l0, 850
build_info_host
build_info.l0, 851
build_info_identity_with_version
build_info.l0, 851
build_info_release_version
build_info.l0, 851
build_info_version_text
build_info.l0, 851
build_search_paths
l0c, 55
BuildPreparedInput, 131
entry_module, 131
search_paths, 131
by_name
LocalsResult, 331
NameResolution, 358
BYTE
l0_lexer.TokenKind, 466
byte_to_s
string.l0, 594
byte_to_string
text.l0, 608
byte_to_string_base
text.l0, 608
byte_type
l0_expr_types.ExpressionTypeChecker, 263
ByteArray, 133
storage, 134
bytes
_l0_h_string, 64
l0_string, 303
bytes_to_s
string.l0, 595

c_compiler
CliOptions, 203
c_emitter.l0
ccb_create, 860
ccb_dedent, 860
ccb_emit, 860
ccb_emit_raw, 861
ccb_free, 861
ccb_indent, 861
ccb_prefix, 861
ccb_to_string, 862
cem_analysis, 862
cem_collect_opt_wrappers_from_type, 862
cem_create, 862
cem_emit_alloc_obj, 863
cem_emit_assignment, 863
cem_emit_binary_op, 863
cem_emit_block_end, 864
cem_emit_block_start, 864
cem_emit_bool_literal, 864
cem_emit_byte_literal, 864
cem_emit_case_label, 865

cem_emit_cast, 865
cem_emit_checked_int_add, 865
cem_emit_checked_int_div, 866
cem_emit_checked_int_mod, 866
cem_emit_checked_int_mul, 866
cem_emit_checked_int_sub, 867
cem_emit_checked_narrow_cast, 867
cem_emit_checked_pointer_assignment, 867
cem_emit_checked_ptr_access, 868
cem_emit_checked_ptr_access_for_base, 868
cem_emit_checked_ptr_index_access, 868
cem_emit_comment, 869
cem_emit_condition_binary_op, 869
cem_emit_condition_pointer_null_check, 869
cem_emit_condition_string_compare_call, 870
cem_emit_const_bool_literal, 870
cem_emit_const_string_literal, 870
cem_emit_default_label, 871
cem_emit_deref_lvalue, 871
cem_emit_discard_expr, 871
cem_emit_drop_begin_expr, 871
cem_emit_drop_finish_call, 872
cem_emit_else, 872
cem_emit_enum, 872
cem_emit_enum_cleanup, 873
cem_emit_enum_payload_field_access, 873
cem_emit_enum_tag, 873
cem_emit_enum_tag_access, 874
cem_emit_enum_variant_init, 874
cem_emit_exit_switch, 874
cem_emit_expr_stmt, 874
cem_emit_field_access, 875
cem_emit_field_lvalue, 875
cem_emit_for_loop_end, 875
cem_emit_for_loop_start, 876
cem_emit_forward_decls, 876
cem_emit_function_call, 876
cem_emit_function_declaration, 876
cem_emit_function_definition_footer, 877
cem_emit_function_definition_header, 877
cem_emit_goto, 877
cem_emit_header, 877
cem_emit_if_header, 878
cem_emit_index_lvalue, 878
cem_emit_int_literal, 878
cem_emit_label, 878
cem_emit_let_decl, 879
cem_emit_let_declaration, 879
cem_emit_line_directive, 879
cem_emit_main_wrapper, 880
cem_emit_match_scrutinee_decl, 880
cem_emit_match_switch_start, 880
cem_emit_module_comment, 880
cem_emit_module_separator, 881
cem_emit_negated_condition, 881
cem_emit_none_value_for_nullable, 881
cem_emit_null_assignment, 881
cem_emit_null_check_eq, 882
cem_emit_null_check_ne, 882
cem_emit_null_literal, 882
cem_emit_optional_has_value, 883
cem_emit_optional_value, 883
cem_emit_optional_wrapper, 883
cem_emit_optional_wrapper_for_defined_type, 884
cem_emit_optional_wrappers, 884
cem_emit_ord, 884
cem_emit_paren_expr, 884
cem_emit_pattern_binding_init, 885
cem_emit_pointer_assignment, 885
cem_emit_pointer_null_check, 885
cem_emit_pointer_type, 886
cem_emit_return_stmt, 886
cem_emit_return_void_stmt, 886
cem_emit_section_comment, 886
cem_emit_sizeof_type, 887
cem_emit_some_value_for_nullable, 887
cem_emit_string_compare_call, 887
cem_emit_string_concat_call, 888
cem_emit_string_equals_call, 888
cem_emit_string_literal, 888
cem_emit_string_release, 889
cem_emit_string_retain, 889
cem_emit_struct, 889
cem_emit_struct_cleanup, 889
cem_emit_struct_constructor, 890
cem_emit_struct_constructor_for_type, 890
cem_emit_struct_init, 890
cem_emit_struct_init_from_fields, 891
cem_emit_struct_static_initializer, 891
cem_emit_struct_static_initializer_for_type, 891
cem_emit_switch_end, 892
cem_emit_switch_start, 892
cem_emit_temp_decl, 892
cem_emit_top_level_let_cleanup, 892
cem_emit_try_check_niche, 893
cem_emit_try_check_value, 893
cem_emit_try_extract_value, 893
cem_emit_type, 893
cem_emit_unary_op, 895
cem_emit_unreachable_comment, 895
cem_emit_unreachable_marker, 895
cem_emit_unwrap_opt, 895
cem_emit_unwrap_ptr, 896
cem_emit_value_cleanup, 896
cem_emit_var_ref, 896
cem_emit_variant_constructor, 897
cem_emit_variant_constructor_for_type, 897
cem_emit_variant_static_initializer, 898
cem_emit_variant_static_initializer_for_type, 898
cem_emit_while_header, 898
cem_emit_widen_int, 899
cem_emit_zero_init, 899
cem_encode_c_byte_literal, 899
cem_enum_has_arc_data, 899
cem_find_enum_decl, 900

cem_find_enum_variant_info, 900
cem_free, 900
cem_fresh_tmp, 901
cem_function_params, 901
cem_get_output, 901
cem_ice, 902
cem_ice_string, 902
cem_is_c_keyword, 902
cem_is_early_inner, 902
cem_is_niche_nullable, 903
cem_mangle_enum_name, 903
cem_mangle_function_name, 903
cem_mangle_identifier, 904
cem_mangle_let_name, 904
cem_mangle_module, 904
cem_mangle_struct_name, 904
cem_module_filename, 905
cem_note_opt_wrapper, 905
cem_opt_key_for_type, 905
cem_opt_wrapper_map_free, 906
cem_opt_wrapper_name_for_inner, 906
cem_prepare_optional_wrappers, 906
cem_reset_optional_wrappers, 906
cem_restore_active_line_directive, 907
cem_set_analysis, 907
cem_upper_ascii, 907
C_KEYWORDS
l0_c_emitter.CEmitter, 199
c_options
CliOptions, 204
c_sources
CliOptions, 204
cached_level
CCodeBuilder, 146
cached_prefix
CCodeBuilder, 146
callee
l0_ast.CallExpr, 137
capacity
ArrayBase, 78
IntSet, 297
StringIntMap, 437
StringPtrMap, 440
StringSet, 441
capture
LoopFixedPointFlow, 339
LoopIterationFlow, 342
CARET
l0_lexer.TokenKind, 467
CASE
l0_lexer.TokenKind, 467
case_arm_vec_free
ast.l0, 780
case_arms
StmtNode, 434
case_else
StmtNode, 434
CaseArm, 137
body, 138
literal, 138
span, 138
CaseElse, 140
body, 140
span, 140
cb_append
text.l0, 608
cb_append_int
text.l0, 609
cb_append_s
text.l0, 609
cb_append_slice
text.l0, 609
cb_capacity
text.l0, 609
cb_clear
text.l0, 610
cb_create
text.l0, 610
cb_free
text.l0, 610
cb_reserve
text.l0, 610
cb_reverse
text.l0, 611
cb_size
text.l0, 611
cb_to_string
text.l0, 611
ccb_create
c_emitter.l0, 860
ccb_dedent
c_emitter.l0, 860
ccb_emit
c_emitter.l0, 860
ccb_emit_raw
c_emitter.l0, 861
ccb_free
c_emitter.l0, 861
ccb_indent
c_emitter.l0, 861
ccb_prefix
c_emitter.l0, 861
ccb_to_string
c_emitter.l0, 862
CCodeBuilder, 146
cached_level, 146
cached_prefix, 146
indent_level, 146
indent_str, 146
lines, 146
cem_analysis
c_emitter.l0, 862
cem_collect_opt_wrappers_from_type
c_emitter.l0, 862
cem_create
c_emitter.l0, 862

cem_emit_alloc_obj
c_emitter.l0, 863
cem_emit_assignment
c_emitter.l0, 863
cem_emit_binary_op
c_emitter.l0, 863
cem_emit_block_end
c_emitter.l0, 864
cem_emit_block_start
c_emitter.l0, 864
cem_emit_bool_literal
c_emitter.l0, 864
cem_emit_byte_literal
c_emitter.l0, 864
cem_emit_case_label
c_emitter.l0, 865
cem_emit_cast
c_emitter.l0, 865
cem_emit_checked_int_add
c_emitter.l0, 865
cem_emit_checked_int_div
c_emitter.l0, 866
cem_emit_checked_int_mod
c_emitter.l0, 866
cem_emit_checked_int_mul
c_emitter.l0, 866
cem_emit_checked_int_sub
c_emitter.l0, 867
cem_emit_checked_narrow_cast
c_emitter.l0, 867
cem_emit_checked_pointer_assignment
c_emitter.l0, 867
cem_emit_checked_ptr_access
c_emitter.l0, 868
cem_emit_checked_ptr_access_for_base
c_emitter.l0, 868
cem_emit_checked_ptr_index_access
c_emitter.l0, 868
cem_emit_comment
c_emitter.l0, 869
cem_emit_condition_binary_op
c_emitter.l0, 869
cem_emit_condition_pointer_null_check
c_emitter.l0, 869
cem_emit_condition_string_compare_call
c_emitter.l0, 870
cem_emit_const_bool_literal
c_emitter.l0, 870
cem_emit_const_string_literal
c_emitter.l0, 870
cem_emit_default_label
c_emitter.l0, 871
cem_emit_deref_lvalue
c_emitter.l0, 871
cem_emit_discard_expr
c_emitter.l0, 871
cem_emit_drop_begin_expr
c_emitter.l0, 871
cem_emit_drop_finish_call
c_emitter.l0, 872
cem_emit_else
c_emitter.l0, 872
cem_emit_enum
c_emitter.l0, 872
cem_emit_enum_cleanup
c_emitter.l0, 873
cem_emit_enum_payload_field_access
c_emitter.l0, 873
cem_emit_enum_tag
c_emitter.l0, 873
cem_emit_enum_tag_access
c_emitter.l0, 874
cem_emit_enum_variant_init
c_emitter.l0, 874
cem_emit_exit_switch
c_emitter.l0, 874
cem_emit_expr_stmt
c_emitter.l0, 874
cem_emit_field_access
c_emitter.l0, 875
cem_emit_field_lvalue
c_emitter.l0, 875
cem_emit_for_loop_end
c_emitter.l0, 875
cem_emit_for_loop_start
c_emitter.l0, 876
cem_emit_forward_decls
c_emitter.l0, 876
cem_emit_function_call
c_emitter.l0, 876
cem_emit_function_declaration
c_emitter.l0, 876
cem_emit_function_definition_footer
c_emitter.l0, 877
cem_emit_function_definition_header
c_emitter.l0, 877
cem_emit_goto
c_emitter.l0, 877
cem_emit_header
c_emitter.l0, 877
cem_emit_if_header
c_emitter.l0, 878
cem_emit_index_lvalue
c_emitter.l0, 878
cem_emit_int_literal
c_emitter.l0, 878
cem_emit_label
c_emitter.l0, 878
cem_emit_let_decl
c_emitter.l0, 879
cem_emit_let_declaration
c_emitter.l0, 879
cem_emit_line_directive
c_emitter.l0, 879
cem_emit_main_wrapper
c_emitter.l0, 880

cem_emit_match_scrutinee_decl
c_emitter.l0, 880
cem_emit_match_switch_start
c_emitter.l0, 880
cem_emit_module_comment
c_emitter.l0, 880
cem_emit_module_separator
c_emitter.l0, 881
cem_emit_negated_condition
c_emitter.l0, 881
cem_emit_none_value_for_nullable
c_emitter.l0, 881
cem_emit_null_assignment
c_emitter.l0, 881
cem_emit_null_check_eq
c_emitter.l0, 882
cem_emit_null_check_ne
c_emitter.l0, 882
cem_emit_null_literal
c_emitter.l0, 882
cem_emit_optional_has_value
c_emitter.l0, 883
cem_emit_optional_value
c_emitter.l0, 883
cem_emit_optional_wrapper
c_emitter.l0, 883
cem_emit_optional_wrapper_for_defined_type
c_emitter.l0, 884
cem_emit_optional_wrappers
c_emitter.l0, 884
cem_emit_ord
c_emitter.l0, 884
cem_emit_paren_expr
c_emitter.l0, 884
cem_emit_pattern_binding_init
c_emitter.l0, 885
cem_emit_pointer_assignment
c_emitter.l0, 885
cem_emit_pointer_null_check
c_emitter.l0, 885
cem_emit_pointer_type
c_emitter.l0, 886
cem_emit_return_stmt
c_emitter.l0, 886
cem_emit_return_void_stmt
c_emitter.l0, 886
cem_emit_section_comment
c_emitter.l0, 886
cem_emit_sizeof_type
c_emitter.l0, 887
cem_emit_some_value_for_nullable
c_emitter.l0, 887
cem_emit_string_compare_call
c_emitter.l0, 887
cem_emit_string_concat_call
c_emitter.l0, 888
cem_emit_string_equals_call
c_emitter.l0, 888
cem_emit_string_literal
c_emitter.l0, 888
cem_emit_string_release
c_emitter.l0, 889
cem_emit_string_retain
c_emitter.l0, 889
cem_emit_struct
c_emitter.l0, 889
cem_emit_struct_cleanup
c_emitter.l0, 889
cem_emit_struct_constructor
c_emitter.l0, 890
cem_emit_struct_constructor_for_type
c_emitter.l0, 890
cem_emit_struct_init
c_emitter.l0, 890
cem_emit_struct_init_from_fields
c_emitter.l0, 891
cem_emit_struct_static_initializer
c_emitter.l0, 891
cem_emit_struct_static_initializer_for_type
c_emitter.l0, 891
cem_emit_switch_end
c_emitter.l0, 892
cem_emit_switch_start
c_emitter.l0, 892
cem_emit_temp_decl
c_emitter.l0, 892
cem_emit_top_level_let_cleanup
c_emitter.l0, 892
cem_emit_try_check_niche
c_emitter.l0, 893
cem_emit_try_check_value
c_emitter.l0, 893
cem_emit_try_extract_value
c_emitter.l0, 893
cem_emit_type
c_emitter.l0, 893
cem_emit_unary_op
c_emitter.l0, 895
cem_emit_unreachable_comment
c_emitter.l0, 895
cem_emit_unreachable_marker
c_emitter.l0, 895
cem_emit_unwrap_opt
c_emitter.l0, 895
cem_emit_unwrap_ptr
c_emitter.l0, 896
cem_emit_value_cleanup
c_emitter.l0, 896
cem_emit_var_ref
c_emitter.l0, 896
cem_emit_variant_constructor
c_emitter.l0, 897
cem_emit_variant_constructor_for_type
c_emitter.l0, 897
cem_emit_variant_static_initializer
c_emitter.l0, 898

cem_emit_variant_static_initializer_for_type
c_emitter.l0, 898
cem_emit_while_header
c_emitter.l0, 898
cem_emit_widen_int
c_emitter.l0, 899
cem_emit_zero_init
c_emitter.l0, 899
cem_encode_c_byte_literal
c_emitter.l0, 899
cem_enum_has_arc_data
c_emitter.l0, 899
cem_find_enum_decl
c_emitter.l0, 900
cem_find_enum_variant_info
c_emitter.l0, 900
cem_free
c_emitter.l0, 900
cem_fresh_tmp
c_emitter.l0, 901
cem_function_params
c_emitter.l0, 901
cem_get_output
c_emitter.l0, 901
cem_ice
c_emitter.l0, 902
cem_ice_string
c_emitter.l0, 902
cem_is_c_keyword
c_emitter.l0, 902
cem_is_early_inner
c_emitter.l0, 902
cem_is_niche_nullable
c_emitter.l0, 903
cem_mangle_enum_name
c_emitter.l0, 903
cem_mangle_function_name
c_emitter.l0, 903
cem_mangle_identifier
c_emitter.l0, 904
cem_mangle_let_name
c_emitter.l0, 904
cem_mangle_module
c_emitter.l0, 904
cem_mangle_struct_name
c_emitter.l0, 904
cem_module_filename
c_emitter.l0, 905
cem_note_opt_wrapper
c_emitter.l0, 905
cem_opt_key_for_type
c_emitter.l0, 905
cem_opt_wrapper_map_free
c_emitter.l0, 906
cem_opt_wrapper_name_for_inner
c_emitter.l0, 906
cem_prepare_optional_wrappers
c_emitter.l0, 906
cem_reset_optional_wrappers
c_emitter.l0, 906
cem_restore_active_line_directive
c_emitter.l0, 907
cem_set_analysis
c_emitter.l0, 907
cem_upper_ascii
c_emitter.l0, 907
CEmitter, 150
active_line_directive, 150
analysis, 150
current_module, 150
opt_emitted, 151
opt_wrappers, 151
out, 151
tmp_counter, 151
cfg
Backend, 82
cfs_apply_cleanup_status
compiler_filesystem.l0, 920
cfs_error
compiler_filesystem.l0, 920
cfs_join_child
compiler_filesystem.l0, 920
cfs_mkdir
compiler_filesystem.l0, 920
CFS_PATH_ABSENT
compiler_filesystem.l0, 923
CFS_PATH_DIRECTORY
compiler_filesystem.l0, 923
cfs_path_kind
compiler_filesystem.l0, 920
cfs_path_kind_follow
compiler_filesystem.l0, 920
CFS_PATH_OTHER
compiler_filesystem.l0, 923
CFS_PATH_REGULAR
compiler_filesystem.l0, 923
cfs_remove_empty_dir
compiler_filesystem.l0, 920
cfs_remove_regular_file
compiler_filesystem.l0, 920
cfs_resolve_trusted_temp_parent
compiler_filesystem.l0, 920
cfs_select_temp_parent
compiler_filesystem.l0, 921
cfs_select_temp_parent_from_values
compiler_filesystem.l0, 921
cfs_workspace_child
compiler_filesystem.l0, 921
cfs_workspace_cleanup
compiler_filesystem.l0, 921
cfs_workspace_create
compiler_filesystem.l0, 921
cfs_workspace_create_for_identity
compiler_filesystem.l0, 922
cfs_workspace_free
compiler_filesystem.l0, 922

cg_options_create
    codegen_options.l0, 918
cg_options_free
    codegen_options.l0, 918
char_at_s
    string.l0, 595
char_value
    EscapedChar, 236
CharBuffer, 200
    chars, 201
chars
    CharBuffer, 201
check
    l0_expr_types.ExpressionTypeChecker, 261
check_basic
    CliOptions, 204
check_trace_log, 21
    _build_arg_parser, 22
    _iter_events, 22
    _print_report, 22
    _validate_events, 23
    KV_RE, 24
    main, 23
    TRACE_RE, 24
check_trace_log._MessageCollector, 64
    __init__, 65
    append, 65
    limit, 65
    messages, 65
    total, 65
checked_add_int
    integer.l0, 552
checked_mul_int
    integer.l0, 553
checked_sub_int
    integer.l0, 553
children
    CompilerWorkspace, 213
civil_from_unix_days
    time.l0, 620
CivilDate, 201
    day, 201
    month, 201
    weekday, 201
    year, 201
    yearday, 201
clamp
    integer.l0, 553
CLEANUP
    l0_lexer.TokenKind, 467
cleanup
    l0_ast.WithItem, 504
cleanup_body
    l0_ast.WithStmt, 507
cleanup_header_guard_stack
    ExpressionTypeChecker, 243
cleanup_stmt
    WithItem, 505
CleanupHeaderGuard, 202
    guarded_decl_ids, 202
    parent, 202
cli_args.l0
    cli_assign_target, 909
    cli_compiler_identity_text, 911
    cli_diag_free_opt, 911
    cli_emit_error, 911
    cli_emit_reserved_option_error, 911
    cli_has_diagnostic_code, 912
    cli_has_help_flag, 912
    cli_has_version_flag, 912
    cli_mode_flag, 913
    cli_mode_name, 913
    cli_options_create, 913
    cli_options_free, 913
    cli_options_free_opt, 914
    cli_parse, 914
    cli_parse_result_free, 914
    cli_print_blank_line, 914
    cli_print_help, 915
    cli_print_line, 915
    cli_print_usage, 915
    cli_print_version, 915
    cli_require_next_value, 915
    cli_split_index, 916
    cli_try_set_mode, 916
    cli_validate_mode_scoped_flags, 916
    cli_verbose_cluster_count, 917
    CliMode, 909
    CM_AST, 909
    CM_BUILD, 909
    CM_CHECK, 909
    CM_COMPILE, 909
    CM_GEN, 909
    CM_RUN, 909
    CM_SYM, 909
    CM_TOK, 909
    CM_TYPE, 909
cli_assign_target
    cli_args.l0, 909
cli_compiler_identity_text
    cli_args.l0, 911
cli_diag_free_opt
    cli_args.l0, 911
cli_emit_error
    cli_args.l0, 911
cli_emit_reserved_option_error
    cli_args.l0, 911
cli_has_diagnostic_code
    cli_args.l0, 912
cli_has_help_flag
    cli_args.l0, 912
cli_has_version_flag
    cli_args.l0, 912
cli_mode_flag
    cli_args.l0, 913
cli_mode_name

cli_args.l0, 913
cli_options_create
cli_args.l0, 913
cli_options_free
cli_args.l0, 913
cli_options_free_opt
cli_args.l0, 914
cli_parse
cli_args.l0, 914
cli_parse_result_free
cli_args.l0, 914
cli_print_blank_line
cli_args.l0, 914
cli_print_help
cli_args.l0, 915
cli_print_line
cli_args.l0, 915
cli_print_usage
cli_args.l0, 915
cli_print_version
cli_args.l0, 915
cli_require_next_value
cli_args.l0, 915
cli_split_index
cli_args.l0, 916
cli_try_set_mode
cli_args.l0, 916
cli_validate_mode_scoped_flags
cli_args.l0, 916
cli_verbose_cluster_count
cli_args.l0, 917
CliMode
cli_args.l0, 909
CliOptions, 203
all_modules, 203
c_compiler, 203
c_options, 204
c_sources, 204
check_basic, 204
entry_target, 204
extra_target_count, 204
include_eof, 204
interface_paths, 204
keep_c, 204
log, 204
mode, 204
mode_explicit, 205
no_line_directives, 205
output, 205
program_args, 205
project_roots, 205
runtime_include, 205
runtime_lib, 205
sys_roots, 205
trace_arc, 205
trace_memory, 205
unchecked, 206
verbosity, 206
CliParseResult, 206
diags, 207
help_requested, 207
options, 207
rich_log, 207
verbosity, 207
version_requested, 207
CM_AST
cli_args.l0, 909
CM_BUILD
cli_args.l0, 909
CM_CHECK
cli_args.l0, 909
CM_COMPILE
cli_args.l0, 909
CM_GEN
cli_args.l0, 909
CM_RUN
cli_args.l0, 909
CM_SYM
cli_args.l0, 909
CM_TOK
cli_args.l0, 909
CM_TYPE
cli_args.l0, 909
cmd_ast
l0c, 56
cmd_build
l0c, 56
cmd_check
l0c, 56
cmd_codegen
l0c, 57
cmd_compile
l0c, 57
cmd_run
l0c, 57
cmd_sym
l0c, 58
cmd_tok
l0c, 58
cmd_type
l0c, 58
cmp_s
string.l0, 595
code
Diagnostic, 217
l0_string_escape.EscapeDecodeError, 238
codegen_options.l0
cg_options_create, 918
cg_options_free, 918
CodegenOptions, 207
emit_line_directives, 208
rt_check_basic, 208
rt_unchecked, 208
trace_arc, 208
trace_memory, 208
cold_index

_rt_alloc_record, 67
COLON
l0_lexer.TokenKind, 467
column
Diagnostic, 217
l0_diagnostics.Diagnostic, 219
l0_lexer.Lexer, 320
l0_lexer.Token, 460
LexerState, 322
Token, 462
column_end
Diagnostic, 217
COMMA
l0_lexer.TokenKind, 467
compiler/shared/l0/stdlib/std/array.l0, 509
compiler/shared/l0/stdlib/std/assert.l0, 514
compiler/shared/l0/stdlib/std/fs.l0, 515
compiler/shared/l0/stdlib/std/hashmap.l0, 518
compiler/shared/l0/stdlib/std/hashset.l0, 539
compiler/shared/l0/stdlib/std/integer.l0, 550
compiler/shared/l0/stdlib/std/io.l0, 559
compiler/shared/l0/stdlib/std/linear_map.l0, 574
compiler/shared/l0/stdlib/std/optional.l0, 589
compiler/shared/l0/stdlib/std/path.l0, 1112
compiler/shared/l0/stdlib/std/rand.l0, 591
compiler/shared/l0/stdlib/std/string.l0, 593
compiler/shared/l0/stdlib/std/system.l0, 603
compiler/shared/l0/stdlib/std/text.l0, 605
compiler/shared/l0/stdlib/std/time.l0, 619
compiler/shared/l0/stdlib/std/unit.l0, 624
compiler/shared/l0/stdlib/std/vector.l0, 625
compiler/shared/l0/stdlib/sys/hash.l0, 633
compiler/shared/l0/stdlib/sys/memory.l0, 638
compiler/shared/l0/stdlib/sys/rt.l0, 643
compiler/shared/runtime/dea_siphash.h, 656, 660
compiler/shared/runtime/l0_runtime.h, 662, 722
compiler/stage1_py/l0_analysis.py, 759
compiler/stage1_py/l0_ast.py, 759
compiler/stage1_py/l0_ast_printer.py, 762
compiler/stage1_py/l0_backend.py, 762
compiler/stage1_py/l0_c_emitter.py, 762
compiler/stage1_py/l0_compilation.py, 763
compiler/stage1_py/l0_context.py, 763
compiler/stage1_py/l0_diagnostics.py, 763
compiler/stage1_py/l0_driver.py, 764
compiler/stage1_py/l0_expr_types.py, 764
compiler/stage1_py/l0_internal_error.py, 764
compiler/stage1_py/l0_lexer.py, 765
compiler/stage1_py/l0_locals.py, 765
compiler/stage1_py/l0_logger.py, 766
compiler/stage1_py/l0_name_resolver.py, 766
compiler/stage1_py/l0_parser.py, 766
compiler/stage1_py/l0_paths.py, 767
compiler/stage1_py/l0_resolve.py, 767
compiler/stage1_py/l0_scope_context.py, 768
compiler/stage1_py/l0_signatures.py, 768
compiler/stage1_py/l0_string_escape.py, 768
compiler/stage1_py/l0_symbols.py, 769
compiler/stage1_py/l0_types.py, 769
compiler/stage1_py/l0c.py, 770
compiler/stage2_l0/scripts/check_trace_log.py, 772
compiler/stage2_l0/src/analysis.l0, 773
compiler/stage2_l0/src/ast.l0, 774
compiler/stage2_l0/src/ast_printer.l0, 792
compiler/stage2_l0/src/backend.l0, 800
compiler/stage2_l0/src/build_driver.l0, 835
compiler/stage2_l0/src/build_info.l0, 849
compiler/stage2_l0/src/c_emitter.l0, 853
compiler/stage2_l0/src/cli_args.l0, 907
compiler/stage2_l0/src/codegen_options.l0, 917
compiler/stage2_l0/src/compiler_filesystem.l0, 918
compiler/stage2_l0/src/diag_print.l0, 923
compiler/stage2_l0/src/driver.l0, 926
compiler/stage2_l0/src/expr_types.l0, 933
compiler/stage2_l0/src/l0c.l0, 964
compiler/stage2_l0/src/l0c_lib.l0, 965
compiler/stage2_l0/src/lexer.l0, 973
compiler/stage2_l0/src/locals.l0, 984
compiler/stage2_l0/src/name_resolver.l0, 993
compiler/stage2_l0/src/parser.l0, 1003
compiler/stage2_l0/src/parser/decl.l0, 1006
compiler/stage2_l0/src/parser/expr.l0, 1012
compiler/stage2_l0/src/parser/shared.l0, 1017
compiler/stage2_l0/src/parser/stmt.l0, 1036
compiler/stage2_l0/src/scope_context.l0, 1043
compiler/stage2_l0/src/sem_context.l0, 1046
compiler/stage2_l0/src/signatures.l0, 1054
compiler/stage2_l0/src/source_paths.l0, 1062
compiler/stage2_l0/src/string_escape.l0, 1069
compiler/stage2_l0/src/symbols.l0, 1071
compiler/stage2_l0/src/tokens.l0, 1073
compiler/stage2_l0/src/type_resolve.l0, 1082
compiler/stage2_l0/src/types.l0, 1087
compiler/stage2_l0/src/util/diag.l0, 1095
compiler/stage2_l0/src/util/intset.l0, 1101
compiler/stage2_l0/src/util/log.l0, 1108
compiler/stage2_l0/src/util/path.l0, 1116
compiler/stage2_l0/src/util/strings.l0, 1119
compiler_filesystem.l0
cfs_apply_cleanup_status, 920
cfs_error, 920
cfs_join_child, 920
cfs_mkdir, 920
CFS_PATH_ABSENT, 923
CFS_PATH_DIRECTORY, 923
cfs_path_kind, 920
cfs_path_kind_follow, 920
CFS_PATH_OTHER, 923
CFS_PATH_REGULAR, 923
cfs_remove_empty_dir, 920
cfs_remove_regular_file, 920
cfs_resolve_trusted_temp_parent, 920
cfs_select_temp_parent, 921
cfs_select_temp_parent_from_values, 921
cfs_workspace_child, 921
cfs_workspace_cleanup, 921

cfs_workspace_create, 921
cfs_workspace_create_for_identity, 922
cfs_workspace_free, 922
l0c_fs_join_child, 922
l0c_fs_mkdir, 922
l0c_fs_path_kind_follow, 922
l0c_fs_path_kind_nofollow, 922
l0c_fs_remove_empty_dir, 922
l0c_fs_remove_regular_file, 922
l0c_fs_resolve_trusted_temp_parent, 923
compiler_identity_text
l0c, 59
CompilerWorkspace, 212
children, 213
root, 213
concat_s
string.l0, 596
cond
l0_ast.ForStmt, 275
l0_ast.IfStmt, 285
l0_ast.WhileStmt, 501
contains_s
string.l0, 596
context
l0_analysis.AnalysisResult, 77
l0_driver.L0Driver, 310
CONTINUE
l0_lexer.TokenKind, 467
continue_label
LoopLabels, 343
continue_scope
LoopCleanupScopes, 337
continue_scope_id
LoopCleanupScopes, 338
continue_states
l0_expr_types.LoopFlowCapture, 340
LoopFlowCapture, 341
count
IntSet, 297
StringIntMap, 437
StringPtrMap, 440
StringSet, 441
cu
l0_analysis.AnalysisResult, 77
l0_name_resolver.NameResolver, 361
l0_signatures.SignatureResolver, 420
current_env
Backend, 82
ExpressionTypeChecker, 243
current_func
ExpressionTypeChecker, 243
current_func_result
Backend, 82
current_loop_cleanup_scopes
Backend, 82
current_loop_labels
Backend, 82
current_module
Backend, 82
CEmitter, 150
ExpressionTypeChecker, 243
l0_backend.Backend, 124
l0_c_emitter.CEmitter, 199
current_ret_type
ExpressionTypeChecker, 243
current_scope
Backend, 82
ExpressionTypeChecker, 243
current_unit
Backend, 82
ExpressionTypeChecker, 243

data
ArrayBase, 78
l0_string, 303
TopLevelDecl, 475
DateTime, 214
day, 215
hour, 215
is_dst, 215
minute, 215
month, 215
nanosecond, 215
second, 215
utc_offset_sec, 216
weekday, 216
year, 216
yearday, 216
datetime_from_unix_sec
time.l0, 620
day
CivilDate, 201
DateTime, 215
day_of_year
time.l0, 621
days_in_month
time.l0, 621
Dea/L0 API Reference, 1
DEA_RUNTIME_FCLOSE
l0_runtime.h, 674
DEA_RUNTIME_FILE_INFO_STAT
l0_runtime.h, 674
DEA_RUNTIME_REMOVE
l0_runtime.h, 674
dea_siphash.h
siphash13, 657
siphash13_tag8_bf, 657
siphash13_tag8_bytes, 658
siphash24, 658
siphash24_tag8_bf, 659
siphash24_tag8_bytes, 659
DEBUG
l0_context.LogLevel, 336
decl
l0_locals.LocalSymbol, 332
decl.l0
impl_parse_module_source, 1007

impl_parse_module_tokens, 1007
ps_finish_function, 1008
ps_lex_error_result, 1008
ps_make_empty_block_stmt, 1008
ps_parse_enum, 1009
ps_parse_extern_func, 1009
ps_parse_function, 1009
ps_parse_module, 1010
ps_parse_struct, 1010
ps_parse_top_level_decl, 1010
ps_parse_top_level_let, 1011
ps_parse_type_alias, 1011
ps_tokens_result, 1011
decl_id
LocalSymbol, 333
decl_ptr
LocalSymbol, 333
Symbol, 455
decl_scope_defs
ExpressionTypeChecker, 243
decl_scopes
ExpressionTypeChecker, 243
declared_vars
l0_scope_context.ScopeContext, 410
ScopeContext, 412
decls
l0_ast.Module, 351
Module, 352
decode_l0_string_token
l0_string_escape, 39
dedent
l0_c_emitter.CCodeBuilder, 148
default
l0_context.CompilationContext, 210
delete_file
fs.l0, 516
details
l0_string_escape.EscapeDecodeError, 238
dgv_create
diag.l0, 1097
dgv_free
diag.l0, 1097
dgv_get
diag.l0, 1097
dgv_length
diag.l0, 1097
dgv_push
diag.l0, 1098
diag.l0
dgv_create, 1097
dgv_free, 1097
dgv_get, 1097
dgv_length, 1097
dgv_push, 1098
diag_copy, 1098
diag_count, 1098
diag_create, 1098
diag_error, 1099
diag_free, 1099
diag_get, 1099
diag_has_errors, 1100
diag_note, 1100
diag_warn, 1100
DiagnosticSeverity, 1096
DiagnosticVector, 1101
Error, 1096
Note, 1096
Warning, 1096
diag_copy
diag.l0, 1098
diag_count
diag.l0, 1098
diag_create
diag.l0, 1098
diag_error
diag.l0, 1099
diag_free
diag.l0, 1099
diag_from_node
l0_diagnostics, 29
diag_from_token
l0_diagnostics, 29
diag_get
diag.l0, 1099
diag_has_errors
diag.l0, 1100
diag_note
diag.l0, 1100
diag_print.l0
dp_format_header, 924
dp_print_collector, 924
dp_print_collector_with_sources, 924
dp_print_diag, 925
dp_print_header, 925
dp_print_snippet, 925
dp_render_snippet, 925
dp_severity_name, 926
dp_source_cached, 926
diag_warn
diag.l0, 1100
DiagCollector, 216
error_count, 216
items, 216
Diagnostic, 217
code, 217
column, 217
column_end, 217
filename, 217
line, 218
line_end, 218
message, 218
phase, 218
severity, 218
diagnostic
l0_lexer.Token, 460
DIAGNOSTIC_CODE_FAMILIES

l0_diagnostics, 30
diagnostics
l0_analysis.AnalysisResult, 77
l0_driver.L0Driver, 310
l0_expr_types.ExpressionTypeChecker, 263
l0_lexer.Lexer, 320
l0_lexer.Token, 461
l0_parser.Parser, 390
l0_symbols.ModuleEnv, 354
DiagnosticSeverity
diag.l0, 1096
DiagnosticVector
diag.l0, 1101
diags
AnalysisResult, 73
CliParseResult, 207
DriverState, 221
ImplParseResult, 287
LexerState, 322
ParseResult, 391
ParserState, 393
digit_to_char
string.l0, 596
div_ceil
integer.l0, 554
div_floor
integer.l0, 554
docs-mainpage-latex.md, 1122
DOT
l0_lexer.TokenKind, 467
DOUBLE_COLON
l0_lexer.TokenKind, 467
dp_format_header
diag_print.l0, 924
dp_print_collector
diag_print.l0, 924
dp_print_collector_with_sources
diag_print.l0, 924
dp_print_diag
diag_print.l0, 925
dp_print_header
diag_print.l0, 925
dp_print_snippet
diag_print.l0, 925
dp_render_snippet
diag_print.l0, 925
dp_severity_name
diag_print.l0, 926
dp_source_cached
diag_print.l0, 926
dr_copy_parse_diags
driver.l0, 928
dr_emit_error
driver.l0, 928
dr_load_module
driver.l0, 928
dr_mark_loaded
driver.l0, 929
dr_parse_source
driver.l0, 929
dr_sv_has
driver.l0, 929
dr_unit_free
driver.l0, 930
dr_units_free
driver.l0, 930
dr_visit_closure
driver.l0, 930
driver.l0
dr_copy_parse_diags, 928
dr_emit_error, 928
dr_load_module, 928
dr_mark_loaded, 929
dr_parse_source, 929
dr_sv_has, 929
dr_unit_free, 930
dr_units_free, 930
dr_visit_closure, 930
driver_analyze_entry, 930
driver_diag_count, 931
driver_diag_get, 931
driver_find_unit, 931
driver_has_errors, 932
driver_module_names_sorted, 932
driver_state_free, 932
driver_unit_count, 933
driver_unit_get, 933
driver_analyze_entry
driver.l0, 930
driver_diag_count
driver.l0, 931
driver_diag_get
driver.l0, 931
driver_find_unit
driver.l0, 931
driver_has_errors
driver.l0, 932
driver_module_names_sorted
driver.l0, 932
driver_state
AnalysisResult, 73
driver_state_free
driver.l0, 932
driver_unit_count
driver.l0, 933
driver_unit_get
driver.l0, 933
DriverState, 221
diags, 221
entry_module, 221
loaded, 221
source_names, 222
source_texts, 222
units, 222
DriverUnit, 222
module_name, 223

parse_result, 223
path, 223
DROP
l0_lexer.TokenKind, 467
drop_file
_rt_alloc_record_cold, 69
drop_line
_rt_alloc_record_cold, 69
Duration, 225
nsec, 225
sec, 225

ediv
integer.l0, 554
element_size
ArrayBase, 78
ELSE
l0_lexer.TokenKind, 467
else_arm
l0_ast.CaseStmt, 143
else_stmt
l0_ast.IfStmt, 285
emit
l0_c_emitter.CCodeBuilder, 148
emit_alloc_obj
l0_c_emitter.CEmitter, 163
emit_assignment
l0_c_emitter.CEmitter, 163
emit_binary_op
l0_c_emitter.CEmitter, 164
emit_block_end
l0_c_emitter.CEmitter, 164
emit_block_start
l0_c_emitter.CEmitter, 164
emit_bool_literal
l0_c_emitter.CEmitter, 164
emit_byte_literal
l0_c_emitter.CEmitter, 165
emit_case_label
l0_c_emitter.CEmitter, 165
emit_cast
l0_c_emitter.CEmitter, 165
emit_checked_int_add
l0_c_emitter.CEmitter, 166
emit_checked_int_div
l0_c_emitter.CEmitter, 166
emit_checked_int_mod
l0_c_emitter.CEmitter, 166
emit_checked_int_mul
l0_c_emitter.CEmitter, 166
emit_checked_int_sub
l0_c_emitter.CEmitter, 166
emit_checked_narrow_cast
l0_c_emitter.CEmitter, 167
emit_checked_pointer_assignment
l0_c_emitter.CEmitter, 167
emit_checked_ptr_access
l0_c_emitter.CEmitter, 167
emit_checked_ptr_access_for_base
l0_c_emitter.CEmitter, 167
emit_checked_ptr_index_access
l0_c_emitter.CEmitter, 167
emit_comment
l0_c_emitter.CEmitter, 168
emit_condition_binary_op
l0_c_emitter.CEmitter, 168
emit_condition_pointer_null_check
l0_c_emitter.CEmitter, 168
emit_condition_string_compare_call
l0_c_emitter.CEmitter, 168
emit_const_bool_literal
l0_c_emitter.CEmitter, 168
emit_const_string_literal
l0_c_emitter.CEmitter, 169
emit_default_label
l0_c_emitter.CEmitter, 169
emit_deref_lvalue
l0_c_emitter.CEmitter, 169
emit_discard_expr
l0_c_emitter.CEmitter, 169
emit_drop_begin_expr
l0_c_emitter.CEmitter, 170
emit_drop_finish_call
l0_c_emitter.CEmitter, 170
emit_else
l0_c_emitter.CEmitter, 170
emit_enum
l0_c_emitter.CEmitter, 170
emit_enum_cleanup
l0_c_emitter.CEmitter, 171
emit_enum_payload_field_access
l0_c_emitter.CEmitter, 171
emit_enum_tag
l0_c_emitter.CEmitter, 171
emit_enum_tag_access
l0_c_emitter.CEmitter, 171
emit_enum_variant_init
l0_c_emitter.CEmitter, 172
emit_exit_switch
l0_c_emitter.CEmitter, 172
emit_expr_stmt
l0_c_emitter.CEmitter, 172
emit_field_access
l0_c_emitter.CEmitter, 172
emit_field_lvalue
l0_c_emitter.CEmitter, 173
emit_for_loop_end
l0_c_emitter.CEmitter, 173
emit_for_loop_start
l0_c_emitter.CEmitter, 173
emit_forward_decls
l0_c_emitter.CEmitter, 173
emit_function_call
l0_c_emitter.CEmitter, 173
emit_function_declaration
l0_c_emitter.CEmitter, 174
emit_function_definition_footer

l0_c_emitter.CEmitter, 174
emit_function_definition_header
l0_c_emitter.CEmitter, 174
emit_goto
l0_c_emitter.CEmitter, 174
emit_header
l0_c_emitter.CEmitter, 175
emit_if_header
l0_c_emitter.CEmitter, 175
emit_index_lvalue
l0_c_emitter.CEmitter, 175
emit_int_literal
l0_c_emitter.CEmitter, 175
emit_label
l0_c_emitter.CEmitter, 176
emit_let_decl
l0_c_emitter.CEmitter, 176
emit_let_declaration
l0_c_emitter.CEmitter, 176
emit_line_directive
l0_c_emitter.CEmitter, 177
emit_line_directives
CodegenOptions, 208
l0_context.CompilationContext, 210
emit_main_wrapper
l0_c_emitter.CEmitter, 177
emit_match_scrutinee_decl
l0_c_emitter.CEmitter, 177
emit_match_switch_start
l0_c_emitter.CEmitter, 178
emit_module_comment
l0_c_emitter.CEmitter, 178
emit_module_separator
l0_c_emitter.CEmitter, 178
emit_negated_condition
l0_c_emitter.CEmitter, 178
emit_none_value_for_nullable
l0_c_emitter.CEmitter, 178
emit_null_assignment
l0_c_emitter.CEmitter, 179
emit_null_check_eq
l0_c_emitter.CEmitter, 179
emit_null_check_ne
l0_c_emitter.CEmitter, 179
emit_null_literal
l0_c_emitter.CEmitter, 179
emit_optional_has_value
l0_c_emitter.CEmitter, 180
emit_optional_value
l0_c_emitter.CEmitter, 180
emit_optional_wrapper_for_defined_type
l0_c_emitter.CEmitter, 180
emit_optional_wrappers
l0_c_emitter.CEmitter, 180
emit_ord
l0_c_emitter.CEmitter, 181
emit_paren_expr
l0_c_emitter.CEmitter, 181
emit_pattern_binding_init
l0_c_emitter.CEmitter, 181
emit_pointer_assignment
l0_c_emitter.CEmitter, 182
emit_pointer_null_check
l0_c_emitter.CEmitter, 182
emit_pointer_type
l0_c_emitter.CEmitter, 182
emit_raw
l0_c_emitter.CCodeBuilder, 148
emit_return_stmt
l0_c_emitter.CEmitter, 183
emit_section_comment
l0_c_emitter.CEmitter, 183
emit_sizeof_type
l0_c_emitter.CEmitter, 183
emit_some_value_for_nullable
l0_c_emitter.CEmitter, 184
emit_string_compare_call
l0_c_emitter.CEmitter, 184
emit_string_concat_call
l0_c_emitter.CEmitter, 184
emit_string_equals_call
l0_c_emitter.CEmitter, 184
emit_string_literal
l0_c_emitter.CEmitter, 184
emit_string_release
l0_c_emitter.CEmitter, 185
emit_string_retain
l0_c_emitter.CEmitter, 185
emit_struct
l0_c_emitter.CEmitter, 185
emit_struct_cleanup
l0_c_emitter.CEmitter, 187
emit_struct_constructor
l0_c_emitter.CEmitter, 187
emit_struct_constructor_for_type
l0_c_emitter.CEmitter, 187
emit_struct_init
l0_c_emitter.CEmitter, 187
emit_struct_init_from_fields
l0_c_emitter.CEmitter, 188
emit_struct_static_initializer
l0_c_emitter.CEmitter, 188
emit_struct_static_initializer_for_type
l0_c_emitter.CEmitter, 188
emit_switch_end
l0_c_emitter.CEmitter, 188
emit_switch_start
l0_c_emitter.CEmitter, 189
emit_temp_decl
l0_c_emitter.CEmitter, 189
emit_top_level_let_cleanup
l0_c_emitter.CEmitter, 189
emit_try_check_niche
l0_c_emitter.CEmitter, 189
emit_try_check_value
l0_c_emitter.CEmitter, 190

emit_try_extract_value
l0_c_emitter.CEmitter, 190
emit_type
l0_c_emitter.CEmitter, 190
emit_unary_op
l0_c_emitter.CEmitter, 191
emit_unreachable_comment
l0_c_emitter.CEmitter, 191
emit_unreachable_marker
l0_c_emitter.CEmitter, 191
emit_unwrap_opt
l0_c_emitter.CEmitter, 191
emit_unwrap_ptr
l0_c_emitter.CEmitter, 192
emit_value_cleanup
l0_c_emitter.CEmitter, 192
emit_var_ref
l0_c_emitter.CEmitter, 192
emit_variant_constructor
l0_c_emitter.CEmitter, 192
emit_variant_constructor_for_type
l0_c_emitter.CEmitter, 193
emit_variant_static_initializer
l0_c_emitter.CEmitter, 193
emit_variant_static_initializer_for_type
l0_c_emitter.CEmitter, 193
emit_while_header
l0_c_emitter.CEmitter, 193
emit_widen_int
l0_c_emitter.CEmitter, 194
emit_zero_init
l0_c_emitter.CEmitter, 194
emitted_lex_errors
ParserState, 393
emitter
Backend, 82
l0_backend.Backend, 124
emod
integer.l0, 555
encode_c_string_bytes
l0_string_escape, 40
end_column
l0_ast.Span, 427
l0_diagnostics.Diagnostic, 219
Span, 428
end_line
l0_ast.Span, 427
l0_diagnostics.Diagnostic, 220
Span, 428
ends_with_s
string.l0, 597
entries
LinearMapBase, 324
entry_module
BuildPreparedInput, 131
DriverState, 221
l0_compilation.CompilationUnit, 212
PreparedInput, 399
entry_name
l0_compilation.CompilationUnit, 212
entry_target
CliOptions, 204
ENUM
l0_lexer.TokenKind, 468
l0_symbols.SymbolKind, 457
enum_decl_free
ast.l0, 780
enum_info_free
types.l0, 1089
enum_infos
l0_analysis.AnalysisResult, 77
l0_expr_types.ExpressionTypeChecker, 263
SignatureTables, 421
enum_type
EnumInfo, 229
l0_signatures.EnumInfo, 230
ENUM_VARIANT
l0_symbols.SymbolKind, 457
enum_variant_free
ast.l0, 780
enum_variant_vec_free
ast.l0, 780
EnumDecl, 225
name, 226
span, 226
variants, 226
EnumInfo, 228
enum_type, 229
filename, 229
key, 229
module_name, 229
name, 229
span, 229
variants, 229
EnumVariant, 232
fields, 233
name, 233
EnumVariantInfo, 234
field_types, 235
name, 235
env_get
system.l0, 604
envs
LocalsResult, 331
NameResolution, 358
EOF
l0_lexer.TokenKind, 468
eof_aborted
l0_parser.Parser, 390
ParserState, 393
EQ
l0_lexer.TokenKind, 468
eq_s
string.l0, 597
EQEQ
l0_lexer.TokenKind, 468

err_print_b
io.l0, 562
err_print_bb
io.l0, 562
err_print_bi
io.l0, 562
err_print_bs
io.l0, 562
err_print_i
io.l0, 562
err_print_ib
io.l0, 563
err_print_ii
io.l0, 563
err_print_is
io.l0, 563
err_print_s
io.l0, 563
err_print_sb
io.l0, 564
err_print_si
io.l0, 564
err_print_ss
io.l0, 564
err_printl
io.l0, 564
err_printl_b
io.l0, 565
err_printl_i
io.l0, 565
err_printl_s
io.l0, 565
errno
system.l0, 604
ERROR
l0_context.LogLevel, 336
Error
diag.l0, 1096
error
l0_resolve.SymbolResolution, 459
l0_resolve.TypeResolution, 488
error_count
DiagCollector, 216
escape_char_value
lexer.l0, 975
escape_str
EscapedChar, 236
EscapedChar, 236
char_value, 236
escape_str, 236
etc_alive_declare
expr_types.l0, 939
etc_alive_lookup
expr_types.l0, 939
etc_alive_revive_assignment_target
expr_types.l0, 939
etc_alive_scope_clone
expr_types.l0, 939
etc_alive_scope_current
expr_types.l0, 940
etc_alive_scope_pop
expr_types.l0, 940
etc_alive_scope_push
expr_types.l0, 940
etc_alive_scope_stack_clone
expr_types.l0, 940
etc_alive_scope_stack_equal
expr_types.l0, 941
etc_alive_scope_stack_free
expr_types.l0, 941
etc_alive_scope_stack_meet
expr_types.l0, 941
etc_alive_scope_stack_meet_many
expr_types.l0, 941
etc_alive_scope_stack_restore
expr_types.l0, 941
etc_alive_set
expr_types.l0, 942
etc_alive_state_vec_clone
expr_types.l0, 942
etc_alive_state_vec_free
expr_types.l0, 942
etc_alive_state_vec_push_clone
expr_types.l0, 942
etc_bind_pattern_types
expr_types.l0, 942
etc_can_assign
expr_types.l0, 943
etc_can_cast
expr_types.l0, 943
etc_capture_break_state
expr_types.l0, 943
etc_capture_continue_state
expr_types.l0, 943
etc_case_literal_key
expr_types.l0, 943
etc_cast_const_int
expr_types.l0, 944
etc_cast_is_const_null
expr_types.l0, 944
etc_check_assign_expr
expr_types.l0, 944
etc_check_dead_liveness_stmt
expr_types.l0, 945
etc_check_expr_liveness
expr_types.l0, 945
etc_check_function
expr_types.l0, 945
etc_check_initializer_expr
expr_types.l0, 945
etc_check_local_declaration
expr_types.l0, 946
etc_check_loop_iteration
expr_types.l0, 946
etc_check_match_exhaustiveness
expr_types.l0, 946

etc_check_match_pattern
expr_types.l0, 947
etc_check_stmt
expr_types.l0, 947
etc_check_top_level_initializers
expr_types.l0, 947
etc_check_update_from_backedge_states
expr_types.l0, 948
etc_cleanup_header_guard_stack_free
expr_types.l0, 948
etc_decl_outer_has
expr_types.l0, 948
etc_decl_scope_current
expr_types.l0, 949
etc_decl_scope_pop
expr_types.l0, 949
etc_decl_scope_push
expr_types.l0, 949
etc_diag_error
expr_types.l0, 949
etc_diag_type_ref_failure
expr_types.l0, 950
etc_diag_warn
expr_types.l0, 950
etc_expr_contains_try
expr_types.l0, 950
etc_find_enum_variant_info
expr_types.l0, 952
etc_format_missing_match_variants
expr_types.l0, 952
etc_get_pattern_var_type
expr_types.l0, 952
etc_infer_binary
expr_types.l0, 953
etc_infer_call
expr_types.l0, 953
etc_infer_expr
expr_types.l0, 953
etc_infer_field
expr_types.l0, 954
etc_infer_index
expr_types.l0, 954
etc_infer_ord_intrinsic
expr_types.l0, 954
etc_infer_sizeof_intrinsic
expr_types.l0, 955
etc_infer_unary
expr_types.l0, 955
etc_is_bool
expr_types.l0, 956
etc_is_case_type
expr_types.l0, 956
etc_is_guarded_cleanup_header_local
expr_types.l0, 956
etc_is_int
expr_types.l0, 956
etc_is_string
expr_types.l0, 957
etc_is_void
expr_types.l0, 957
etc_key
expr_types.l0, 957
etc_local_symbol_type
expr_types.l0, 958
etc_lookup_local
expr_types.l0, 958
etc_lookup_symbol
expr_types.l0, 958
etc_lookup_visible_local_symbol
expr_types.l0, 959
etc_loop_flow_capture_free
expr_types.l0, 959
etc_loop_liveness_fixed_point
expr_types.l0, 959
etc_match_variants_fully_covered
expr_types.l0, 959
etc_match_wildcard_is_unreachable
expr_types.l0, 959
etc_nested_pattern_path_message
expr_types.l0, 959
etc_nested_symbol_path_full
expr_types.l0, 960
etc_nested_symbol_path_message
expr_types.l0, 960
etc_nested_symbol_path_simple
expr_types.l0, 960
etc_pattern_symbol_name
expr_types.l0, 961
etc_pattern_var_key
expr_types.l0, 961
etc_scope_chain_push
expr_types.l0, 961
etc_set_expr_type
expr_types.l0, 961
etc_set_intrinsic_target
expr_types.l0, 962
etc_set_pattern_var_type
expr_types.l0, 962
etc_set_var_res
expr_types.l0, 962
etc_stmt_contains_try
expr_types.l0, 963
etc_symbol_as_type
expr_types.l0, 963
etc_try_resolve_type_name
expr_types.l0, 963
etc_type_free_opt
expr_types.l0, 964
etc_types_equal
expr_types.l0, 964
EX_BINARY
ast.l0, 778
EX_BOOL
ast.l0, 778
EX_BYTE
ast.l0, 778

EX_CALL
ast.l0, 778
EX_CAST
ast.l0, 778
EX_FIELD
ast.l0, 778
EX_INDEX
ast.l0, 778
EX_INT
ast.l0, 778
EX_NEW
ast.l0, 778
EX_NULL
ast.l0, 778
EX_PAREN
ast.l0, 778
EX_STRING
ast.l0, 778
EX_TRY
ast.l0, 778
EX_TYPE
ast.l0, 778
EX_UNARY
ast.l0, 778
EX_VAR_REF
ast.l0, 778
exists
FileInfo, 272
fs.l0, 516
l0_sys_rt_RtFileInfo, 305
RtFileInfo, 404
exit
system.l0, 604
expect_b
optional.l0, 589
expect_i
optional.l0, 589
expect_s
optional.l0, 590
expr
l0_ast.CaseStmt, 144
l0_ast.CastExpr, 145
l0_ast.ExprStmt, 267
l0_ast.MatchStmt, 349
l0_ast.TryExpr, 477
expr.l0
ps_is_builtin_type_name, 1012
ps_is_unambiguous_type_start, 1013
ps_parse_add_expr, 1013
ps_parse_and_expr, 1013
ps_parse_call_argument, 1014
ps_parse_cast_expr, 1014
ps_parse_equality_expr, 1014
ps_parse_expr, 1014
ps_parse_mul_expr, 1015
ps_parse_or_expr, 1015
ps_parse_postfix_expr, 1015
ps_parse_primary_expr, 1016
ps_parse_rel_expr, 1016
ps_parse_unary_expr, 1016
expr_a
ExprNode, 265
StmtNode, 434
expr_arena
ImplParseResult, 287
ParseResult, 391
ParserState, 393
expr_arena_add
ast.l0, 780
expr_arena_create
ast.l0, 781
expr_arena_free
ast.l0, 781
expr_arena_get
ast.l0, 781
expr_b
ExprNode, 265
StmtNode, 434
expr_c
ExprNode, 265
expr_node_create
ast.l0, 782
expr_node_free
ast.l0, 782
expr_types
AnalysisResult, 73
l0_analysis.AnalysisResult, 77
l0_expr_types.ExpressionTypeChecker, 263
expr_types.l0
ASSIGN_ACCEPT, 938
ASSIGN_REJECT, 938
ASSIGN_REPORTED, 938
AssignCheckResult, 938
etc_alive_declare, 939
etc_alive_lookup, 939
etc_alive_revive_assignment_target, 939
etc_alive_scope_clone, 939
etc_alive_scope_current, 940
etc_alive_scope_pop, 940
etc_alive_scope_push, 940
etc_alive_scope_stack_clone, 940
etc_alive_scope_stack_equal, 941
etc_alive_scope_stack_free, 941
etc_alive_scope_stack_meet, 941
etc_alive_scope_stack_meet_many, 941
etc_alive_scope_stack_restore, 941
etc_alive_set, 942
etc_alive_state_vec_clone, 942
etc_alive_state_vec_free, 942
etc_alive_state_vec_push_clone, 942
etc_bind_pattern_types, 942
etc_can_assign, 943
etc_can_cast, 943
etc_capture_break_state, 943
etc_capture_continue_state, 943
etc_case_literal_key, 943

etc_cast_const_int, 944
etc_cast_is_const_null, 944
etc_check_assign_expr, 944
etc_check_dead_liveness_stmt, 945
etc_check_expr_liveness, 945
etc_check_function, 945
etc_check_initializer_expr, 945
etc_check_local_declaration, 946
etc_check_loop_iteration, 946
etc_check_match_exhaustiveness, 946
etc_check_match_pattern, 947
etc_check_stmt, 947
etc_check_top_level_initializers, 947
etc_check_update_from_backedge_states, 948
etc_cleanup_header_guard_stack_free, 948
etc_decl_outer_has, 948
etc_decl_scope_current, 949
etc_decl_scope_pop, 949
etc_decl_scope_push, 949
etc_diag_error, 949
etc_diag_type_ref_failure, 950
etc_diag_warn, 950
etc_expr_contains_try, 950
etc_find_enum_variant_info, 952
etc_format_missing_match_variants, 952
etc_get_pattern_var_type, 952
etc_infer_binary, 953
etc_infer_call, 953
etc_infer_expr, 953
etc_infer_field, 954
etc_infer_index, 954
etc_infer_ord_intrinsic, 954
etc_infer_sizeof_intrinsic, 955
etc_infer_unary, 955
etc_is_bool, 956
etc_is_case_type, 956
etc_is_guarded_cleanup_header_local, 956
etc_is_int, 956
etc_is_string, 957
etc_is_void, 957
etc_key, 957
etc_local_symbol_type, 958
etc_lookup_local, 958
etc_lookup_symbol, 958
etc_lookup_visible_local_symbol, 959
etc_loop_flow_capture_free, 959
etc_loop_liveness_fixed_point, 959
etc_match_variants_fully_covered, 959
etc_match_wildcard_is_unreachable, 959
etc_nested_pattern_path_message, 959
etc_nested_symbol_path_full, 960
etc_nested_symbol_path_message, 960
etc_nested_symbol_path_simple, 960
etc_pattern_symbol_name, 961
etc_pattern_var_key, 961
etc_scope_chain_push, 961
etc_set_expr_type, 961
etc_set_intrinsic_target, 962
etc_set_pattern_var_type, 962
etc_set_var_res, 962
etc_stmt_contains_try, 963
etc_symbol_as_type, 963
etc_try_resolve_type_name, 963
etc_type_free_opt, 964
etc_types_equal, 964
expr_types_check, 964
FLOW_FALLTHROUGH, 938
FLOW_RETURNS, 938
FLOW_STOPS, 938
StmtFlowKind, 938
expr_types_check
expr_types.l0, 964
ExprArena, 240
nodes, 241
ExpressionTypeChecker, 242
alive_scopes, 243
cleanup_header_guard_stack, 243
current_env, 243
current_func, 243
current_module, 243
current_ret_type, 243
current_scope, 243
current_unit, 243
decl_scope_defs, 243
decl_scopes, 243
liveness_diagnostics_only, 244
loop_depth, 244
loop_flow_capture_stack, 244
next_stmt_unreachable, 244
result, 244
suppress_diagnostics, 244
suppress_liveness_diagnostics, 244
ExprId
ast.l0, 792
ExprKind
ast.l0, 778
ExprNode, 264
bool_value, 265
expr_a, 265
expr_b, 265
expr_c, 265
ids, 265
int_value, 265
kind, 265
module_path, 266
name_qualifier, 266
span, 266
text_value, 266
type_ref, 266
EXTERN
l0_lexer.TokenKind, 468
extra_target_count
CliOptions, 204

FALLTHROUGH
l0_expr_types.StmtFlow, 432
FALSE

l0_lexer.TokenKind, 468
field
l0_ast.FieldAccessExpr, 269
field_decl_free
ast.l0, 782
field_decl_vec_free
ast.l0, 782
field_type
StructFieldInfo, 448
field_types
EnumVariantInfo, 235
l0_signatures.EnumVariantInfo, 236
FieldDecl, 269
name, 270
type_ref, 270
fields
EnumVariant, 233
l0_ast.EnumVariant, 234
l0_ast.StructDecl, 445
l0_signatures.StructInfo, 449
StructDecl, 446
StructInfo, 450
file_size
fs.l0, 516
FileInfo, 272
exists, 272
is_dir, 272
is_file, 272
mtime_nsec, 272
mtime_sec, 272
size, 273
filename
Diagnostic, 217
EnumInfo, 229
l0_ast.Module, 351
l0_diagnostics.Diagnostic, 220
l0_internal_error.ICELocation, 284
l0_lexer.Lexer, 320
l0_parser.Parser, 390
LexerState, 322
Module, 352
ParserState, 393
StructInfo, 450
find_from_s
string.l0, 597
find_last_s
string.l0, 598
find_s
string.l0, 598
find_variant_decl
l0_backend.Backend, 121
l0_c_emitter.CEmitter, 195
FLOW_FALLTHROUGH
expr_types.l0, 938
FLOW_RETURNS
expr_types.l0, 938
FLOW_STOPS
expr_types.l0, 938
flush_stderr
io.l0, 565
flush_stdout
io.l0, 565
FOR
l0_lexer.TokenKind, 468
format
l0_diagnostics.Diagnostic, 219
l0_internal_error.InternalCompilerError, 294
format_module
l0_ast_printer, 27
format_node
l0_ast_printer, 27
format_type
l0_types, 42
fresh_tmp
l0_c_emitter.CEmitter, 195
from_source
l0_lexer.Lexer, 319
l0_parser.Parser, 389
fs.l0
_fs_from_rt, 515
delete_file, 516
exists, 516
file_size, 516
is_dir, 516
is_file, 517
mtime_sec, 517
read_file, 517
stat, 518
write_file, 518
FUNC
l0_lexer.TokenKind, 468
l0_symbols.SymbolKind, 457
func
l0_locals.FunctionEnv, 281
func_decl
FunctionEnv, 280
func_decl_free
ast.l0, 783
func_envs
l0_analysis.AnalysisResult, 77
l0_expr_types.ExpressionTypeChecker, 263
func_name
FunctionEnv, 280
func_types
l0_analysis.AnalysisResult, 77
l0_expr_types.ExpressionTypeChecker, 263
SignatureTables, 421
FuncDecl, 275
body, 276
is_extern, 276
name, 276
params, 276
return_type, 276
span, 276
FunctionEnv, 279
arm_scopes, 279

block_scopes, 279
func_decl, 280
func_name, 280
key, 280
module_name, 280
pattern_arena, 280
root_scope, 280
scopes, 280
stmt_arena, 280
FUTURE_EXTENSION
l0_lexer.TokenKind, 468

gcd
integer.l0, 555
GE
l0_lexer.TokenKind, 468
generate
l0_backend.Backend, 122
generation
_rt_alloc_record, 67
_rt_ptr_site, 71
get_block_scope
l0_locals.LocalScopeResolver, 329
get_builtin_type
l0_types, 42
get_match_arm_scope
l0_locals.LocalScopeResolver, 330
get_null_type
l0_types, 42
get_output
l0_c_emitter.CEmitter, 195
get_pid
system.l0, 605
GT
l0_lexer.TokenKind, 469
guarded_decl_ids
CleanupHeaderGuard, 202

h_str
l0_string, 303
has_arc_data
l0_analysis.AnalysisResult, 75
has_cleanup
WithItem, 505
has_errors
l0_analysis.AnalysisResult, 76
has_extension
path.l0, 1114
has_parent
path.l0, 1114
has_value
_l0_base_opt, 63
l0_opt_bool, 300
l0_opt_byte, 300
l0_opt_int, 301
l0_opt_string, 302
has_warnings
l0_analysis.AnalysisResult, 76
hash.l0
rt_hash_bool, 634
rt_hash_byte, 634
rt_hash_data, 635
rt_hash_int, 635
rt_hash_opt_bool, 635
rt_hash_opt_byte, 636
rt_hash_opt_int, 636
rt_hash_opt_ptr, 636
rt_hash_opt_string, 637
rt_hash_ptr, 637
rt_hash_string, 637
hash_key
LinearMapEntryBase, 324
hash_value
LinearMapEntryBase, 324
hashes
StringIntMap, 437
StringPtrMap, 440
StringSet, 442
hashmap.l0
_hm_slot, 521
_sim_alloc, 521
_sim_find, 522
_sim_find_insert, 522
_sim_get_hash, 522
_sim_get_key, 523
_sim_get_state, 523
_sim_get_value, 523
_sim_needs_grow, 524
_sim_rehash, 524
_sim_release_key, 524
_sim_set_hash, 524
_sim_set_key_retain, 525
_sim_set_state, 525
_sim_set_value, 525
_spm_alloc, 526
_spm_find, 526
_spm_find_insert, 526
_spm_get_hash, 526
_spm_get_key, 527
_spm_get_state, 527
_spm_get_value, 527
_spm_needs_grow, 528
_spm_rehash, 528
_spm_release_key, 528
_spm_set_hash, 528
_spm_set_key_retain, 529
_spm_set_state, 529
_spm_set_value, 529
HM_DEFAULT_CAP, 538
HM_EMPTY, 538
HM_OCCUPIED, 538
HM_TOMBSTONE, 538
sim_capacity, 530
sim_clear, 530
sim_create, 530
sim_create_with_capacity, 530
sim_free, 531

sim_get, 531
sim_has, 531
sim_keys, 531
sim_put, 532
sim_remove, 532
sim_size, 532
sim_slot_key, 533
sim_slot_occupied, 533
sim_slot_value, 533
spm_capacity, 534
spm_clear, 534
spm_create, 534
spm_create_with_capacity, 534
spm_free, 535
spm_get, 535
spm_has, 535
spm_keys, 536
spm_put, 536
spm_remove, 536
spm_size, 537
spm_slot_key, 537
spm_slot_occupied, 537
spm_slot_value, 538
hashset.l0
_hs_slot, 540
_ss_alloc, 540
_ss_find, 542
_ss_find_insert, 542
_ss_get_hash, 542
_ss_get_key, 543
_ss_get_state, 543
_ss_needs_grow, 543
_ss_rehash, 544
_ss_release_key, 544
_ss_set_hash, 544
_ss_set_key_retain, 544
_ss_set_state, 546
HS_DEFAULT_CAP, 550
HS_EMPTY, 550
HS_OCCUPIED, 550
HS_TOMBSTONE, 550
ss_add, 546
ss_capacity, 546
ss_clear, 547
ss_create, 547
ss_create_with_capacity, 547
ss_free, 547
ss_has, 548
ss_remove, 548
ss_size, 548
ss_slot_key, 549
ss_slot_occupied, 549
ss_to_vector, 549
head
LoopFixedPointFlow, 339
help_requested
CliParseResult, 207
hex_char_to_int
lexer.l0, 975
HEX_CHARS
l0_lexer, 33
HM_DEFAULT_CAP
hashmap.l0, 538
HM_EMPTY
hashmap.l0, 538
HM_OCCUPIED
hashmap.l0, 538
HM_TOMBSTONE
hashmap.l0, 538
hot_pad
_rt_alloc_record, 68
hour
DateTime, 215
HS_DEFAULT_CAP
hashset.l0, 550
HS_EMPTY
hashset.l0, 550
HS_OCCUPIED
hashset.l0, 550
HS_TOMBSTONE
hashset.l0, 550

ice
l0_backend.Backend, 122
l0_c_emitter.CEmitter, 195
id_vec_create
ast.l0, 783
id_vec_free
ast.l0, 783
id_vec_get
ast.l0, 783
id_vec_push
ast.l0, 784
IDENT
l0_lexer.TokenKind, 469
ident_to_token
tokens.l0, 1075
ids
ExprNode, 265
StmtNode, 434
IF
l0_lexer.TokenKind, 469
impl_parse_module_source
decl.l0, 1007
impl_parse_module_tokens
decl.l0, 1007
impl_result_to_public
parser.l0, 1003
ImplParseResult, 286
diags, 287
expr_arena, 287
pattern_arena, 287
root_module, 287
stmt_arena, 287
IMPORT
l0_lexer.TokenKind, 469
Import, 287

name, 288
span, 288
import_vec_free
ast.l0, 784
imported
l0_symbols.ModuleEnv, 354
ModuleEnv, 356
imported_modules
ModuleEnv, 356
imports
l0_ast.Module, 351
Module, 352
include_eof
CliOptions, 204
indent
l0_c_emitter.CCodeBuilder, 148
indent_level
CCodeBuilder, 146
l0_c_emitter.CCodeBuilder, 149
indent_str
CCodeBuilder, 146
l0_c_emitter.CCodeBuilder, 149
index
l0_ast.IndexExpr, 292
l0_lexer.Lexer, 320
l0_parser.Parser, 390
LexerState, 322
ParserState, 393
Token, 462
INFO
l0_context.LogLevel, 336
init
l0_ast.ForStmt, 275
l0_ast.WithItem, 504
WithItem, 505
inner
l0_ast.ParenExpr, 373
l0_types.NullableType, 367
l0_types.PointerType, 398
Type, 479
INT
l0_lexer.TokenKind, 469
int
l0_string, 303
int_max
integer.l0, 555
int_min
integer.l0, 555
int_to_bin_string
text.l0, 611
int_to_hex_string
text.l0, 612
int_to_string
text.l0, 612
int_to_string_base
text.l0, 612
int_type
l0_expr_types.ExpressionTypeChecker, 263
int_value
ExprNode, 265
integer.l0
abs, 551
align_down, 552
align_up, 552
checked_add_int, 552
checked_mul_int, 553
checked_sub_int, 553
clamp, 553
div_ceil, 554
div_floor, 554
ediv, 554
emod, 555
gcd, 555
int_max, 555
int_min, 555
is_aligned, 556
is_even, 556
is_multiple, 556
is_odd, 557
isqrt, 557
lcm, 557
max, 558
min, 558
pow, 558
sign, 559
interface_paths
CliOptions, 204
intrinsic_targets
AnalysisResult, 73
l0_analysis.AnalysisResult, 77
IntSet, 296
capacity, 297
count, 297
keys, 297
states, 297
tomb_count, 297
intset.l0
_iset_alloc, 1102
_iset_find, 1102
_iset_find_insert, 1103
_iset_get_key, 1103
_iset_get_state, 1103
_iset_needs_grow, 1104
_iset_rehash, 1104
_iset_set_key, 1104
_iset_set_state, 1105
_iset_slot, 1105
iset_add, 1105
iset_capacity, 1106
iset_clear, 1106
iset_create, 1106
iset_create_with_capacity, 1106
ISET_DEFAULT_CAP, 1108
ISET_EMPTY, 1108
iset_free, 1107
iset_has, 1107

ISET_OCCUPIED, 1108
iset_remove, 1107
iset_size, 1108
ISET_TOMBSTONE, 1108
IntStringLinearMap, 298
base, 298
IntStringLinearMapEntry, 299
key, 299
value, 299
invalid_expr_id
ast.l0, 784
INVALID_NULLABLE_VOID
l0_resolve.TypeResolveErrorKind, 490
invalid_pattern_id
ast.l0, 784
invalid_stmt_id
ast.l0, 784
io.l0
_io_check_byte_range, 561
err_print_b, 562
err_print_bb, 562
err_print_bi, 562
err_print_bs, 562
err_print_i, 562
err_print_ib, 563
err_print_ii, 563
err_print_is, 563
err_print_s, 563
err_print_sb, 564
err_print_si, 564
err_print_ss, 564
err_printl, 564
err_printl_b, 565
err_printl_i, 565
err_printl_s, 565
flush_stderr, 565
flush_stdout, 565
print_b, 565
print_bb, 566
print_bi, 566
print_bs, 566
print_i, 566
print_ib, 567
print_ii, 567
print_is, 567
print_s, 567
print_sb, 568
print_si, 568
print_ss, 568
printl, 568
printl_b, 569
printl_bb, 569
printl_bi, 569
printl_bs, 569
printl_i, 570
printl_ib, 570
printl_ii, 570
printl_is, 570
printl_s, 571
printl_sb, 571
printl_si, 571
printl_ss, 571
read_char, 572
read_char_or_eof, 572
read_line, 572
read_stdin_some, 572
write_stderr_all, 573
write_stderr_some, 573
write_stdout_all, 573
write_stdout_some, 574
is_absolute
path.l0, 1114
is_aligned
integer.l0, 556
is_alnum
string.l0, 598
is_alpha
string.l0, 599
is_arc_type
l0_analysis.AnalysisResult, 76
is_digit
string.l0, 599
is_digit_base
string.l0, 599
is_dir
FileInfo, 272
fs.l0, 516
l0_sys_rt_RtFileInfo, 305
RtFileInfo, 404
is_dst
DateTime, 215
is_empty_s
string.l0, 599
is_escape_char
lexer.l0, 976
is_even
integer.l0, 556
is_expr_ending_token_type
tokens.l0, 1077
is_extern
FuncDecl, 276
l0_ast.FuncDecl, 278
is_file
FileInfo, 272
fs.l0, 517
l0_sys_rt_RtFileInfo, 305
RtFileInfo, 404
is_hex_digit
lexer.l0, 976
is_ident_part
lexer.l0, 976
is_ident_start
lexer.l0, 976
is_leap_year
time.l0, 621
is_multiple

integer.l0, 556
is_niche_nullable
l0_c_emitter.CEmitter, 196
is_nullable
l0_ast.TypeRef, 485
TypeRef, 486
is_octal_digit
lexer.l0, 977
is_odd
integer.l0, 557
is_printable_ascii
lexer.l0, 977
is_reserved_ident_word
tokens.l0, 1078
is_reserved_keyword
l0_lexer, 32
tokens.l0, 1078
is_sep
path.l0, 1115
is_space
string.l0, 600
is_valid_nsec
time.l0, 622
iset_add
intset.l0, 1105
iset_capacity
intset.l0, 1106
iset_clear
intset.l0, 1106
iset_create
intset.l0, 1106
iset_create_with_capacity
intset.l0, 1106
ISET_DEFAULT_CAP
intset.l0, 1108
ISET_EMPTY
intset.l0, 1108
iset_free
intset.l0, 1107
iset_has
intset.l0, 1107
ISET_OCCUPIED
intset.l0, 1108
iset_remove
intset.l0, 1107
iset_size
intset.l0, 1108
ISET_TOMBSTONE
intset.l0, 1108
islm_contains
linear_map.l0, 580
islm_create
linear_map.l0, 580
islm_free
linear_map.l0, 581
islm_get
linear_map.l0, 581
islm_key_at
linear_map.l0, 581
islm_len
linear_map.l0, 581
islm_remove
linear_map.l0, 582
islm_set
linear_map.l0, 582
islm_value_at
linear_map.l0, 582
isqrt
integer.l0, 557
items
DiagCollector, 216
l0_ast.WithStmt, 507

join
path.l0, 1115
join_s
text.l0, 613

keep_c
CliOptions, 204
key
EnumInfo, 229
FunctionEnv, 280
IntStringLinearMapEntry, 299
StringStringLinearMapEntry, 443
StructInfo, 451
key_size
LinearMapBase, 324
keys
IntSet, 297
StringIntMap, 437
StringPtrMap, 440
StringSet, 442
KEYWORDS
l0_lexer, 33
kind
ExprNode, 265
l0_diagnostics.Diagnostic, 220
l0_lexer.Token, 461
l0_locals.LocalSymbol, 332
l0_string, 303
l0_symbols.Symbol, 453
LocalSymbol, 334
PatternNode, 396
StmtNode, 434
Symbol, 455
TopLevelDecl, 475
Type, 479
KV_RE
check_trace_log, 24

l0_analysis, 24
l0_analysis.AnalysisResult, 74
context, 77
cu, 77
diagnostics, 77
enum_infos, 77

expr_types, 77
func_envs, 77
func_types, 77
has_arc_data, 75
has_errors, 76
has_warnings, 76
intrinsic_targets, 77
is_arc_type, 76
let_types, 77
module_envs, 77
struct_infos, 77
var_ref_resolution, 78
l0_analysis.VarRefResolution, 497
LOCAL, 497
MODULE, 497
l0_ast, 24
l0_ast.AssignStmt, 79
target, 80
value, 80
l0_ast.BinaryOp, 125
left, 126
op, 126
right, 126
l0_ast.Block, 127
stmts, 128
l0_ast.BoolLiteral, 128
value, 129
l0_ast.BreakStmt, 130
l0_ast.ByteLiteral, 134
value, 135
l0_ast.CallExpr, 136
args, 137
callee, 137
l0_ast.CaseArm, 138
body, 139
literal, 139
l0_ast.CaseElse, 141
body, 142
l0_ast.CaseStmt, 142
arms, 143
else_arm, 143
expr, 144
l0_ast.CastExpr, 144
expr, 145
target_type, 145
l0_ast.ContinueStmt, 213
l0_ast.DropStmt, 223
name, 224
l0_ast.EnumDecl, 226
name, 227
variants, 227
l0_ast.EnumVariant, 233
fields, 234
name, 234
l0_ast.Expr, 238
l0_ast.ExprStmt, 266
expr, 267
l0_ast.FieldAccessExpr, 268
field, 269
obj, 269
l0_ast.FieldDecl, 270
name, 271
type, 271
l0_ast.ForStmt, 273
body, 275
cond, 275
init, 275
update, 275
l0_ast.FuncDecl, 277
body, 278
is_extern, 278
name, 278
params, 278
return_type, 278
l0_ast.IfStmt, 284
cond, 285
else_stmt, 285
then_stmt, 286
l0_ast.Import, 288
name, 289
l0_ast.IndexExpr, 291
array, 292
index, 292
l0_ast.IntLiteral, 294
value, 296
l0_ast.LetDecl, 310
name, 311
type, 311
value, 312
l0_ast.LetStmt, 313
name, 314
type, 314
value, 315
l0_ast.MatchArm, 344
body, 346
pattern, 346
l0_ast.MatchStmt, 348
arms, 349
expr, 349
l0_ast.Module, 350
decls, 351
filename, 351
imports, 351
name, 351
l0_ast.NewExpr, 361
args, 363
type_ref, 363
l0_ast.Node, 363
span, 365
l0_ast.NullLiteral, 367
l0_ast.Param, 369
name, 371
type, 371
l0_ast.ParenExpr, 372
inner, 373
l0_ast.Pattern, 394

l0_ast.ReturnStmt, 402
    value, 403
l0_ast.Span, 426
    end_column, 427
    end_line, 427
    start_column, 427
    start_line, 427
l0_ast.Stmt, 429
l0_ast.StringLiteral, 438
    value, 439
l0_ast.StructDecl, 444
    fields, 445
    name, 445
l0_ast.TopLevelDecl, 474
l0_ast.TryExpr, 476
    expr, 477
l0_ast.TypeAliasDecl, 480
    name, 481
    target, 481
l0_ast.TypeExpr, 482
    type_ref, 483
l0_ast.TypeRef, 484
    is_nullable, 485
    module_path, 485
    name, 485
    name_qualifier, 485
    pointer_depth, 485
l0_ast.UnaryOp, 491
    op, 492
    operand, 492
l0_ast.VariantPattern, 493
    module_path, 494
    name, 494
    name_qualifier, 494
    vars, 494
l0_ast.VarRef, 495
    module_path, 496
    name, 496
    name_qualifier, 496
l0_ast.WhileStmt, 500
    body, 501
    cond, 501
l0_ast.WildcardPattern, 502
l0_ast.WithItem, 503
    cleanup, 504
    init, 504
l0_ast.WithStmt, 506
    body, 507
    cleanup_body, 507
    items, 507
l0_ast_printer, 26
    _format_span, 27
    format_module, 27
    format_node, 27
l0_backend, 28
l0_backend.Backend, 83
    __post_init__, 89
    _alignof_expr_for_type, 89
    _build_type_dependency_graph, 89
    _collect_reassigned_arc_params, 89
    _convert_expr_with_expected_type, 90
    _current_func_result, 123
    _current_scope, 123
    _emit_bare_variant_static_initializer, 90
    _emit_binary_op, 90
    _emit_block, 91
    _emit_block_sequence, 91
    _emit_case, 91
    _emit_case_literal, 92
    _emit_checked_pointer_expr, 92
    _emit_cleanup_at_scope_exit, 92
    _emit_cleanup_for_loop_exit, 93
    _emit_cleanup_for_return, 93
    _emit_condition_branch, 93
    _emit_condition_expr, 94
    _emit_condition_value, 94
    _emit_const_constructor, 94
    _emit_const_struct_constructor, 95
    _emit_const_variant_constructor, 95
    _emit_copy_expr_with_retains, 96
    _emit_drop, 96
    _emit_enum_cleanup, 96
    _emit_expr, 97
    _emit_expr_with_expected_type, 97
    _emit_for, 98
    _emit_function_declaration, 98
    _emit_function_declarations, 98
    _emit_function_definition, 98
    _emit_function_definitions, 99
    _emit_if_else, 99
    _emit_inline_with_header_item, 99
    _emit_let, 99
    _emit_let_declaration, 100
    _emit_let_declarations, 100
    _emit_let_initializer, 100, 123
    _emit_line_directive, 101
    _emit_lvalue_with_caching, 101
    _emit_main_wrapper_if_needed, 102
    _emit_match, 102
    _emit_new_expr, 102
    _emit_ord_intrinsic, 102
    _emit_owned_expr_with_expected_type, 103
    _emit_pattern_bindings, 103
    _emit_pointer_index_lvalue, 104
    _emit_reassignment, 104
    _emit_retain_for_copied_value, 104
    _emit_return, 105
    _emit_sizeof_intrinsic, 105
    _emit_stmt, 106
    _emit_struct_cleanup, 106
    _emit_struct_constructor, 106
    _emit_unwrap, 107
    _emit_value_cleanup, 107
    _emit_variant_constructor, 108
    _emit_while, 108
    _emit_with, 108

_emit_with_cleanup_from_scope, 109
_emit_with_cleanup_header_let_assign, 109
_emit_with_cleanup_header_let_predecl, 109
_expect_expr_type, 110
_extract_value_type_dependencies, 110
_find_cycle_details, 111
_find_enum_decl, 111
_find_struct_decl, 112
_fresh_label, 112
_gen_if_else_branch, 112
_has_side_effects, 113
_int_type_size, 113
_is_binary_op_enabled, 114
_is_extern_function, 114
_is_int_assignable, 114
_is_place_expr, 115
_is_unwrap_cast_from_place, 115
_iter_body_stmts, 115
_label_counter, 123
_lookup_local_var_type, 116
_lookup_owned_local_name, 116
_lookup_symbol, 116
_loop_cleanup_scope_stack, 123
_loop_label_stack, 123
_materialize_arc_temp, 117
_needs_arc_temp, 117
_next_stmt_unreachable, 123, 124
_pointer_type_or_none, 118
_pop_scope, 118
_push_scope, 118
_register_inline_with_cleanup, 118
_resolve_let_type, 118
_resolve_type_ref, 119
_scope_chain_has_cleanup, 119
_should_materialize_arc_temp, 119
_sizeof_expr_for_type, 120
_switch_depth, 124
_topological_sort, 120
_try_emit_constructor, 120
_try_emit_intrinsic, 121
_types_equal, 121
analysis, 124
current_module, 124
emitter, 124
find_variant_decl, 121
generate, 122
ice, 122
l0_bool
l0_runtime.h, 676
l0_byte
l0_runtime.h, 676
l0_c_emitter, 28
l0_c_emitter.CCodeBuilder, 147
dedent, 148
emit, 148
emit_raw, 148
indent, 148
indent_level, 149
indent_str, 149
lines, 149
to_string, 149
l0_c_emitter.CEmitter, 151
_active_line_directive, 199
_collect_opt_wrappers_from_type, 159
_emit_cleanup_by_type, 159
_emit_enum_cleanup_switch, 159
_emit_enum_value_cleanup, 160
_emit_field_cleanup, 160
_emit_optional_wrapper, 160
_enum_has_arc_data, 160
_get_enum_info, 161
_get_struct_info, 161
_is_early_inner, 162
_is_niche_nullable, 162
_iter_variant_cleanup_fields, 162
_opt_emitted, 199
_opt_key_for_type, 162
_opt_wrapper_name_for_inner, 163
_opt_wrappers, 199
_string_token_to_c_bytes_and_len, 163
_tmp_counter, 199
analysis, 199
C_KEYWORDS, 199
current_module, 199
emit_alloc_obj, 163
emit_assignment, 163
emit_binary_op, 164
emit_block_end, 164
emit_block_start, 164
emit_bool_literal, 164
emit_byte_literal, 165
emit_case_label, 165
emit_cast, 165
emit_checked_int_add, 166
emit_checked_int_div, 166
emit_checked_int_mod, 166
emit_checked_int_mul, 166
emit_checked_int_sub, 166
emit_checked_narrow_cast, 167
emit_checked_pointer_assignment, 167
emit_checked_ptr_access, 167
emit_checked_ptr_access_for_base, 167
emit_checked_ptr_index_access, 167
emit_comment, 168
emit_condition_binary_op, 168
emit_condition_pointer_null_check, 168
emit_condition_string_compare_call, 168
emit_const_bool_literal, 168
emit_const_string_literal, 169
emit_default_label, 169
emit_deref_lvalue, 169
emit_discard_expr, 169
emit_drop_begin_expr, 170
emit_drop_finish_call, 170
emit_else, 170
emit_enum, 170

emit_enum_cleanup, 171
emit_enum_payload_field_access, 171
emit_enum_tag, 171
emit_enum_tag_access, 171
emit_enum_variant_init, 172
emit_exit_switch, 172
emit_expr_stmt, 172
emit_field_access, 172
emit_field_lvalue, 173
emit_for_loop_end, 173
emit_for_loop_start, 173
emit_forward_decls, 173
emit_function_call, 173
emit_function_declaration, 174
emit_function_definition_footer, 174
emit_function_definition_header, 174
emit_goto, 174
emit_header, 175
emit_if_header, 175
emit_index_lvalue, 175
emit_int_literal, 175
emit_label, 176
emit_let_decl, 176
emit_let_declaration, 176
emit_line_directive, 177
emit_main_wrapper, 177
emit_match_scrutinee_decl, 177
emit_match_switch_start, 178
emit_module_comment, 178
emit_module_separator, 178
emit_negated_condition, 178
emit_none_value_for_nullable, 178
emit_null_assignment, 179
emit_null_check_eq, 179
emit_null_check_ne, 179
emit_null_literal, 179
emit_optional_has_value, 180
emit_optional_value, 180
emit_optional_wrapper_for_defined_type, 180
emit_optional_wrappers, 180
emit_ord, 181
emit_paren_expr, 181
emit_pattern_binding_init, 181
emit_pointer_assignment, 182
emit_pointer_null_check, 182
emit_pointer_type, 182
emit_return_stmt, 183
emit_section_comment, 183
emit_sizeof_type, 183
emit_some_value_for_nullable, 184
emit_string_compare_call, 184
emit_string_concat_call, 184
emit_string_equals_call, 184
emit_string_literal, 184
emit_string_release, 185
emit_string_retain, 185
emit_struct, 185
emit_struct_cleanup, 187
emit_struct_constructor, 187
emit_struct_constructor_for_type, 187
emit_struct_init, 187
emit_struct_init_from_fields, 188
emit_struct_static_initializer, 188
emit_struct_static_initializer_for_type, 188
emit_switch_end, 188
emit_switch_start, 189
emit_temp_decl, 189
emit_top_level_let_cleanup, 189
emit_try_check_niche, 189
emit_try_check_value, 190
emit_try_extract_value, 190
emit_type, 190
emit_unary_op, 191
emit_unreachable_comment, 191
emit_unreachable_marker, 191
emit_unwrap_opt, 191
emit_unwrap_ptr, 192
emit_value_cleanup, 192
emit_var_ref, 192
emit_variant_constructor, 192
emit_variant_constructor_for_type, 193
emit_variant_static_initializer, 193
emit_variant_static_initializer_for_type, 193
emit_while_header, 193
emit_widen_int, 194
emit_zero_init, 194
find_variant_decl, 195
fresh_tmp, 195
get_output, 195
ice, 195
is_niche_nullable, 196
mangle_enum_name, 196
mangle_function_name, 197
mangle_identifier, 197
mangle_let_name, 197
mangle_struct_name, 198
out, 200
prepare_optional_wrappers, 198
restore_active_line_directive, 198
set_analysis, 198
l0_compilation, 28
l0_compilation.CompilationUnit, 211
__contains__, 212
entry_module, 212
entry_name, 212
modules, 212
l0_context, 28
l0_context.CompilationContext, 209
default, 210
emit_line_directives, 210
log_level, 210
log_rich_format, 210
rt_check_basic, 210
rt_unchecked, 210
trace_arc, 210
trace_memory, 210

l0_context.LogLevel, 335
DEBUG, 336
ERROR, 336
INFO, 336
SILENT, 336
WARNING, 336
L0_DEFINED_l0_sys_rt_RtFileInfo
l0_runtime.h, 674
L0_DEFINED_l0_sys_rt_RtTimeParts
l0_runtime.h, 674
l0_diagnostics, 29
diag_from_node, 29
diag_from_token, 29
DIAGNOSTIC_CODE_FAMILIES, 30
l0_diagnostics.Diagnostic, 218
column, 219
end_column, 219
end_line, 220
filename, 220
format, 219
kind, 220
line, 220
message, 220
module_name, 220
l0_double
l0_runtime.h, 676
l0_driver, 30
load_source_utf8, 30
l0_driver.ImportCycleError, 290
l0_driver.L0Driver, 306
__init__, 307
_load_single_file, 308
_parse_source, 308
analyze, 308
build_compilation_unit, 309
context, 310
diagnostics, 310
load_module, 309
search_paths, 310
l0_driver.SourceEncodingError, 422
__init__, 423
message, 423
path, 423
l0_expr_types, 31
l0_expr_types.ExpressionTypeChecker, 245
__post_init__, 249
_alive_scopes, 262
_binary_equality, 249
_binary_expect_both_bool, 249
_binary_expect_both_int, 250
_breakable_loop_depth, 262
_can_assign, 250
_case_literal_info, 250
_check_block, 250
_check_dead_liveness_block, 251
_check_expr_liveness, 251
_check_function, 251
_check_loop_iteration, 251
_check_return, 251
_check_stmt, 252
_check_update_from_backedge_states, 252
_cleanup_header_ref_guard_stack, 262
_clone_alive_scopes, 252
_current_func_env, 262
_current_func_type, 262
_declare_local, 252
_decode_escaped_bytes, 253
_describe_lvalue, 253
_diagnostic, 253
_error, 253
_expr_contains_try, 253
_get_const_int_for_explicit_cast, 253
_infer_binary, 254
_infer_call, 254
_infer_cast, 254
_infer_expr, 254
_infer_field_access, 255
_infer_index, 255
_infer_new, 255
_infer_ord_intrinsic, 255
_infer_sizeof_intrinsic, 255
_infer_struct_constructor, 255
_infer_try, 256
_infer_type_expr, 256
_infer_unary, 256
_infer_var_ref, 256
_infer_variant_constructor, 256
_is_bool, 256
_is_const_null_for_explicit_cast, 257
_is_guarded_cleanup_header_ref, 257
_is_int_assignable, 257
_is_nullable_or_ptr, 257
_is_string, 257
_is_void, 257
_liveness_diagnostics_only, 262
_local_scopes, 262
_lookup_alive, 258
_lookup_local, 258
_lookup_local_scope_index, 258
_loop_flow_capture_stack, 262
_loop_liveness_fixed_point, 258
_make_param_alive_scope, 258
_make_param_scope, 258
_meet_alive_scopes, 259
_meet_alive_state, 259
_next_stmt_unreachable, 262
_pop_scope, 259
_push_scope, 259
_reject_name_qualifier, 259
_resolve_type_ref, 259
_return_paths, 262
_set_alive, 260
_stmt_contains_try, 260
_store_sizeof_target, 260
_store_var_resolution, 260
_suppress_diagnostics, 262

_suppress_liveness_diagnostics, 263
_try_infer_intrinsic, 260
_try_resolve_type_name, 260
_types_equal, 261
_validate_match_variant_pattern, 261
_warn, 261
analysis, 263
bool_type, 263
byte_type, 263
check, 261
diagnostics, 263
enum_infos, 263
expr_types, 263
func_envs, 263
func_types, 263
int_type, 263
module_envs, 264
string_type, 264
struct_infos, 264
void_type, 264
l0_expr_types.LoopFlowCapture, 339
break_states, 340
continue_states, 340
l0_expr_types.StmtFlow, 432
FALLTHROUGH, 432
RETURNS, 432
STOPS, 433
l0_float
l0_runtime.h, 676
l0_int
l0_runtime.h, 676
l0_internal_error, 31
l0_internal_error.ICELocation, 283
filename, 284
span, 284
l0_internal_error.InternalCompilerError, 292
__init__, 293
format, 294
loc, 294
message, 294
l0_lexer, 31
_EXPR_ENDING_TOKENS, 33
_is_ascii_ident_part, 32
_is_ascii_ident_start, 32
_logical_predecessor_ends_expression, 32
HEX_CHARS, 33
is_reserved_keyword, 32
KEYWORDS, 33
OCT_CHARS, 33
l0_lexer.Lexer, 315
__init__, 317
_advance, 317
_at_end, 317
_collect_invalid_characters, 317
_error, 317
_lexer_error_token, 317
_next_token, 318
_peek, 318
_peek_next, 318
_prev_ends_expression, 320
_read_byte_literal, 318
_read_number, 318
_read_string_literal, 318
_read_valid_char_escape, 319
_skip_ws_and_comments, 319
_wrap_new_diagnostics, 319
column, 320
diagnostics, 320
filename, 320
from_source, 319
index, 320
length, 320
line, 320
source, 320
tokenize, 319
l0_lexer.Token, 459
__repr__, 460
column, 460
diagnostic, 460
diagnostics, 461
kind, 461
line, 461
recovery, 461
text, 461
l0_lexer.TokenKind, 462
AMP, 466
ANDAND, 466
ARROW_FUNC, 466
ARROW_MATCH, 466
AS, 466
BANG, 466
BREAK, 466
BYTE, 466
CARET, 467
CASE, 467
CLEANUP, 467
COLON, 467
COMMA, 467
CONTINUE, 467
DOT, 467
DOUBLE_COLON, 467
DROP, 467
ELSE, 467
ENUM, 468
EOF, 468
EQ, 468
EQEQ, 468
EXTERN, 468
FALSE, 468
FOR, 468
FUNC, 468
FUTURE_EXTENSION, 468
GE, 468
GT, 469
IDENT, 469
IF, 469

IMPORT, 469
INT, 469
LBRACE, 469
LBRACKET, 469
LE, 469
LET, 469
LEXER_ERROR, 469
LPAREN, 470
LSHIFT, 470
LT, 470
MATCH, 470
MINUS, 470
MODULE, 470
MODULO, 470
NE, 470
NEW, 470
NULL, 470
ORELSE, 471
PIPE, 471
PLUS, 471
QUESTION, 471
RBRACE, 471
RBRACKET, 471
RETURN, 471
RPAREN, 471
RSHIFT, 471
SEMI, 471
SLASH, 472
STAR, 472
STRING, 472
STRUCT, 472
TILDE, 472
TRUE, 472
TYPE, 472
UNDERSCORE, 472
WHILE, 472
WITH, 472
l0_locals, 33
l0_locals.FunctionEnv, 281
func, 281
module_name, 281
root_scope, 282
l0_locals.LocalKind, 325
LOCAL, 326
PARAM, 326
PATTERN_VAR, 326
l0_locals.LocalScopeResolver, 326
__init__, 327
_bind_pattern_vars, 328
_build_function_env, 328
_declare, 328
_visit_block, 328
_visit_case_arm, 328
_visit_case_else, 329
_visit_match_arm, 329
_visit_stmt, 329
_visit_with_stmt, 329
get_block_scope, 329
get_match_arm_scope, 330
modules, 330
resolve, 330
l0_locals.LocalSymbol, 332
decl, 332
kind, 332
name, 333
type_ref, 333
l0_locals.Scope, 406
lookup, 406
parent, 407
symbols, 407
l0_logger, 34
log, 34
log_debug, 34
log_error, 34
log_info, 35
log_stage, 35
log_warning, 35
l0_long
l0_runtime.h, 676
l0_name_resolver, 36
l0_name_resolver.NameResolver, 359
__init__, 359
_collect_locals, 360
_define_local, 360
_extern_signatures_compatible, 360
_open_imports, 360
cu, 361
resolve, 361
l0_opt_bool, 299
has_value, 300
value, 300
L0_OPT_BOOL_DEFINED
l0_runtime.h, 674
l0_opt_byte, 300
has_value, 300
value, 300
L0_OPT_BYTE_DEFINED
l0_runtime.h, 674
l0_opt_int, 301
has_value, 301
value, 301
L0_OPT_INT_DEFINED
l0_runtime.h, 675
l0_opt_string, 301
has_value, 302
value, 302
L0_OPT_STRING_DEFINED
l0_runtime.h, 675
L0_OPT_STRING_EMPTY
l0_runtime.h, 721
L0_OPT_STRING_NULL
l0_runtime.h, 721
l0_parser, 36
token_len, 36
l0_parser._ParseSyncException, 66
l0_parser.Parser, 374

_RESERVED_BINARY_OPS, 389
__init__, 377
_advance, 378
_at_case_arm_start, 378
_at_end, 378
_check, 378
_check_reserved_binary_op, 378
_emit_lexer_error, 378
_emit_remaining_lexer_errors, 378
_error, 379
_error_bail, 379
_error_unexpected, 379
_expect, 380
_expect_semicolon, 380
_expect_variable_name, 380
_extend_span, 380
_get_dotted_module_name, 380
_is_builtin_type_name, 381
_is_unambiguous_type_start, 381
_last, 381
_last_token, 389
_logical_token_at_cursor, 381
_lookahead_is_type_suffix, 381
_match, 382
_parse_add_expr, 382
_parse_and_expr, 382
_parse_block, 382
_parse_break_stmt, 382
_parse_call_argument, 382
_parse_case_literal, 383
_parse_case_stmt, 383
_parse_cast_expr, 383
_parse_continue_stmt, 383
_parse_drop_stmt, 383
_parse_enum, 383
_parse_equality_expr, 384
_parse_expr, 384
_parse_extern_func, 384
_parse_for_stmt, 384
_parse_for_update_stmt, 384
_parse_function, 384
_parse_if_stmt, 384
_parse_let_stmt, 385
_parse_match_stmt, 385
_parse_mul_expr, 385
_parse_or_expr, 385
_parse_pattern, 385
_parse_postfix_expr, 385
_parse_primary_expr, 385
_parse_rel_expr, 386
_parse_return_stmt, 386
_parse_simple_stmt, 386
_parse_stmt, 386
_parse_struct, 386
_parse_top_level_decl, 386
_parse_top_level_let, 386
_parse_type, 387
_parse_type_alias, 387
_parse_unary_expr, 387
_parse_while_stmt, 387
_parse_with_item, 387
_parse_with_stmt, 387
_peek, 387
_skip_lexer_errors, 388
_span_start, 388
_sync_case_invalid_arm, 388
_sync_stmt, 388
_sync_top_level, 388
_try_parse_qualified_name, 388
diagnostics, 390
eof_aborted, 390
filename, 390
from_source, 389
index, 390
parse_module, 389
tokens, 390
l0_paths, 37
l0_paths.SourceSearchPaths, 423
add_project_root, 424
add_system_root, 424
module_relpath, 425
project_roots, 425
resolve, 425
system_roots, 425
L0_PRIMITIVE_TYPES
l0_types, 43
l0_resolve, 37
_is_imported, 37
resolve_symbol, 37
resolve_type_ref, 38
l0_resolve.ResolveErrorKind, 401
AMBIGUOUS_SYMBOL, 401
MODULE_NOT_IMPORTED, 401
UNKNOWN_MODULE, 402
UNKNOWN_SYMBOL, 402
l0_resolve.SymbolResolution, 458
ambiguous_modules, 459
error, 459
module_name, 459
name, 459
symbol, 459
l0_resolve.TypeResolution, 487
ambiguous_modules, 488
error, 488
module_name, 488
name, 488
symbol, 488
type, 488
l0_resolve.TypeResolveErrorKind, 489
AMBIGUOUS_TYPE, 490
INVALID_NULLABLE_VOID, 490
MODULE_NOT_IMPORTED, 490
NOT_A_TYPE, 490
UNKNOWN_MODULE, 490
UNKNOWN_TYPE, 490
UNRESOLVED_ALIAS, 490

VARIANT_AS_TYPE, 490
l0_runtime.h
_L0_TAG_ENUM, 671
_L0_TAG_OPT, 671
_L0_TAG_PTR, 671
_L0_TAG_STRUCT, 671
_RT_ACCESS_READ, 672
_RT_ACCESS_UNTRACKED_OK, 672
_RT_ACCESS_WRITE, 672
_RT_ALIGNOF, 672
_RT_ALLOC_INIT_CAP, 672
_RT_ALLOC_LIVE, 672
_RT_ALLOC_POOLED, 672
_RT_ALLOC_QUARANTINED, 672
_RT_ALLOC_TOMBSTONE, 672
_RT_MEM_ARC, 672
_RT_MEM_FOREIGN, 673
_RT_MEM_NEW, 673
_RT_MEM_RAW, 673
_RT_MEM_SENTINEL, 720
_RT_MEM_STATIC, 673
_RT_QUARANTINE_MAX_BYTES, 673
_RT_QUARANTINE_MAX_COUNT, 673
_RT_REC_POOL_CHUNK, 673
_RT_TRACE_ARC, 673
_RT_TRACE_ARC_LOC, 673
_RT_TRACE_MEM, 674
_RT_TRACE_MEM_LOC, 674
_l0_sh_tag_bool, 719
_l0_sh_tag_byte, 719
_l0_sh_tag_data, 719
_l0_sh_tag_int, 719
_l0_sh_tag_string, 719
_rt_alloc_hash, 677
_rt_alloc_next_generation, 719
_rt_alloc_obj, 677
_rt_alloc_record, 675
_rt_alloc_record_cold, 675
_rt_alloc_record_size_check, 675
_rt_alloc_string, 677
_rt_alloc_table, 719
_rt_alloc_table_cap, 719
_rt_alloc_table_cnt, 720
_rt_alloc_table_insert, 678
_rt_alloc_table_lookup, 678
_rt_alloc_table_rehash, 678
_rt_alloc_table_remove, 678
_rt_alloc_table_tombstones, 720
_rt_alloc_tree_find_containing, 678
_rt_alloc_tree_glb, 678
_rt_alloc_tree_insert, 678
_rt_alloc_tree_lub, 679
_rt_alloc_tree_remove, 679
_rt_alloc_tree_root, 720
_rt_argc, 720
_rt_argv, 720
_rt_check_index_ptr_site, 679
_rt_check_index_ptr_site_slow, 679
_rt_check_ptr_align, 680
_rt_check_ptr_site, 680
_rt_check_ptr_site_slow, 681
_rt_check_record_writeable, 681
_rt_cold_chunks, 720
_rt_drop_begin_impl, 681
_rt_drop_finish_impl, 681
_rt_evict_quarantine, 681
_rt_fmix32, 681
_rt_fold_u64_to_u32_fmix, 682
_rt_free_string, 682
_rt_hash_bool, 682
_rt_hash_byte, 683
_rt_hash_data, 683
_rt_hash_int, 683
_rt_hash_string, 684
_rt_hash_tag8, 684
_rt_iadd, 685
_rt_idiv, 685
_rt_imod, 685
_rt_imul, 686
_rt_index_access_mode, 686
_rt_index_allows_untracked, 686
_rt_index_element_size, 686
_rt_index_offset, 686
_rt_init_args, 686
_rt_init_heap_string, 687
_rt_isub, 687
_rt_l0_string_from_const_literal, 688
_rt_narrow_l0_byte, 688
_rt_new_l0_string, 688
_rt_panic, 689
_rt_panic_fmt, 689
_rt_panic_invalid_access, 689
_rt_panic_invalid_drop, 689
_rt_panic_invalid_release, 689
_rt_pid_to_l0_int, 690
_rt_print, 690
_rt_promote_new_alloc, 690
_rt_ptr_is_aligned, 690
_rt_ptr_site, 676
_rt_ptr_site_store, 690
_rt_quarantine_alloc_record, 691
_rt_quarantine_bytes, 720
_rt_quarantine_count, 720
_rt_quarantine_head, 720
_rt_quarantine_tail, 721
_rt_range_contains, 691
_rt_realloc_string, 691
_rt_realloc_tracked_alloc, 691
_rt_rec_cold, 673
_rt_rec_free_list, 721
_rt_rec_new, 691
_rt_rec_pool_chunks, 721
_rt_rec_recycle, 692
_rt_release_tracked_alloc, 692
_rt_required_access_mode, 692
_rt_required_align, 692

_rt_required_size, 692
_rt_sh_key, 721
_rt_siphash_key_t, 676
_rt_siphash_tag8_t, 676
_rt_stream_write_some, 692
_rt_string_bytes, 693
_rt_time_to_l0_int_nsec, 693
_rt_time_to_l0_int_sec, 693
_rt_time_write_parts, 693
_rt_trace_invalid_drop, 694
_rt_track_alloc_record, 694
_rt_track_alloc_record_kind, 694
_rt_track_arc_bytes, 694
_rt_track_static_bytes, 694
_rt_tree_insert_at, 695
_rt_tree_merge, 695
_rt_tree_prio_for, 695
_rt_tree_remove_at, 695
_rt_untrack_arc_alloc, 695
_rt_validate_derived_ptr, 695
_unwrap_opt, 695
_unwrap_ptr, 696
DEA_RUNTIME_FCLOSE, 674
DEA_RUNTIME_FILE_INFO_STAT, 674
DEA_RUNTIME_REMOVE, 674
l0_bool, 676
l0_byte, 676
L0_DEFINED_l0_sys_rt_RtFileInfo, 674
L0_DEFINED_l0_sys_rt_RtTimeParts, 674
l0_double, 676
l0_float, 676
l0_int, 676
l0_long, 676
L0_OPT_BOOL_DEFINED, 674
L0_OPT_BYTE_DEFINED, 674
L0_OPT_INT_DEFINED, 675
L0_OPT_STRING_DEFINED, 675
L0_OPT_STRING_EMPTY, 721
L0_OPT_STRING_NULL, 721
l0_short, 676
L0_STRING_CONST, 675
L0_STRING_EMPTY, 721
L0_STRING_K_HEAP, 675
L0_STRING_K_STATIC, 675
l0_tiny, 677
l0_uint, 677
l0_ulong, 677
L0_UNREACHABLE, 675
l0_ushort, 677
rt_abort, 696
rt_alloc, 696
rt_array_element, 697
rt_calloc, 697
rt_delete_file, 698
rt_errno, 698
rt_exit, 698
rt_file_info, 699
rt_flush_stderr, 699
rt_flush_stdout, 699
rt_free, 699
rt_get_argc, 700
rt_get_argv, 700
rt_get_env_var, 700
rt_get_pid, 701
rt_hash_bool, 701
rt_hash_byte, 701
rt_hash_data, 702
rt_hash_int, 702
rt_hash_opt_bool, 702
rt_hash_opt_byte, 703
rt_hash_opt_int, 703
rt_hash_opt_ptr, 703
rt_hash_opt_string, 704
rt_hash_ptr, 704
rt_hash_string, 705
rt_memcmp, 705
rt_memcpy, 705
rt_memset, 706
rt_print, 706
rt_print_bool, 707
rt_print_bool_stderr, 707
rt_print_int, 707
rt_print_int_stderr, 707
rt_print_stderr, 708
rt_println, 708
rt_println_stderr, 708
rt_rand, 708
rt_read_char, 709
rt_read_file_all, 709
rt_read_line, 709
rt_realloc, 710
rt_register_foreign, 710
rt_srand, 710
rt_stderr_write, 711
rt_stdin_read, 711
rt_stdout_write, 711
rt_string_bytes_ptr, 712
rt_string_compare, 712
rt_string_concat, 713
rt_string_equals, 713
rt_string_from_byte, 713
rt_string_from_byte_array, 714
rt_string_get, 714
rt_string_release, 715
rt_string_retain, 715
rt_string_slice, 715
rt_strlen, 716
rt_system, 716
rt_time_local_is_dst, 716
rt_time_local_offset_sec, 717
rt_time_monotonic, 717
rt_time_monotonic_supported, 717
rt_time_unix, 718
rt_unregister_foreign, 718
rt_write_file_all, 718
l0_scope_context, 38

l0_scope_context.ScopeContext, 408
add_declared, 409
add_owned, 410
declared_vars, 410
owned_vars, 410
parent, 410
remove_owned, 410
with_cleanup_block, 411
with_cleanup_in_progress, 411
with_cleanup_inline, 411
l0_short
l0_runtime.h, 676
l0_signatures, 39
l0_signatures.EnumInfo, 230
enum_type, 230
variants, 230
l0_signatures.EnumVariantInfo, 235
field_types, 236
name, 236
l0_signatures.SignatureResolver, 415
__init__, 416
_detect_value_type_cycles, 417
_emit, 417
_enum_type_for_variant_symbol, 417
_extract_value_type_dependencies, 417
_infer_literal_type, 417
_resolve_enum, 417
_resolve_func, 418
_resolve_let, 418
_resolve_module_signatures, 418
_resolve_struct, 418
_resolve_type_alias, 418
_resolve_type_alias_symbol, 418
_resolve_type_ref, 419
_symbol_is_zero_arg_enum_variant, 419
cu, 420
module_envs, 420
resolve, 419
l0_signatures.StructFieldInfo, 446
name, 447
type, 447
l0_signatures.StructInfo, 449
fields, 449
struct_type, 449
l0_string, 302
bytes, 303
data, 303
h_str, 303
int, 303
kind, 303
len, 304
s_str, 304
L0_STRING_CONST
l0_runtime.h, 675
L0_STRING_EMPTY
l0_runtime.h, 721
l0_string_escape, 39
_HEX_CHARS, 40
_OCT_CHARS, 40
_SIMPLE_ESCAPES, 40
_TRIGRAPH_THIRD_BYTES, 41
decode_l0_string_token, 39
encode_c_string_bytes, 40
l0_string_escape.EscapeDecodeError, 237
code, 238
details, 238
L0_STRING_K_HEAP
l0_runtime.h, 675
L0_STRING_K_STATIC
l0_runtime.h, 675
l0_symbols, 41
l0_symbols.ModuleEnv, 353
all, 354
ambiguous_imports, 354
diagnostics, 354
imported, 354
locals, 355
module, 355
name, 354
l0_symbols.Symbol, 453
kind, 453
module, 453
name, 454
node, 454
type, 454
l0_symbols.SymbolKind, 456
ENUM, 457
ENUM_VARIANT, 457
FUNC, 457
LET, 457
MODULE, 457
STRUCT, 457
TYPE_ALIAS, 457
l0_sys_rt_RtFileInfo, 304
exists, 305
is_dir, 305
is_file, 305
mtime_nsec, 305
mtime_sec, 305
size, 305
l0_sys_rt_RtTimeParts, 306
nsec, 306
sec, 306
l0_tiny
l0_runtime.h, 677
l0_types, 41
_NULL_TYPE, 43
format_type, 42
get_builtin_type, 42
get_null_type, 42
L0_PRIMITIVE_TYPES, 43
l0_types.BuiltinType, 132
name, 133
l0_types.EnumType, 231
module, 232
name, 232

l0_types.FuncType, 282
    params, 283
    result, 283
l0_types.NullableType, 366
    inner, 367
l0_types.NullType, 368
l0_types.PointerType, 397
    inner, 398
l0_types.StructType, 451
    module, 452
    name, 452
l0_types.Type, 477
l0_uint
    l0_runtime.h, 677
l0_ulong
    l0_runtime.h, 677
L0_UNREACHABLE
    l0_runtime.h, 675
l0_ushort
    l0_runtime.h, 677
l0c, 43
    _CLI_CANONICAL_SHORT_VALUE_OPTIONS, 60
    _CLI_LONG_VALUE_OPTIONS, 60
    _CLI_NAMESPACED_SHORT_VALUE_OPTIONS, 60
    _CLI_OTHER_SHORT_VALUE_OPTIONS, 61
    _CLI_VALUE_OPTIONS, 61
    _add_all_modules_arg, 45
    _add_codegen_arg, 46
    _add_runtime_args, 46
    _add_target_args, 46
    _check_entry_main_for_build, 46
    _compile_generated_c, 47
    _compiler_flag_family, 47
    _default_executable_name, 47
    _dump_tokens_for_file, 48
    _emit_diagnostic, 48
    _emit_temporary_source_cleanup_failure, 48
    _emit_verbose_compiler_identity, 48
    _find_cc, 49
    _format_token_dump_text, 49
    _get_module_names, 49
    _get_optimize_flag, 49
    _init_env_defaults, 50
    _is_valid_module_name, 50
    _is_windows_host, 50
    _load_file_lines, 50
    _normalize_cli_argv, 51
    _output_flags, 51
    _remove_temporary_c_source, 51
    _run_analysis, 52
    _runtime_include_flags, 52
    _runtime_library_flags, 52
    _scan_cli_presentation_options, 52
    _split_c_options, 53
    _split_cli_and_program_args, 53
    _validate_mode_scoped_flags, 53
    _validate_reserved_canonical_flags, 54
    _validate_runtime_library_path, 54
    _validated_temporary_directory, 54
    _write_temporary_c_source, 55
    build_compilation_context, 55
    build_search_paths, 55
    cmd_ast, 56
    cmd_build, 56
    cmd_check, 56
    cmd_codegen, 57
    cmd_compile, 57
    cmd_run, 57
    cmd_sym, 58
    cmd_tok, 58
    cmd_type, 58
    compiler_identity_text, 59
    main, 59
    print_diagnostic_list, 59
    print_diagnostic_with_snippet, 60
    print_diagnostics, 60
l0c._TemporarySourceWriteError, 71
    __init__, 72
    retained_path, 72
l0c.l0
    main, 965
l0c_append_c_option_words
    l0c_lib.l0, 966
l0c_cmd_ast
    l0c_lib.l0, 967
l0c_cmd_check
    l0c_lib.l0, 967
l0c_cmd_gen
    l0c_lib.l0, 967
l0c_cmd_nyi
    l0c_lib.l0, 968
l0c_cmd_sym
    l0c_lib.l0, 968
l0c_cmd_tok
    l0c_lib.l0, 968
l0c_cmd_type
    l0c_lib.l0, 969
l0c_collect_c_option_words
    l0c_lib.l0, 969
l0c_copy_collector
    l0c_lib.l0, 969
l0c_dump_tokens_for_path
    l0c_lib.l0, 970
l0c_fs_join_child
    compiler_filesystem.l0, 922
l0c_fs_mkdir
    compiler_filesystem.l0, 922
l0c_fs_path_kind_follow
    compiler_filesystem.l0, 922
l0c_fs_path_kind_nofollow
    compiler_filesystem.l0, 922
l0c_fs_remove_empty_dir
    compiler_filesystem.l0, 922
l0c_fs_remove_regular_file

compiler_filesystem.l0, 922
l0c_fs_resolve_trusted_temp_parent
compiler_filesystem.l0, 923
l0c_lib.l0
l0c_append_c_option_words, 966
l0c_cmd_ast, 967
l0c_cmd_check, 967
l0c_cmd_gen, 967
l0c_cmd_nyi, 968
l0c_cmd_sym, 968
l0c_cmd_tok, 968
l0c_cmd_type, 969
l0c_collect_c_option_words, 969
l0c_copy_collector, 969
l0c_dump_tokens_for_path, 970
l0c_log_effective_roots, 970
l0c_merge_c_option_words, 970
l0c_prepare_input, 971
l0c_print_module_token_header, 971
l0c_print_token, 971
prepared_input_free, 971
prepared_input_free_opt, 973
run_with_argv, 973
l0c_log_effective_roots
l0c_lib.l0, 970
l0c_merge_c_option_words
l0c_lib.l0, 970
l0c_prepare_input
l0c_lib.l0, 971
l0c_print_module_token_header
l0c_lib.l0, 971
l0c_print_token
l0c_lib.l0, 971
label_counter
Backend, 82
last_token
ParserState, 393
LBRACE
l0_lexer.TokenKind, 469
LBRACKET
l0_lexer.TokenKind, 469
lc_bind_pattern
locals.l0, 986
lc_func_env_free
locals.l0, 986
lc_func_env_get_block_scope
locals.l0, 986
lc_func_env_get_match_arm_scope
locals.l0, 987
lc_key
locals.l0, 987
lc_make_function_env
locals.l0, 987
lc_scope_create
locals.l0, 988
lc_scope_define
locals.l0, 988
lc_scope_free
locals.l0, 988
lc_scope_lookup
locals.l0, 989
lc_scope_set_block
locals.l0, 989
lc_scope_track
locals.l0, 989
lc_visit_block
locals.l0, 989
lc_visit_case_arm
locals.l0, 990
lc_visit_case_else
locals.l0, 990
lc_visit_match_arm
locals.l0, 990
lc_visit_scoped_stmt
locals.l0, 991
lc_visit_stmt
locals.l0, 991
lc_visit_with
locals.l0, 991
lcm
integer.l0, 557
LE
l0_lexer.TokenKind, 469
left
l0_ast.BinaryOp, 126
len
_l0_h_string, 64
l0_string, 304
len_s
string.l0, 600
length
l0_lexer.Lexer, 320
LexerState, 322
VectorBase, 498
LET
l0_lexer.TokenKind, 469
l0_symbols.SymbolKind, 457
let_decl_free
ast.l0, 785
let_types
l0_analysis.AnalysisResult, 77
SignatureTables, 421
LetDecl, 312
name, 313
span, 313
type_ref, 313
value, 313
level
LogConfig, 334
lexer.l0
escape_char_value, 975
hex_char_to_int, 975
is_escape_char, 976
is_hex_digit, 976
is_ident_part, 976
is_ident_start, 976

is_octal_digit, 977
is_printable_ascii, 977
ls_advance, 977
ls_at_end, 978
ls_clear_pending_error, 978
ls_create, 978
ls_defer_recoverable_error, 978
ls_emit_error, 979
ls_free, 979
ls_has_errors, 979
ls_has_queued_tokens, 980
ls_next_token, 980
ls_peek, 980
ls_peek_next, 980
ls_queue_pending_errors, 981
ls_queue_pending_recovery, 981
ls_queue_terminal_recovery, 981
ls_queue_token, 982
ls_read_byte_literal, 982
ls_read_number, 982
ls_read_string_literal, 982
ls_read_valid_char_escape, 983
ls_skip_invalid_characters, 983
ls_skip_whitespace_and_comments, 983
ls_take_queued_token, 983
tokenize, 984
LEXER_ERROR
l0_lexer.TokenKind, 469
LexerState, 321
column, 322
diags, 322
filename, 322
index, 322
length, 322
line, 322
pending_errors, 322
prev_ends_expression, 322
queued_index, 322
queued_tokens, 322
source, 323
limit
check_trace_log._MessageCollector, 65
line
Diagnostic, 218
l0_diagnostics.Diagnostic, 220
l0_lexer.Lexer, 320
l0_lexer.Token, 461
LexerState, 322
Token, 462
line_end
Diagnostic, 218
linear_map.l0
_islm_entry_by_index, 576
_islm_find_by_key, 576
_lm_entry_by_hk_key, 577
_lm_entry_by_hv_value, 577
_lm_entry_by_index, 577
_lm_grow, 578
_lm_key_by_entry, 578
_lm_new_entry, 578
_lm_set_entry, 579
_lm_value_by_entry, 579
_sslm_entry_by_index, 579
_sslm_find_by_key, 580
islm_contains, 580
islm_create, 580
islm_free, 581
islm_get, 581
islm_key_at, 581
islm_len, 581
islm_remove, 582
islm_set, 582
islm_value_at, 582
lm_contains_key, 583
lm_contains_value, 583
lm_create, 583
lm_free, 584
lm_get, 584
lm_len, 584
lm_remove, 584
lm_set, 586
sslm_contains, 586
sslm_create, 586
sslm_free, 587
sslm_get, 587
sslm_key_at, 587
sslm_len, 587
sslm_remove, 588
sslm_set, 588
sslm_value_at, 588
LinearMapBase, 323
entries, 324
key_size, 324
value_size, 324
LinearMapEntryBase, 324
hash_key, 324
hash_value, 324
lines
CCodeBuilder, 146
l0_c_emitter.CCodeBuilder, 149
lines_s
text.l0, 613
literal
CaseArm, 138
l0_ast.CaseArm, 139
liveness_diagnostics_only
ExpressionTypeChecker, 244
LK_LOCAL
locals.l0, 986
LK_PARAM
locals.l0, 986
LK_PATTERN_VAR
locals.l0, 986
LL_DEBUG
log.l0, 1109
LL_ERROR

log.l0, 1109
LL_INFO
log.l0, 1109
LL_SILENT
log.l0, 1109
LL_WARNING
log.l0, 1109
lm_contains_key
linear_map.l0, 583
lm_contains_value
linear_map.l0, 583
lm_create
linear_map.l0, 583
lm_free
linear_map.l0, 584
lm_get
linear_map.l0, 584
lm_len
linear_map.l0, 584
lm_remove
linear_map.l0, 584
lm_set
linear_map.l0, 586
load_module
l0_driver.L0Driver, 309
load_source_utf8
l0_driver, 30
loaded
DriverState, 221
loc
l0_internal_error.InternalCompilerError, 294
LOCAL
l0_analysis.VarRefResolution, 497
l0_locals.LocalKind, 326
local_now_datetime
time.l0, 622
LocalKind
locals.l0, 986
locals
l0_symbols.ModuleEnv, 355
ModuleEnv, 356
locals.l0
lc_bind_pattern, 986
lc_func_env_free, 986
lc_func_env_get_block_scope, 986
lc_func_env_get_match_arm_scope, 987
lc_key, 987
lc_make_function_env, 987
lc_scope_create, 988
lc_scope_define, 988
lc_scope_free, 988
lc_scope_lookup, 989
lc_scope_set_block, 989
lc_scope_track, 989
lc_visit_block, 989
lc_visit_case_arm, 990
lc_visit_case_else, 990
lc_visit_match_arm, 990
lc_visit_scoped_stmt, 991
lc_visit_stmt, 991
lc_visit_with, 991
LK_LOCAL, 986
LK_PARAM, 986
LK_PATTERN_VAR, 986
LocalKind, 986
locals_env_count, 991
locals_env_find, 992
locals_env_get, 992
locals_free, 992
locals_resolve, 993
locals_env_count
locals.l0, 991
locals_env_find
locals.l0, 992
locals_env_get
locals.l0, 992
locals_free
locals.l0, 992
locals_resolve
locals.l0, 993
locals_result
AnalysisResult, 74
LocalsResult, 331
by_name, 331
envs, 331
LocalSymbol, 333
binding_slot, 333
decl_id, 333
decl_ptr, 333
kind, 334
name, 334
log
CliOptions, 204
l0_logger, 34
log.l0
LL_DEBUG, 1109
LL_ERROR, 1109
LL_INFO, 1109
LL_SILENT, 1109
LL_WARNING, 1109
log_config_create, 1110
log_debug, 1110
log_error, 1110
log_format_stage, 1110
log_info, 1111
log_level_from_verbosity, 1111
log_level_name, 1111
log_msg, 1112
log_stage, 1112
log_warning, 1112
LogLevel, 1109
log_config_create
log.l0, 1110
log_debug
l0_logger, 34
log.l0, 1110

log_error
    l0_logger, 34
    log.l0, 1110
log_format_stage
    log.l0, 1110
log_info
    l0_logger, 35
    log.l0, 1111
log_level
    l0_context.CompilationContext, 210
log_level_from_verbosity
    log.l0, 1111
log_level_name
    log.l0, 1111
log_msg
    log.l0, 1112
log_rich_format
    l0_context.CompilationContext, 210
log_stage
    l0_logger, 35
    log.l0, 1112
log_warning
    l0_logger, 35
    log.l0, 1112
LogConfig, 334
    level, 334
    rich, 334
LogLevel
    log.l0, 1109
lookup
    l0_locals.Scope, 406
loop_depth
    ExpressionTypeChecker, 244
loop_flow_capture_stack
    ExpressionTypeChecker, 244
LoopCleanupScopes, 336
    break_scope, 337
    break_scope_id, 337
    continue_scope, 337
    continue_scope_id, 338
    parent, 338
LoopFixedPointFlow, 338
    backedge_states, 339
    capture, 339
    head, 339
LoopFlowCapture, 340
    break_states, 341
    continue_states, 341
    parent, 341
LoopIterationFlow, 341
    backedge_states, 342
    body_flow, 342
    capture, 342
LoopLabels, 343
    break_label, 343
    continue_label, 343
    parent, 343
LPAREN
    l0_lexer.TokenKind, 470
ls_advance
    lexer.l0, 977
ls_at_end
    lexer.l0, 978
ls_clear_pending_error
    lexer.l0, 978
ls_create
    lexer.l0, 978
ls_defer_recoverable_error
    lexer.l0, 978
ls_emit_error
    lexer.l0, 979
ls_free
    lexer.l0, 979
ls_has_errors
    lexer.l0, 979
ls_has_queued_tokens
    lexer.l0, 980
ls_next_token
    lexer.l0, 980
ls_peek
    lexer.l0, 980
ls_peek_next
    lexer.l0, 980
ls_queue_pending_errors
    lexer.l0, 981
ls_queue_pending_recovery
    lexer.l0, 981
ls_queue_terminal_recovery
    lexer.l0, 981
ls_queue_token
    lexer.l0, 982
ls_read_byte_literal
    lexer.l0, 982
ls_read_number
    lexer.l0, 982
ls_read_string_literal
    lexer.l0, 982
ls_read_valid_char_escape
    lexer.l0, 983
ls_skip_invalid_characters
    lexer.l0, 983
ls_skip_whitespace_and_comments
    lexer.l0, 983
ls_take_queued_token
    lexer.l0, 983
LSHIFT
    l0_lexer.TokenKind, 470
LT
    l0_lexer.TokenKind, 470

main
    check_trace_log, 23
    l0c, 59
    l0c.l0, 965
mangle_enum_name
    l0_c_emitter.CEmitter, 196
mangle_function_name

l0_c_emitter.CEmitter, 197
mangle_identifier
l0_c_emitter.CEmitter, 197
mangle_let_name
l0_c_emitter.CEmitter, 197
mangle_struct_name
l0_c_emitter.CEmitter, 198
MATCH
l0_lexer.TokenKind, 470
match_arm_vec_free
ast.l0, 785
match_arms
StmtNode, 434
MatchArm, 346
body, 347
pattern, 347
span, 347
MatchArmScopeEntry, 347
arm, 348
scope, 348
max
integer.l0, 558
mem_kind
_rt_alloc_record, 68
memory.l0
rt_alloc, 638
rt_array_element, 639
rt_calloc, 639
rt_free, 639
rt_memcmp, 640
rt_memcpy, 640
rt_memset, 640
rt_realloc, 641
rt_register_foreign, 641
rt_stderr_write, 641
rt_stdin_read, 642
rt_stdout_write, 642
rt_unregister_foreign, 642
message
Diagnostic, 218
l0_diagnostics.Diagnostic, 220
l0_driver.SourceEncodingError, 423
l0_internal_error.InternalCompilerError, 294
messages
check_trace_log._MessageCollector, 65
min
integer.l0, 558
MINUS
l0_lexer.TokenKind, 470
minute
DateTime, 215
mode
CliOptions, 204
mode_explicit
CliOptions, 205
MODULE
l0_analysis.VarRefResolution, 497
l0_lexer.TokenKind, 470
l0_symbols.SymbolKind, 457
Module, 352
decls, 352
filename, 352
imports, 352
name, 353
span, 353
module
l0_symbols.ModuleEnv, 355
l0_symbols.Symbol, 453
l0_types.EnumType, 232
l0_types.StructType, 452
module_envs
l0_analysis.AnalysisResult, 77
l0_expr_types.ExpressionTypeChecker, 264
l0_signatures.SignatureResolver, 420
module_free
ast.l0, 785
module_name
DriverUnit, 223
EnumInfo, 229
FunctionEnv, 280
l0_diagnostics.Diagnostic, 220
l0_locals.FunctionEnv, 281
l0_resolve.SymbolResolution, 459
l0_resolve.TypeResolution, 488
ModuleEnv, 356
StructInfo, 451
Symbol, 455
Type, 479
MODULE_NOT_IMPORTED
l0_resolve.ResolveErrorKind, 401
l0_resolve.TypeResolveErrorKind, 490
module_path
ExprNode, 266
l0_ast.TypeRef, 485
l0_ast.VariantPattern, 494
l0_ast.VarRef, 496
PatternNode, 396
QualifiedNameResult, 400
TypeRef, 486
module_ptr
ModuleEnv, 356
module_relpath
l0_paths.SourceSearchPaths, 425
ModuleEnv, 355
all, 356
ambiguous_imports, 356
imported, 356
imported_modules, 356
locals, 356
module_name, 356
module_ptr, 356
modules
l0_compilation.CompilationUnit, 212
l0_locals.LocalScopeResolver, 330
MODULO
l0_lexer.TokenKind, 470

monotonic_diff
    time.l0, 622
monotonic_now
    time.l0, 623
monotonic_supported
    time.l0, 623
MonotonicTime, 357
    nsec, 357
    sec, 357
month
    CivilDate, 201
    DateTime, 215
mtime_nsec
    FileInfo, 272
    l0_sys_rt_RtFileInfo, 305
    RtFileInfo, 404
mtime_sec
    FileInfo, 272
    fs.l0, 517
    l0_sys_rt_RtFileInfo, 305
    RtFileInfo, 404

name
    EnumDecl, 226
    EnumInfo, 229
    EnumVariant, 233
    EnumVariantInfo, 235
    FieldDecl, 270
    FuncDecl, 276
    Import, 288
    l0_ast.DropStmt, 224
    l0_ast.EnumDecl, 227
    l0_ast.EnumVariant, 234
    l0_ast.FieldDecl, 271
    l0_ast.FuncDecl, 278
    l0_ast.Import, 289
    l0_ast.LetDecl, 311
    l0_ast.LetStmt, 314
    l0_ast.Module, 351
    l0_ast.Param, 371
    l0_ast.StructDecl, 445
    l0_ast.TypeAliasDecl, 481
    l0_ast.TypeRef, 485
    l0_ast.VariantPattern, 494
    l0_ast.VarRef, 496
    l0_locals.LocalSymbol, 333
    l0_resolve.SymbolResolution, 459
    l0_resolve.TypeResolution, 488
    l0_signatures.EnumVariantInfo, 236
    l0_signatures.StructFieldInfo, 447
    l0_symbols.ModuleEnv, 354
    l0_symbols.Symbol, 454
    l0_types.BuiltinType, 133
    l0_types.EnumType, 232
    l0_types.StructType, 452
    LetDecl, 313
    LocalSymbol, 334
    Module, 353
    Param, 372
    PatternNode, 397
    ScopeVar, 415
    StmtNode, 434
    StructDecl, 446
    StructFieldInfo, 448
    StructInfo, 451
    Symbol, 455
    Type, 479
    TypeAliasDecl, 482
    TypeRef, 486
name_qualifier
    ExprNode, 266
    l0_ast.TypeRef, 485
    l0_ast.VariantPattern, 494
    l0_ast.VarRef, 496
    PatternNode, 397
    QualifiedNameResult, 400
    TypeRef, 486
name_resolver.l0
    nr_build_top_key, 995
    nr_collect_locals, 995
    nr_define_local, 995
    nr_env_count, 995
    nr_env_find, 996
    nr_env_get, 996
    nr_extern_signatures_compatible, 996
    nr_free, 997
    nr_make_env, 997
    nr_map_symbol_get, 997
    nr_module_env_all, 998
    nr_module_env_all_keys, 998
    nr_module_env_free, 998
    nr_module_env_imported, 998
    nr_module_env_imported_keys, 999
    nr_module_env_imports_module, 999
    nr_module_env_is_ambiguous, 999
    nr_module_env_local, 1000
    nr_module_env_local_keys, 1000
    nr_open_imports, 1000
    nr_open_imports_one, 1001
    nr_path_is_same_signature, 1001
    nr_resolve, 1001
    nr_span_diag_error, 1002
    nr_span_diag_warn, 1002
    nr_symbol_free_map, 1002
name_tok
    QualifiedNameResult, 400
NameResolution, 358
    by_name, 358
    envs, 358
names
    AnalysisResult, 74
nanosecond
    DateTime, 215
NE
    l0_lexer.TokenKind, 470
NEW
    l0_lexer.TokenKind, 470

next_stmt_unreachable
Backend, 83
ExpressionTypeChecker, 244
no_line_directives
CliOptions, 205
node
l0_symbols.Symbol, 454
nodes
ExprArena, 241
PatternArena, 396
StmtArena, 431
NOT_A_TYPE
l0_resolve.TypeResolveErrorKind, 490
Note
diag.l0, 1096
nr_build_top_key
name_resolver.l0, 995
nr_collect_locals
name_resolver.l0, 995
nr_define_local
name_resolver.l0, 995
nr_env_count
name_resolver.l0, 995
nr_env_find
name_resolver.l0, 996
nr_env_get
name_resolver.l0, 996
nr_extern_signatures_compatible
name_resolver.l0, 996
nr_free
name_resolver.l0, 997
nr_make_env
name_resolver.l0, 997
nr_map_symbol_get
name_resolver.l0, 997
nr_module_env_all
name_resolver.l0, 998
nr_module_env_all_keys
name_resolver.l0, 998
nr_module_env_free
name_resolver.l0, 998
nr_module_env_imported
name_resolver.l0, 998
nr_module_env_imported_keys
name_resolver.l0, 999
nr_module_env_imports_module
name_resolver.l0, 999
nr_module_env_is_ambiguous
name_resolver.l0, 999
nr_module_env_local
name_resolver.l0, 1000
nr_module_env_local_keys
name_resolver.l0, 1000
nr_open_imports
name_resolver.l0, 1000
nr_open_imports_one
name_resolver.l0, 1001
nr_path_is_same_signature
name_resolver.l0, 1001
nr_resolve
name_resolver.l0, 1001
nr_span_diag_error
name_resolver.l0, 1002
nr_span_diag_warn
name_resolver.l0, 1002
nr_symbol_free_map
name_resolver.l0, 1002
nsec
Duration, 225
l0_sys_rt_RtTimeParts, 306
MonotonicTime, 357
RtTimeParts, 405
WallTime, 499
NULL
l0_lexer.TokenKind, 470

obj
l0_ast.FieldAccessExpr, 269
OCT_CHARS
l0_lexer, 33
op
l0_ast.BinaryOp, 126
l0_ast.UnaryOp, 492
operand
l0_ast.UnaryOp, 492
opt_emitted
CEmitter, 151
opt_wrappers
CEmitter, 151
optional.l0
expect_b, 589
expect_i, 589
expect_s, 590
unwrap_or_b, 590
unwrap_or_i, 590
unwrap_or_s, 591
options
CliParseResult, 207
opts
Backend, 83
OROR
l0_lexer.TokenKind, 471
out
CEmitter, 151
l0_c_emitter.CEmitter, 200
output
CliOptions, 205
owned_vars
l0_scope_context.ScopeContext, 410
ScopeContext, 412
owner
_rt_ptr_site, 71
owner_name
Symbol, 455

PARAM
l0_locals.LocalKind, 326

Param, 371
    name, 372
    type_ref, 372
param_free
    ast.l0, 785
param_vec_free
    ast.l0, 786
params
    FuncDecl, 276
    l0_ast.FuncDecl, 278
    l0_types.FuncType, 283
    Type, 479
parent
    CleanupHeaderGuard, 202
    l0_locals.Scope, 407
    l0_scope_context.ScopeContext, 410
    LoopCleanupScopes, 338
    LoopFlowCapture, 341
    LoopLabels, 343
    path.l0, 1115
    Scope, 408
    ScopeContext, 412
parse_diag_count
    parser.l0, 1004
parse_diag_get
    parser.l0, 1004
parse_dotted_name_rest
    shared.l0, 1019
parse_has_errors
    parser.l0, 1004
parse_lex_error_result
    parser.l0, 1004
parse_module
    l0_parser.Parser, 389
parse_module_source
    parser.l0, 1005
parse_module_tokens
    parser.l0, 1005
parse_module_tokens_owned
    parser.l0, 1005
parse_result
    DriverUnit, 223
parse_result_free
    parser.l0, 1006
parser.l0
    impl_result_to_public, 1003
    parse_diag_count, 1004
    parse_diag_get, 1004
    parse_has_errors, 1004
    parse_lex_error_result, 1004
    parse_module_source, 1005
    parse_module_tokens, 1005
    parse_module_tokens_owned, 1005
    parse_result_free, 1006
ParseResult, 391
    diags, 391
    expr_arena, 391
    pattern_arena, 391
    root_module, 392
    stmt_arena, 392
ParserState, 392
    diags, 393
    emitted_lex_errors, 393
    eof_aborted, 393
    expr_arena, 393
    filename, 393
    index, 393
    last_token, 393
    pattern_arena, 393
    stmt_arena, 393
    tokens, 393
parts
    StringBuffer, 435
path
    DriverUnit, 223
    l0_driver.SourceEncodingError, 423
path.l0
    _path_ext_with_dot, 1113
    _path_find_last_sep, 1113
    basename, 1114
    has_extension, 1114
    has_parent, 1114
    is_absolute, 1114
    is_sep, 1115
    join, 1115
    parent, 1115
    stem, 1116
    up_basename, 1117
    up_find_last_sep, 1117
    up_has_parent, 1117
    up_is_absolute, 1117
    up_is_l0_file, 1118
    up_is_sep, 1118
    up_join, 1118
    up_parent, 1119
    up_stem, 1119
pattern
    l0_ast.MatchArm, 346
    MatchArm, 347
pattern_arena
    FunctionEnv, 280
    ImplParseResult, 287
    ParseResult, 391
    ParserState, 393
pattern_arena_add
    ast.l0, 786
pattern_arena_create
    ast.l0, 786
pattern_arena_free
    ast.l0, 786
pattern_arena_get
    ast.l0, 787
pattern_node_create
    ast.l0, 787
pattern_node_free
    ast.l0, 787

PATTERN_VAR
l0_locals.LocalKind, 326
PatternArena, 395
nodes, 396
PatternId
ast.l0, 792
PatternKind
ast.l0, 779
PatternNode, 396
kind, 396
module_path, 396
name, 397
name_qualifier, 397
span, 397
vars, 397
pending_errors
LexerState, 322
phase
Diagnostic, 218
PIPE
l0_lexer.TokenKind, 471
PLUS
l0_lexer.TokenKind, 471
pointer_depth
l0_ast.TypeRef, 485
TypeRef, 487
pow
integer.l0, 558
prepare_optional_wrappers
l0_c_emitter.CEmitter, 198
prepared_input_free
l0c_lib.l0, 971
prepared_input_free_opt
l0c_lib.l0, 973
PreparedInput, 399
entry_module, 399
search_paths, 399
present
unit.l0, 624
prev_ends_expression
LexerState, 322
print_b
io.l0, 565
print_bb
io.l0, 566
print_bi
io.l0, 566
print_bs
io.l0, 566
print_diagnostic_list
l0c, 59
print_diagnostic_with_snippet
l0c, 60
print_diagnostics
l0c, 60
print_i
io.l0, 566
print_ib
io.l0, 567
print_ii
io.l0, 567
print_is
io.l0, 567
print_s
io.l0, 567
print_sb
io.l0, 568
print_si
io.l0, 568
print_ss
io.l0, 568
printl
io.l0, 568
printl_b
io.l0, 569
printl_bb
io.l0, 569
printl_bi
io.l0, 569
printl_bs
io.l0, 569
printl_i
io.l0, 570
printl_ib
io.l0, 570
printl_ii
io.l0, 570
printl_is
io.l0, 570
printl_s
io.l0, 571
printl_sb
io.l0, 571
printl_si
io.l0, 571
printl_ss
io.l0, 571
program_args
CliOptions, 205
project_roots
CliOptions, 205
l0_paths.SourceSearchPaths, 425
SourceSearchPaths, 426
ps_advance
shared.l0, 1020
ps_at_case_arm_start
stmt.l0, 1037
ps_at_end
shared.l0, 1020
ps_at_stmt_start
shared.l0, 1020
ps_check
shared.l0, 1020
ps_check_byte
shared.l0, 1021
ps_check_ident

shared.l0, 1021
ps_check_int
shared.l0, 1021
ps_check_reserved_binary_op
shared.l0, 1022
ps_check_string
shared.l0, 1022
ps_create
shared.l0, 1022
ps_destroy
shared.l0, 1023
ps_emit_error
shared.l0, 1023
ps_emit_remaining_lexer_errors
shared.l0, 1023
ps_expect
shared.l0, 1023
ps_expect_ident
shared.l0, 1024
ps_expect_variable_name
shared.l0, 1024
ps_extend_span
shared.l0, 1024
ps_finish_function
decl.l0, 1008
ps_has_error
shared.l0, 1025
ps_is_builtin_type_name
expr.l0, 1012
ps_is_top_level_start
shared.l0, 1025
ps_is_unambiguous_type_start
expr.l0, 1013
ps_last
shared.l0, 1025
ps_lex_error_result
decl.l0, 1008
ps_logical_token_at_cursor
shared.l0, 1026
ps_make_empty_block_stmt
decl.l0, 1008
ps_match
shared.l0, 1026
ps_match_any2
shared.l0, 1026
ps_match_any3
shared.l0, 1027
ps_match_any4
shared.l0, 1027
ps_match_byte
shared.l0, 1027
ps_match_ident
shared.l0, 1028
ps_match_int
shared.l0, 1028
ps_match_string
shared.l0, 1028
ps_parse_add_expr
expr.l0, 1013
ps_parse_and_expr
expr.l0, 1013
ps_parse_block
stmt.l0, 1037
ps_parse_break_stmt
stmt.l0, 1037
ps_parse_call_argument
expr.l0, 1014
ps_parse_case_literal
stmt.l0, 1038
ps_parse_case_stmt
stmt.l0, 1038
ps_parse_cast_expr
expr.l0, 1014
ps_parse_continue_stmt
stmt.l0, 1038
ps_parse_drop_stmt
stmt.l0, 1039
ps_parse_enum
decl.l0, 1009
ps_parse_equality_expr
expr.l0, 1014
ps_parse_expr
expr.l0, 1014
ps_parse_extern_func
decl.l0, 1009
ps_parse_for_stmt
stmt.l0, 1039
ps_parse_for_update_stmt
stmt.l0, 1039
ps_parse_function
decl.l0, 1009
ps_parse_if_stmt
stmt.l0, 1039
ps_parse_let_stmt
stmt.l0, 1040
ps_parse_match_stmt
stmt.l0, 1040
ps_parse_module
decl.l0, 1010
ps_parse_mul_expr
expr.l0, 1015
ps_parse_or_expr
expr.l0, 1015
ps_parse_pattern
stmt.l0, 1040
ps_parse_postfix_expr
expr.l0, 1015
ps_parse_primary_expr
expr.l0, 1016
ps_parse_rel_expr
expr.l0, 1016
ps_parse_return_stmt
stmt.l0, 1041
ps_parse_simple_stmt
stmt.l0, 1041
ps_parse_stmt

stmt.l0, 1041
ps_parse_struct
decl.l0, 1010
ps_parse_top_level_decl
decl.l0, 1010
ps_parse_top_level_let
decl.l0, 1011
ps_parse_type
shared.l0, 1029
ps_parse_type_alias
decl.l0, 1011
ps_parse_unary_expr
expr.l0, 1016
ps_parse_while_stmt
stmt.l0, 1042
ps_parse_with_item
stmt.l0, 1042
ps_parse_with_stmt
stmt.l0, 1042
ps_peek
shared.l0, 1029
ps_peek_logical_ahead
shared.l0, 1029
ps_skip_lexer_errors
shared.l0, 1030
ps_span_start
shared.l0, 1030
ps_sync_case_invalid_arm
stmt.l0, 1042
ps_sync_stmt
shared.l0, 1030
ps_sync_top_level
shared.l0, 1030
ps_tokens_result
decl.l0, 1011
ps_try_parse_qualified_name
shared.l0, 1031
PT_VARIANT
ast.l0, 779
PT_WILDCARD
ast.l0, 779
ptr_vec_create
ast.l0, 787
ptr_vec_get
ast.l0, 788
ptr_vec_push
ast.l0, 788

q_next
_rt_alloc_record, 68
qn_result_free_all
shared.l0, 1031
QualifiedNameResult, 400
module_path, 400
name_qualifier, 400
name_tok, 400
QUESTION
l0_lexer.TokenKind, 471
queued_index
LexerState, 322
queued_tokens
LexerState, 322

rand.l0
rand_bool, 592
rand_dice, 592
rand_int, 592
rand_int_range, 592
rand_seed, 593
rand_bool
rand.l0, 592
rand_dice
rand.l0, 592
rand_int
rand.l0, 592
rand_int_range
rand.l0, 592
rand_seed
rand.l0, 593
RBRACE
l0_lexer.TokenKind, 471
RBRACKET
l0_lexer.TokenKind, 471
read_char
io.l0, 572
read_char_or_eof
io.l0, 572
read_file
fs.l0, 517
read_line
io.l0, 572
read_only
_rt_alloc_record, 68
read_stdin_some
io.l0, 572
recovery
l0_lexer.Token, 461
refcount
_l0_h_string, 64
remove_owned
l0_scope_context.ScopeContext, 410
repeat_s
text.l0, 613
replace_s
text.l0, 614
reserved
_rt_alloc_record_cold, 69
resolve
l0_locals.LocalScopeResolver, 330
l0_name_resolver.NameResolver, 361
l0_paths.SourceSearchPaths, 425
l0_signatures.SignatureResolver, 419
resolve_symbol
l0_resolve, 37
resolve_type_ref
l0_resolve, 38
resolved_type
Symbol, 455

restore_active_line_directive
l0_c_emitter.CEmitter, 198
result
Backend, 83
ExpressionTypeChecker, 244
l0_types.FuncType, 283
Type, 479
retained_path
l0c._TemporarySourceWriteError, 72
RETURN
l0_lexer.TokenKind, 471
return_type
FuncDecl, 276
l0_ast.FuncDecl, 278
RETURNS
l0_expr_types.StmtFlow, 432
reverse_s
text.l0, 614
rich
LogConfig, 334
rich_log
CliParseResult, 207
right
l0_ast.BinaryOp, 126
root
CompilerWorkspace, 213
root_module
ImplParseResult, 287
ParseResult, 392
root_scope
FunctionEnv, 280
l0_locals.FunctionEnv, 282
RPAREN
l0_lexer.TokenKind, 471
RSHIFT
l0_lexer.TokenKind, 471
rt.l0
rt_abort, 645
rt_delete_file, 645
rt_errno, 645
rt_exit, 645
rt_file_info, 645
rt_flush_stderr, 646
rt_flush_stdout, 646
rt_get_argc, 646
rt_get_argv, 646
rt_get_env_var, 647
rt_get_pid, 647
rt_print, 647
rt_print_bool, 647
rt_print_bool_stderr, 648
rt_print_int, 648
rt_print_int_stderr, 648
rt_print_stderr, 648
rt_println, 648
rt_println_stderr, 649
rt_rand, 649
rt_read_char, 649
rt_read_file_all, 649
rt_read_line, 650
rt_srand, 650
rt_string_bytes_ptr, 650
rt_string_compare, 650
rt_string_concat, 651
rt_string_equals, 651
rt_string_from_byte, 651
rt_string_from_byte_array, 652
rt_string_get, 652
rt_string_release, 652
rt_string_retain, 653
rt_string_slice, 653
rt_strlen, 653
rt_system, 653
rt_time_local_is_dst, 654
rt_time_local_offset_sec, 654
rt_time_monotonic, 654
rt_time_monotonic_supported, 655
rt_time_unix, 655
rt_write_file_all, 655
rt_abort
l0_runtime.h, 696
rt.l0, 645
rt_alloc
l0_runtime.h, 696
memory.l0, 638
rt_array_element
l0_runtime.h, 697
memory.l0, 639
rt_calloc
l0_runtime.h, 697
memory.l0, 639
rt_check_basic
CodegenOptions, 208
l0_context.CompilationContext, 210
rt_delete_file
l0_runtime.h, 698
rt.l0, 645
rt_errno
l0_runtime.h, 698
rt.l0, 645
rt_exit
l0_runtime.h, 698
rt.l0, 645
rt_file_info
l0_runtime.h, 699
rt.l0, 645
rt_flush_stderr
l0_runtime.h, 699
rt.l0, 646
rt_flush_stdout
l0_runtime.h, 699
rt.l0, 646
rt_free
l0_runtime.h, 699
memory.l0, 639
rt_get_argc

l0_runtime.h, 700
rt.l0, 646
rt_get_argv
l0_runtime.h, 700
rt.l0, 646
rt_get_env_var
l0_runtime.h, 700
rt.l0, 647
rt_get_pid
l0_runtime.h, 701
rt.l0, 647
rt_hash_bool
hash.l0, 634
l0_runtime.h, 701
rt_hash_byte
hash.l0, 634
l0_runtime.h, 701
rt_hash_data
hash.l0, 635
l0_runtime.h, 702
rt_hash_int
hash.l0, 635
l0_runtime.h, 702
rt_hash_opt_bool
hash.l0, 635
l0_runtime.h, 702
rt_hash_opt_byte
hash.l0, 636
l0_runtime.h, 703
rt_hash_opt_int
hash.l0, 636
l0_runtime.h, 703
rt_hash_opt_ptr
hash.l0, 636
l0_runtime.h, 703
rt_hash_opt_string
hash.l0, 637
l0_runtime.h, 704
rt_hash_ptr
hash.l0, 637
l0_runtime.h, 704
rt_hash_string
hash.l0, 637
l0_runtime.h, 705
rt_memcmp
l0_runtime.h, 705
memory.l0, 640
rt_memcpy
l0_runtime.h, 705
memory.l0, 640
rt_memset
l0_runtime.h, 706
memory.l0, 640
rt_print
l0_runtime.h, 706
rt.l0, 647
rt_print_bool
l0_runtime.h, 707
rt.l0, 647
rt_print_bool_stderr
l0_runtime.h, 707
rt.l0, 648
rt_print_int
l0_runtime.h, 707
rt.l0, 648
rt_print_int_stderr
l0_runtime.h, 707
rt.l0, 648
rt_print_stderr
l0_runtime.h, 708
rt.l0, 648
rt_println
l0_runtime.h, 708
rt.l0, 648
rt_println_stderr
l0_runtime.h, 708
rt.l0, 649
rt_rand
l0_runtime.h, 708
rt.l0, 649
rt_read_char
l0_runtime.h, 709
rt.l0, 649
rt_read_file_all
l0_runtime.h, 709
rt.l0, 649
rt_read_line
l0_runtime.h, 709
rt.l0, 650
rt_realloc
l0_runtime.h, 710
memory.l0, 641
rt_register_foreign
l0_runtime.h, 710
memory.l0, 641
rt_srand
l0_runtime.h, 710
rt.l0, 650
rt_stderr_write
l0_runtime.h, 711
memory.l0, 641
rt_stdin_read
l0_runtime.h, 711
memory.l0, 642
rt_stdout_write
l0_runtime.h, 711
memory.l0, 642
rt_string_bytes_ptr
l0_runtime.h, 712
rt.l0, 650
rt_string_compare
l0_runtime.h, 712
rt.l0, 650
rt_string_concat
l0_runtime.h, 713
rt.l0, 651

rt_string_equals
  l0_runtime.h, 713
  rt.l0, 651
rt_string_from_byte
  l0_runtime.h, 713
  rt.l0, 651
rt_string_from_byte_array
  l0_runtime.h, 714
  rt.l0, 652
rt_string_get
  l0_runtime.h, 714
  rt.l0, 652
rt_string_release
  l0_runtime.h, 715
  rt.l0, 652
rt_string_retain
  l0_runtime.h, 715
  rt.l0, 653
rt_string_slice
  l0_runtime.h, 715
  rt.l0, 653
rt_strlen
  l0_runtime.h, 716
  rt.l0, 653
rt_system
  l0_runtime.h, 716
  rt.l0, 653
rt_time_local_is_dst
  l0_runtime.h, 716
  rt.l0, 654
rt_time_local_offset_sec
  l0_runtime.h, 717
  rt.l0, 654
rt_time_monotonic
  l0_runtime.h, 717
  rt.l0, 654
rt_time_monotonic_supported
  l0_runtime.h, 717
  rt.l0, 655
rt_time_unix
  l0_runtime.h, 718
  rt.l0, 655
rt_unchecked
  CodegenOptions, 208
  l0_context.CompilationContext, 210
rt_unregister_foreign
  l0_runtime.h, 718
  memory.l0, 642
rt_write_file_all
  l0_runtime.h, 718
  rt.l0, 655
RtFileInfo, 404
  exists, 404
  is_dir, 404
  is_file, 404
  mtime_nsec, 404
  mtime_sec, 404
  size, 404
RtTimeParts, 405
  nsec, 405
  sec, 405
run_with_argv
  l0c_lib.l0, 973
runtime_include
  CliOptions, 205
runtime_lib
  CliOptions, 205

s_str
  l0_string, 304
sb_append
  text.l0, 614
sb_append_byte
  text.l0, 615
sb_append_int
  text.l0, 615
sb_create
  text.l0, 615
sb_free
  text.l0, 615
sb_size
  text.l0, 616
sb_to_string
  text.l0, 616
sc_add_declared
  scope_context.l0, 1044
sc_add_owned
  scope_context.l0, 1044
sc_create
  scope_context.l0, 1044
sc_free
  scope_context.l0, 1044
sc_has_inline_cleanup
  scope_context.l0, 1045
sc_inline_cleanup_free_opt
  scope_context.l0, 1045
sc_lookup_declared_type
  scope_context.l0, 1045
sc_push_inline_cleanup
  scope_context.l0, 1045
sc_remove_owned
  scope_context.l0, 1046
sc_var_free
  scope_context.l0, 1046
Scope, 407
  parent, 408
  symbols, 408
scope
  MatchArmScopeEntry, 348
  ScopeSlot, 414
scope_context.l0
  sc_add_declared, 1044
  sc_add_owned, 1044
  sc_create, 1044
  sc_free, 1044
  sc_has_inline_cleanup, 1045
  sc_inline_cleanup_free_opt, 1045

sc_lookup_declared_type, 1045
sc_push_inline_cleanup, 1045
sc_remove_owned, 1046
sc_var_free, 1046
scope_counter
Backend, 83
scope_id
ScopeContext, 412
scope_stack
Backend, 83
ScopeContext, 411
declared_vars, 412
owned_vars, 412
parent, 412
scope_id, 412
with_cleanup_block, 412
with_cleanup_in_progress, 412
with_cleanup_inline, 412
scopes
FunctionEnv, 280
ScopeSlot, 413
scope, 414
ScopeVar, 414
name, 415
var_type, 415
se_append_octal_digit
string_escape.l0, 1069
se_append_octal_escape
string_escape.l0, 1070
se_encode_c_string_bytes
string_escape.l0, 1070
se_is_direct_c_byte
string_escape.l0, 1070
se_is_trigraph_suffix
string_escape.l0, 1070
search_paths
BuildPreparedInput, 131
l0_driver.L0Driver, 310
PreparedInput, 399
sec
Duration, 225
l0_sys_rt_RtTimeParts, 306
MonotonicTime, 357
RtTimeParts, 405
WallTime, 499
second
DateTime, 215
sem_context.l0
analysis_diag_count, 1048
analysis_diag_get, 1048
analysis_expr_key, 1048
analysis_get_expr_type, 1049
analysis_get_intrinsic_target, 1049
analysis_get_var_ref_resolution, 1049
analysis_has_arc_data, 1050
analysis_has_errors, 1050
analysis_is_arc_type, 1050
analysis_locals, 1050
analysis_name_resolution, 1051
analysis_owned_type_map_put, 1051
analysis_result_free, 1051
analysis_signature_tables, 1053
analysis_source_names, 1053
analysis_source_texts, 1053
VAR_RES_LOCAL, 1048
VAR_RES_MODULE, 1048
VarRefResolution, 1047
SEMI
l0_lexer.TokenKind, 471
set_analysis
l0_c_emitter.CEmitter, 198
severity
Diagnostic, 218
shared.l0
parse_dotted_name_rest, 1019
ps_advance, 1020
ps_at_end, 1020
ps_at_stmt_start, 1020
ps_check, 1020
ps_check_byte, 1021
ps_check_ident, 1021
ps_check_int, 1021
ps_check_reserved_binary_op, 1022
ps_check_string, 1022
ps_create, 1022
ps_destroy, 1023
ps_emit_error, 1023
ps_emit_remaining_lexer_errors, 1023
ps_expect, 1023
ps_expect_ident, 1024
ps_expect_variable_name, 1024
ps_extend_span, 1024
ps_has_error, 1025
ps_is_top_level_start, 1025
ps_last, 1025
ps_logical_token_at_cursor, 1026
ps_match, 1026
ps_match_any2, 1026
ps_match_any3, 1027
ps_match_any4, 1027
ps_match_byte, 1027
ps_match_ident, 1028
ps_match_int, 1028
ps_match_string, 1028
ps_parse_type, 1029
ps_peek, 1029
ps_peek_logical_ahead, 1029
ps_skip_lexer_errors, 1030
ps_span_start, 1030
ps_sync_stmt, 1030
ps_sync_top_level, 1030
ps_try_parse_qualified_name, 1031
qn_result_free_all, 1031
tok_is, 1031
tok_is_byte, 1031
tok_is_future_extension, 1032

tok_is_ident, 1032
tok_is_int, 1032
tok_is_string, 1033
token_byte_text, 1033
token_byte_value, 1033
token_ident_text, 1034
token_int_text, 1034
token_int_value, 1034
token_string_text, 1034
token_string_value, 1035
token_text_for_width, 1035
token_width, 1035
sig_cycle_visit
signatures.l0, 1055
sig_detect_value_type_cycles
signatures.l0, 1055
sig_diag_error
signatures.l0, 1056
sig_enum_variant_vec_temp_free
signatures.l0, 1056
sig_extract_dependencies
signatures.l0, 1056
sig_infer_literal_type
signatures.l0, 1056
sig_key
signatures.l0, 1057
sig_lookup_enum_info
signatures.l0, 1057
sig_lookup_func_type
signatures.l0, 1057
sig_lookup_let_type
signatures.l0, 1058
sig_lookup_struct_info
signatures.l0, 1058
sig_make_tables
signatures.l0, 1059
sig_resolve
signatures.l0, 1059
sig_resolve_enum
signatures.l0, 1059
sig_resolve_func
signatures.l0, 1059
sig_resolve_let
signatures.l0, 1060
sig_resolve_struct
signatures.l0, 1060
sig_set_free_opt
signatures.l0, 1060
sig_struct_field_vec_temp_free
signatures.l0, 1061
sig_symbol_is_zero_field_enum_variant
signatures.l0, 1061
sig_tables_free
types.l0, 1089
sig_top_decl_name
signatures.l0, 1061
sig_top_decl_span
signatures.l0, 1062
sig_type_vec_free
signatures.l0, 1062
sig_type_vec_free_opt
signatures.l0, 1062
sign
integer.l0, 559
signatures
AnalysisResult, 74
signatures.l0
sig_cycle_visit, 1055
sig_detect_value_type_cycles, 1055
sig_diag_error, 1056
sig_enum_variant_vec_temp_free, 1056
sig_extract_dependencies, 1056
sig_infer_literal_type, 1056
sig_key, 1057
sig_lookup_enum_info, 1057
sig_lookup_func_type, 1057
sig_lookup_let_type, 1058
sig_lookup_struct_info, 1058
sig_make_tables, 1059
sig_resolve, 1059
sig_resolve_enum, 1059
sig_resolve_func, 1059
sig_resolve_let, 1060
sig_resolve_struct, 1060
sig_set_free_opt, 1060
sig_struct_field_vec_temp_free, 1061
sig_symbol_is_zero_field_enum_variant, 1061
sig_top_decl_name, 1061
sig_top_decl_span, 1062
sig_type_vec_free, 1062
sig_type_vec_free_opt, 1062
SignatureTables, 420
enum_infos, 421
func_types, 421
let_types, 421
struct_infos, 421
SILENT
l0_context.LogLevel, 336
sim_capacity
hashmap.l0, 530
sim_clear
hashmap.l0, 530
sim_create
hashmap.l0, 530
sim_create_with_capacity
hashmap.l0, 530
sim_free
hashmap.l0, 531
sim_get
hashmap.l0, 531
sim_has
hashmap.l0, 531
sim_keys
hashmap.l0, 531
sim_put
hashmap.l0, 532

sim_remove
hashmap.l0, 532
sim_size
hashmap.l0, 532
sim_slot_key
hashmap.l0, 533
sim_slot_occupied
hashmap.l0, 533
sim_slot_value
hashmap.l0, 533
siphash13
dea_siphash.h, 657
siphash13_tag8_bf
dea_siphash.h, 657
siphash13_tag8_bytes
dea_siphash.h, 658
siphash24
dea_siphash.h, 658
siphash24_tag8_bf
dea_siphash.h, 659
siphash24_tag8_bytes
dea_siphash.h, 659
size
_rt_alloc_record, 68
FileInfo, 273
l0_sys_rt_RtFileInfo, 305
RtFileInfo, 404
StringBuffer, 435
SLASH
l0_lexer.TokenKind, 472
slice_s
string.l0, 600
source
l0_lexer.Lexer, 320
LexerState, 323
source_names
DriverState, 222
source_paths.l0
sp_add_project_root, 1064
sp_add_system_root, 1064
sp_append_env_sys_roots, 1064
sp_apply_default_project_roots, 1064
sp_apply_default_sys_roots, 1064
sp_apply_default_sys_roots_env, 1065
sp_build_from_roots, 1065
sp_create, 1065
sp_format_root_list, 1065
sp_free, 1066
sp_is_valid_ident_component, 1066
sp_is_valid_module_name, 1066
sp_module_relpath, 1067
sp_normalize_entry_target, 1067
sp_project_root_count, 1067
sp_project_root_get, 1068
sp_resolve, 1068
sp_system_root_count, 1068
sp_system_root_get, 1069
source_texts
DriverState, 222
SourceSearchPaths, 426
project_roots, 426
system_roots, 426
sp_add_project_root
source_paths.l0, 1064
sp_add_system_root
source_paths.l0, 1064
sp_append_env_sys_roots
source_paths.l0, 1064
sp_apply_default_project_roots
source_paths.l0, 1064
sp_apply_default_sys_roots
source_paths.l0, 1064
sp_apply_default_sys_roots_env
source_paths.l0, 1065
sp_build_from_roots
source_paths.l0, 1065
sp_create
source_paths.l0, 1065
sp_format_root_list
source_paths.l0, 1065
sp_free
source_paths.l0, 1066
sp_is_valid_ident_component
source_paths.l0, 1066
sp_is_valid_module_name
source_paths.l0, 1066
sp_module_relpath
source_paths.l0, 1067
sp_normalize_entry_target
source_paths.l0, 1067
sp_project_root_count
source_paths.l0, 1067
sp_project_root_get
source_paths.l0, 1068
sp_resolve
source_paths.l0, 1068
sp_system_root_count
source_paths.l0, 1068
sp_system_root_get
source_paths.l0, 1069
Span, 427
end_column, 428
end_line, 428
start_column, 428
start_line, 428
span
CaseArm, 138
CaseElse, 140
EnumDecl, 226
EnumInfo, 229
ExprNode, 266
FuncDecl, 276
Import, 288
l0_ast.Node, 365
l0_internal_error.ICELocation, 284
LetDecl, 313

MatchArm, 347
Module, 353
PatternNode, 397
StmtNode, 434
StructDecl, 446
StructInfo, 451
Symbol, 455
TypeAliasDecl, 482
TypeRef, 487
WithItem, 505
span_here
ast.l0, 788
split_s
text.l0, 616
spm_capacity
hashmap.l0, 534
spm_clear
hashmap.l0, 534
spm_create
hashmap.l0, 534
spm_create_with_capacity
hashmap.l0, 534
spm_free
hashmap.l0, 535
spm_get
hashmap.l0, 535
spm_has
hashmap.l0, 535
spm_keys
hashmap.l0, 536
spm_put
hashmap.l0, 536
spm_remove
hashmap.l0, 536
spm_size
hashmap.l0, 537
spm_slot_key
hashmap.l0, 537
spm_slot_occupied
hashmap.l0, 537
spm_slot_value
hashmap.l0, 538
ss_add
hashset.l0, 546
ss_capacity
hashset.l0, 546
ss_clear
hashset.l0, 547
ss_create
hashset.l0, 547
ss_create_with_capacity
hashset.l0, 547
ss_free
hashset.l0, 547
ss_has
hashset.l0, 548
ss_remove
hashset.l0, 548
ss_size
hashset.l0, 548
ss_slot_key
hashset.l0, 549
ss_slot_occupied
hashset.l0, 549
ss_to_vector
hashset.l0, 549
sslm_contains
linear_map.l0, 586
sslm_create
linear_map.l0, 586
sslm_free
linear_map.l0, 587
sslm_get
linear_map.l0, 587
sslm_key_at
linear_map.l0, 587
sslm_len
linear_map.l0, 587
sslm_remove
linear_map.l0, 588
sslm_set
linear_map.l0, 588
sslm_value_at
linear_map.l0, 588
ST_ASSIGN
ast.l0, 779
ST_BLOCK
ast.l0, 779
ST_BREAK
ast.l0, 779
ST_CASE
ast.l0, 779
ST_CONTINUE
ast.l0, 779
ST_DROP
ast.l0, 779
ST_EXPR
ast.l0, 779
ST_FOR
ast.l0, 779
ST_IF
ast.l0, 779
ST_LET
ast.l0, 779
ST_MATCH
ast.l0, 779
ST_RETURN
ast.l0, 779
ST_WHILE
ast.l0, 779
ST_WITH
ast.l0, 779
STAR
l0_lexer.TokenKind, 472
start_column
l0_ast.Span, 427

Span, 428
start_line
l0_ast.Span, 427
Span, 428
starts_with_s
string.l0, 601
stat
fs.l0, 518
state
_rt_alloc_record, 68
states
IntSet, 297
StringIntMap, 437
StringPtrMap, 440
StringSet, 442
stem
path.l0, 1116
stmt.l0
ps_at_case_arm_start, 1037
ps_parse_block, 1037
ps_parse_break_stmt, 1037
ps_parse_case_literal, 1038
ps_parse_case_stmt, 1038
ps_parse_continue_stmt, 1038
ps_parse_drop_stmt, 1039
ps_parse_for_stmt, 1039
ps_parse_for_update_stmt, 1039
ps_parse_if_stmt, 1039
ps_parse_let_stmt, 1040
ps_parse_match_stmt, 1040
ps_parse_pattern, 1040
ps_parse_return_stmt, 1041
ps_parse_simple_stmt, 1041
ps_parse_stmt, 1041
ps_parse_while_stmt, 1042
ps_parse_with_item, 1042
ps_parse_with_stmt, 1042
ps_sync_case_invalid_arm, 1042
stmt_a
StmtNode, 434
stmt_arena
FunctionEnv, 280
ImplParseResult, 287
ParseResult, 392
ParserState, 393
stmt_arena_add
ast.l0, 789
stmt_arena_create
ast.l0, 789
stmt_arena_free
ast.l0, 789
stmt_arena_get
ast.l0, 790
stmt_b
StmtNode, 435
stmt_c
StmtNode, 435
stmt_node_create
ast.l0, 790
stmt_node_free
ast.l0, 790
StmtArena, 430
nodes, 431
StmtFlowKind
expr_types.l0, 938
StmtId
ast.l0, 792
StmtKind
ast.l0, 779
StmtNode, 433
case_arms, 434
case_else, 434
expr_a, 434
expr_b, 434
ids, 434
kind, 434
match_arms, 434
name, 434
span, 434
stmt_a, 434
stmt_b, 435
stmt_c, 435
type_ref, 435
with_items, 435
stmts
l0_ast.Block, 128
STOPS
l0_expr_types.StmtFlow, 433
storage
ByteArray, 134
STRING
l0_lexer.TokenKind, 472
string.l0
byte_to_s, 594
bytes_to_s, 595
char_at_s, 595
cmp_s, 595
concat_s, 596
contains_s, 596
digit_to_char, 596
ends_with_s, 597
eq_s, 597
find_from_s, 597
find_last_s, 598
find_s, 598
is_alnum, 598
is_alpha, 599
is_digit, 599
is_digit_base, 599
is_empty_s, 599
is_space, 600
len_s, 600
slice_s, 600
starts_with_s, 601
to_digit, 601
to_digit_base, 601

to_lower, 602
to_upper, 602
trim_s, 602
string_escape.l0
se_append_octal_digit, 1069
se_append_octal_escape, 1070
se_encode_c_string_bytes, 1070
se_is_direct_c_byte, 1070
se_is_trigraph_suffix, 1070
string_to_bool
text.l0, 617
string_to_byte
text.l0, 617
string_to_byte_base
text.l0, 617
string_to_int
text.l0, 617
string_to_int_base
text.l0, 618
string_type
l0_expr_types.ExpressionTypeChecker, 264
StringBuffer, 435
parts, 435
size, 435
StringIntMap, 436
capacity, 437
count, 437
hashes, 437
keys, 437
states, 437
tomb_count, 437
values, 437
StringPtrMap, 439
capacity, 440
count, 440
hashes, 440
keys, 440
states, 440
tomb_count, 440
values, 440
strings.l0
us_is_utf8_continuation, 1120
us_is_valid_utf8, 1120
us_split_char, 1121
us_split_chars2_non_empty, 1121
us_split_is_space_non_empty, 1121
us_strip_utf8_bom, 1122
us_sv_clone, 1122
StringSet, 441
capacity, 441
count, 441
hashes, 442
keys, 442
states, 442
tomb_count, 442
StringStringLinearMap, 442
base, 443
StringStringLinearMapEntry, 443
key, 443
value, 443
StringVector
vector.l0, 633
STRUCT
l0_lexer.TokenKind, 472
l0_symbols.SymbolKind, 457
struct_decl_free
ast.l0, 790
struct_info_free
types.l0, 1089
struct_infos
l0_analysis.AnalysisResult, 77
l0_expr_types.ExpressionTypeChecker, 264
SignatureTables, 421
struct_type
l0_signatures.StructInfo, 449
StructInfo, 451
StructDecl, 445
fields, 446
name, 446
span, 446
StructFieldInfo, 447
field_type, 448
name, 448
StructInfo, 450
fields, 450
filename, 450
key, 451
module_name, 451
name, 451
span, 451
struct_type, 451
suppress_diagnostics
ExpressionTypeChecker, 244
suppress_liveness_diagnostics
ExpressionTypeChecker, 244
sv_capacity
vector.l0, 627
sv_clear
vector.l0, 627
sv_create
vector.l0, 627
sv_free
vector.l0, 627
sv_free_opt
ast.l0, 791
sv_get
vector.l0, 628
sv_push
vector.l0, 628
sv_size
vector.l0, 628
sv_sort
vector.l0, 628
SYM_ENUM
symbols.l0, 1072
SYM_ENUM_VARIANT

symbols.l0, 1072
SYM_FUNC
symbols.l0, 1072
SYM_LET
symbols.l0, 1072
SYM_STRUCT
symbols.l0, 1072
SYM_TYPE_ALIAS
symbols.l0, 1072
Symbol, 454
decl_ptr, 455
kind, 455
module_name, 455
name, 455
owner_name, 455
resolved_type, 455
span, 455
symbol
l0_resolve.SymbolResolution, 459
l0_resolve.TypeResolution, 488
symbol_create
symbols.l0, 1072
symbol_free
symbols.l0, 1072
symbol_kind_name
symbols.l0, 1072
symbol_set_type_clone
symbols.l0, 1073
SymbolKind
symbols.l0, 1071
symbols
l0_locals.Scope, 407
Scope, 408
symbols.l0
SYM_ENUM, 1072
SYM_ENUM_VARIANT, 1072
SYM_FUNC, 1072
SYM_LET, 1072
SYM_STRUCT, 1072
SYM_TYPE_ALIAS, 1072
symbol_create, 1072
symbol_free, 1072
symbol_kind_name, 1072
symbol_set_type_clone, 1073
SymbolKind, 1071
sys_roots
CliOptions, 205
system
system.l0, 605
system.l0
abort, 603
argc, 603
argv, 604
env_get, 604
errno, 604
exit, 604
get_pid, 605
system, 605
system_roots
l0_paths.SourceSearchPaths, 425
SourceSearchPaths, 426

target
l0_ast.AssignStmt, 80
l0_ast.TypeAliasDecl, 481
TypeAliasDecl, 482
target_type
l0_ast.CastExpr, 145
TD_ENUM
ast.l0, 779
TD_FUNC
ast.l0, 779
TD_LET
ast.l0, 779
TD_STRUCT
ast.l0, 779
TD_TYPE_ALIAS
ast.l0, 779
text
l0_lexer.Token, 461
text.l0
_line_strip_trailing_cr, 607
bool_to_string, 607
byte_to_string, 608
byte_to_string_base, 608
cb_append, 608
cb_append_int, 609
cb_append_s, 609
cb_append_slice, 609
cb_capacity, 609
cb_clear, 610
cb_create, 610
cb_free, 610
cb_reserve, 610
cb_reverse, 611
cb_size, 611
cb_to_string, 611
int_to_bin_string, 611
int_to_hex_string, 612
int_to_string, 612
int_to_string_base, 612
join_s, 613
lines_s, 613
repeat_s, 613
replace_s, 614
reverse_s, 614
sb_append, 614
sb_append_byte, 615
sb_append_int, 615
sb_create, 615
sb_free, 615
sb_size, 616
sb_to_string, 616
split_s, 616
string_to_bool, 617
string_to_byte, 617
string_to_byte_base, 617

string_to_int, 617
string_to_int_base, 618
to_lower_s, 618
to_upper_s, 619
text_value
ExprNode, 266
then_stmt
l0_ast.IfStmt, 286
TILDE
l0_lexer.TokenKind, 472
time.l0
civil_from_unix_days, 620
datetime_from_unix_sec, 620
day_of_year, 621
days_in_month, 621
is_leap_year, 621
is_valid_nsec, 622
local_now_datetime, 622
monotonic_diff, 622
monotonic_now, 623
monotonic_supported, 623
utc_now_datetime, 623
wall_now, 623
wall_to_local_datetime, 623
wall_to_utc_datetime, 624
tmp_counter
CEmitter, 151
to_digit
string.l0, 601
to_digit_base
string.l0, 601
to_lower
string.l0, 602
to_lower_s
text.l0, 618
to_string
l0_c_emitter.CCodeBuilder, 149
to_upper
string.l0, 602
to_upper_s
text.l0, 619
tok_is
shared.l0, 1031
tok_is_byte
shared.l0, 1031
tok_is_future_extension
shared.l0, 1032
tok_is_ident
shared.l0, 1032
tok_is_int
shared.l0, 1032
tok_is_string
shared.l0, 1033
Token, 461
column, 462
index, 462
line, 462
token_type, 462
token_byte_text
shared.l0, 1033
token_byte_value
shared.l0, 1033
token_ident_text
shared.l0, 1034
token_int_text
shared.l0, 1034
token_int_value
shared.l0, 1034
token_len
l0_parser, 36
token_logical_predecessor_ends_expression
tokens.l0, 1078
token_recovered
tokens.l0, 1079
token_recovery_is_some
tokens.l0, 1079
token_recovery_release_payload
tokens.l0, 1079
token_recovery_to_string
tokens.l0, 1080
token_release_payload
tokens.l0, 1080
token_string_text
shared.l0, 1034
token_string_value
shared.l0, 1035
token_text_for_width
shared.l0, 1035
token_to_string
tokens.l0, 1080
token_type
Token, 462
token_width
shared.l0, 1035
tokenize
l0_lexer.Lexer, 319
lexer.l0, 984
TokenRecovery
tokens.l0, 1075
tokens
l0_parser.Parser, 390
ParserState, 393
tokens.l0
ident_to_token, 1075
is_expr_ending_token_type, 1077
is_reserved_ident_word, 1078
is_reserved_keyword, 1078
token_logical_predecessor_ends_expression, 1078
token_recovered, 1079
token_recovery_is_some, 1079
token_recovery_release_payload, 1079
token_recovery_to_string, 1080
token_release_payload, 1080
token_to_string, 1080
TokenRecovery, 1075

TokenType, 1075
tv_create, 1080
tv_free, 1080
tv_get, 1081
tv_len, 1081
tv_push, 1081
TokenType
tokens.l0, 1075
TokenVector, 473
vec, 473
tomb_count
IntSet, 297
StringIntMap, 437
StringPtrMap, 440
StringSet, 442
top_level_decl_free
ast.l0, 791
top_level_decl_vec_free
ast.l0, 791
TopLevelDecl, 475
data, 475
kind, 475
TopLevelDeclKind
ast.l0, 779
total
check_trace_log._MessageCollector, 65
tr_ambiguous_type_modules
type_resolve.l0, 1083
tr_apply_suffix
type_resolve.l0, 1083
tr_diag_error
type_resolve.l0, 1083
tr_format_ambiguous_hints
type_resolve.l0, 1084
tr_format_ambiguous_modules
type_resolve.l0, 1084
tr_key
type_resolve.l0, 1084
tr_module_from_path
type_resolve.l0, 1084
tr_resolve_alias
type_resolve.l0, 1085
tr_resolve_type_ref
type_resolve.l0, 1085
tr_symbol_lookup
type_resolve.l0, 1086
tr_type_ref_full_name
type_resolve.l0, 1086
tr_type_ref_simple_name
type_resolve.l0, 1086
trace_arc
CliOptions, 205
CodegenOptions, 208
l0_context.CompilationContext, 210
trace_memory
CliOptions, 205
CodegenOptions, 208
l0_context.CompilationContext, 210
TRACE_RE
check_trace_log, 24
tree_left
_rt_alloc_record, 68
tree_prio
_rt_alloc_record, 68
tree_right
_rt_alloc_record, 68
trim_s
string.l0, 602
TRUE
l0_lexer.TokenKind, 472
tv_create
tokens.l0, 1080
tv_free
tokens.l0, 1080
tv_get
tokens.l0, 1081
tv_len
tokens.l0, 1081
tv_push
tokens.l0, 1081
TY_BUILTIN
types.l0, 1088
TY_ENUM
types.l0, 1088
TY_FUNC
types.l0, 1088
TY_NULL
types.l0, 1088
TY_NULLABLE
types.l0, 1088
TY_POINTER
types.l0, 1088
TY_STRUCT
types.l0, 1088
TYPE
l0_lexer.TokenKind, 472
Type, 478
inner, 479
kind, 479
module_name, 479
name, 479
params, 479
result, 479
type
l0_ast.FieldDecl, 271
l0_ast.LetDecl, 311
l0_ast.LetStmt, 314
l0_ast.Param, 371
l0_resolve.TypeResolution, 488
l0_signatures.StructFieldInfo, 447
l0_symbols.Symbol, 454
TYPE_ALIAS
l0_symbols.SymbolKind, 457
type_alias_decl_free
ast.l0, 791
type_builtin_bool

types.l0, 1089
type_builtin_byte
types.l0, 1089
type_builtin_int
types.l0, 1090
type_builtin_string
types.l0, 1090
type_builtin_void
types.l0, 1090
type_clone
types.l0, 1090
type_equals
types.l0, 1091
type_format
types.l0, 1091
type_free
types.l0, 1091
type_free_opt
backend.l0, 835
type_id
_rt_alloc_record_cold, 70
type_is_builtin
types.l0, 1092
type_is_builtin_name
types.l0, 1092
type_new
types.l0, 1092
type_new_builtin
types.l0, 1093
type_new_enum
types.l0, 1093
type_new_func
types.l0, 1093
type_new_null
types.l0, 1094
type_new_nullable
types.l0, 1094
type_new_pointer
types.l0, 1094
type_new_struct
types.l0, 1094
type_param_vec_free
types.l0, 1095
type_ref
ExprNode, 266
FieldDecl, 270
l0_ast.NewExpr, 363
l0_ast.TypeExpr, 483
l0_locals.LocalSymbol, 333
LetDecl, 313
Param, 372
StmtNode, 435
type_ref_free
ast.l0, 791
type_resolve.l0
tr_ambiguous_type_modules, 1083
tr_apply_suffix, 1083
tr_diag_error, 1083
tr_format_ambiguous_hints, 1084
tr_format_ambiguous_modules, 1084
tr_key, 1084
tr_module_from_path, 1084
tr_resolve_alias, 1085
tr_resolve_type_ref, 1085
tr_symbol_lookup, 1086
tr_type_ref_full_name, 1086
tr_type_ref_simple_name, 1086
TypeAliasDecl, 481
name, 482
span, 482
target, 482
TypeKind
types.l0, 1088
TypeRef, 486
is_nullable, 486
module_path, 486
name, 486
name_qualifier, 486
pointer_depth, 487
span, 487
types.l0
enum_info_free, 1089
sig_tables_free, 1089
struct_info_free, 1089
TY_BUILTIN, 1088
TY_ENUM, 1088
TY_FUNC, 1088
TY_NULL, 1088
TY_NULLABLE, 1088
TY_POINTER, 1088
TY_STRUCT, 1088
type_builtin_bool, 1089
type_builtin_byte, 1089
type_builtin_int, 1090
type_builtin_string, 1090
type_builtin_void, 1090
type_clone, 1090
type_equals, 1091
type_format, 1091
type_free, 1091
type_is_builtin, 1092
type_is_builtin_name, 1092
type_new, 1092
type_new_builtin, 1093
type_new_enum, 1093
type_new_func, 1093
type_new_null, 1094
type_new_nullable, 1094
type_new_pointer, 1094
type_new_struct, 1094
type_param_vec_free, 1095
TypeKind, 1088

unchecked
CliOptions, 206
UNDERSCORE
l0_lexer.TokenKind, 472

Unit, 492
unit
    unit.l0, 624
unit.l0
    present, 624
    unit, 624
units
    DriverState, 222
UNKNOWN_MODULE
    l0_resolve.ResolveErrorKind, 402
    l0_resolve.TypeResolveErrorKind, 490
UNKNOWN_SYMBOL
    l0_resolve.ResolveErrorKind, 402
UNKNOWN_TYPE
    l0_resolve.TypeResolveErrorKind, 490
UNRESOLVED_ALIAS
    l0_resolve.TypeResolveErrorKind, 490
unwrap_or_b
    optional.l0, 590
unwrap_or_i
    optional.l0, 590
unwrap_or_s
    optional.l0, 591
up_basename
    path.l0, 1117
up_find_last_sep
    path.l0, 1117
up_has_parent
    path.l0, 1117
up_is_absolute
    path.l0, 1117
up_is_l0_file
    path.l0, 1118
up_is_sep
    path.l0, 1118
up_join
    path.l0, 1118
up_parent
    path.l0, 1119
up_stem
    path.l0, 1119
update
    l0_ast.ForStmt, 275
us_is_utf8_continuation
    strings.l0, 1120
us_is_valid_utf8
    strings.l0, 1120
us_split_char
    strings.l0, 1121
us_split_chars2_non_empty
    strings.l0, 1121
us_split_is_space_non_empty
    strings.l0, 1121
us_strip_utf8_bom
    strings.l0, 1122
us_sv_clone
    strings.l0, 1122
utc_now_datetime
    time.l0, 623
utc_offset_sec
    DateTime, 216

value
    IntStringLinearMapEntry, 299
    l0_ast.AssignStmt, 80
    l0_ast.BoolLiteral, 129
    l0_ast.ByteLiteral, 135
    l0_ast.IntLiteral, 296
    l0_ast.LetDecl, 312
    l0_ast.LetStmt, 315
    l0_ast.ReturnStmt, 403
    l0_ast.StringLiteral, 439
    l0_opt_bool, 300
    l0_opt_byte, 300
    l0_opt_int, 301
    l0_opt_string, 302
    LetDecl, 313
    StringStringLinearMapEntry, 443
value_size
    LinearMapBase, 324
values
    StringIntMap, 437
    StringPtrMap, 440
var_ref_resolution
    AnalysisResult, 74
    l0_analysis.AnalysisResult, 78
VAR_RES_LOCAL
    sem_context.l0, 1048
VAR_RES_MODULE
    sem_context.l0, 1048
var_type
    ScopeVar, 415
VARIANT_AS_TYPE
    l0_resolve.TypeResolveErrorKind, 490
variants
    EnumDecl, 226
    EnumInfo, 229
    l0_ast.EnumDecl, 227
    l0_signatures.EnumInfo, 230
VarRefResolution
    sem_context.l0, 1047
vars
    l0_ast.VariantPattern, 494
    PatternNode, 397
vec
    TokenVector, 473
vec_capacity
    vector.l0, 629
vec_check
    vector.l0, 629
vec_clear
    vector.l0, 629
vec_create
    vector.l0, 629
vec_free
    vector.l0, 630
vec_get

vector.l0, 630
vec_grow
vector.l0, 630
vec_push
vector.l0, 631
vec_push_bool
vector.l0, 631
vec_push_byte
vector.l0, 631
vec_push_bytes
vector.l0, 631
vec_push_int
vector.l0, 632
vec_push_ptr
vector.l0, 632
vec_reserve
vector.l0, 632
vec_size
vector.l0, 632
vec_zap
vector.l0, 633
vector.l0
_sv_less, 626
StringVector, 633
sv_capacity, 627
sv_clear, 627
sv_create, 627
sv_free, 627
sv_get, 628
sv_push, 628
sv_size, 628
sv_sort, 628
vec_capacity, 629
vec_check, 629
vec_clear, 629
vec_create, 629
vec_free, 630
vec_get, 630
vec_grow, 630
vec_push, 631
vec_push_bool, 631
vec_push_byte, 631
vec_push_bytes, 631
vec_push_int, 632
vec_push_ptr, 632
vec_reserve, 632
vec_size, 632
vec_zap, 633
vi_sort, 633
VectorBase, 498
arr, 498
length, 498
verbosity
CliOptions, 206
CliParseResult, 207
version_requested
CliParseResult, 207
vi_sort
vector.l0, 633
void_type
l0_expr_types.ExpressionTypeChecker, 264

wall_now
time.l0, 623
wall_to_local_datetime
time.l0, 623
wall_to_utc_datetime
time.l0, 624
WallTime, 499
nsec, 499
sec, 499
WARNING
l0_context.LogLevel, 336
Warning
diag.l0, 1096
weekday
CivilDate, 201
DateTime, 216
WHILE
l0_lexer.TokenKind, 472
WITH
l0_lexer.TokenKind, 472
with_cleanup_block
l0_scope_context.ScopeContext, 411
ScopeContext, 412
with_cleanup_in_progress
l0_scope_context.ScopeContext, 411
ScopeContext, 412
with_cleanup_inline
l0_scope_context.ScopeContext, 411
ScopeContext, 412
with_item_vec_free
ast.l0, 792
with_items
StmtNode, 435
WithItem, 504
cleanup_stmt, 505
has_cleanup, 505
init, 505
span, 505
write_file
fs.l0, 518
write_stderr_all
io.l0, 573
write_stderr_some
io.l0, 573
write_stdout_all
io.l0, 573
write_stdout_some
io.l0, 574

year
CivilDate, 201
DateTime, 216
yearday
CivilDate, 201
DateTime, 216