

Kalashnikov DB

0.9.2

Generated by Doxygen 1.8.13



# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Todo List</b>                             | <b>1</b>  |
| <b>2</b> | <b>Class Index</b>                           | <b>3</b>  |
| 2.1      | Class List . . . . .                         | 3         |
| <b>3</b> | <b>File Index</b>                            | <b>7</b>  |
| 3.1      | File List . . . . .                          | 7         |
| <b>4</b> | <b>Class Documentation</b>                   | <b>11</b> |
| 4.1      | _dictionary_ Struct Reference . . . . .      | 11        |
| 4.1.1    | Detailed Description . . . . .               | 11        |
| 4.1.2    | Member Data Documentation . . . . .          | 11        |
| 4.1.2.1  | hash . . . . .                               | 11        |
| 4.1.2.2  | key . . . . .                                | 12        |
| 4.1.2.3  | size . . . . .                               | 12        |
| 4.1.2.4  | val . . . . .                                | 12        |
| 4.2      | _file_metadata Struct Reference . . . . .    | 12        |
| 4.3      | _notifyDetails Struct Reference . . . . .    | 12        |
| 4.4      | AK_agg_input Struct Reference . . . . .      | 13        |
| 4.4.1    | Detailed Description . . . . .               | 13        |
| 4.5      | AK_agg_value Struct Reference . . . . .      | 13        |
| 4.5.1    | Detailed Description . . . . .               | 14        |
| 4.6      | AK_block Struct Reference . . . . .          | 14        |
| 4.6.1    | Detailed Description . . . . .               | 15        |
| 4.7      | AK_block_activity Struct Reference . . . . . | 15        |

|   |    |
|---|----|
| 4.7.1 Detailed Description . . . . .                      | 16 |
| 4.8 AK_blocktable Struct Reference . . . . .              | 16 |
| 4.9 AK_command_recovery_struct Struct Reference . . . . . | 16 |
| 4.9.1 Detailed Description . . . . .                      | 17 |
| 4.10 AK_command_struct Struct Reference . . . . .         | 17 |
| 4.11 AK_create_table_struct Struct Reference . . . . .    | 17 |
| 4.12 AK_db_cache Struct Reference . . . . .               | 18 |
| 4.12.1 Detailed Description . . . . .                     | 18 |
| 4.13 AK_debmod_state Struct Reference . . . . .           | 19 |
| 4.13.1 Detailed Description . . . . .                     | 19 |
| 4.14 AK_header Struct Reference . . . . .                 | 19 |
| 4.14.1 Detailed Description . . . . .                     | 20 |
| 4.15 AK_mem_block Struct Reference . . . . .              | 20 |
| 4.15.1 Detailed Description . . . . .                     | 21 |
| 4.16 AK_operand Struct Reference . . . . .                | 21 |
| 4.17 AK_query_mem Struct Reference . . . . .              | 22 |
| 4.17.1 Detailed Description . . . . .                     | 22 |
| 4.18 AK_query_mem_dict Struct Reference . . . . .         | 23 |
| 4.18.1 Detailed Description . . . . .                     | 23 |
| 4.19 AK_query_mem_lib Struct Reference . . . . .          | 23 |
| 4.19.1 Detailed Description . . . . .                     | 24 |
| 4.20 AK_query_mem_result Struct Reference . . . . .       | 24 |
| 4.20.1 Detailed Description . . . . .                     | 25 |
| 4.21 AK_redo_log Struct Reference . . . . .               | 25 |
| 4.21.1 Detailed Description . . . . .                     | 26 |
| 4.22 AK_ref_item Struct Reference . . . . .               | 26 |
| 4.22.1 Detailed Description . . . . .                     | 26 |
| 4.23 AK_results Struct Reference . . . . .                | 27 |
| 4.23.1 Detailed Description . . . . .                     | 27 |
| 4.24 AK_synchronization_info Struct Reference . . . . .   | 28 |

|  |    |
|--|----|
| 4.24.1 Detailed Description . . . . .              | 28 |
| 4.25 AK_tuple_dict Struct Reference . . . . .      | 28 |
| 4.25.1 Detailed Description . . . . .              | 29 |
| 4.26 blocktable Struct Reference . . . . .         | 29 |
| 4.26.1 Detailed Description . . . . .              | 29 |
| 4.27 btree_node Struct Reference . . . . .         | 29 |
| 4.28 bucket_elem Struct Reference . . . . .        | 30 |
| 4.28.1 Detailed Description . . . . .              | 30 |
| 4.29 cost_eval_t Struct Reference . . . . .        | 31 |
| 4.29.1 Detailed Description . . . . .              | 31 |
| 4.30 DEBUG_LEVEL Struct Reference . . . . .        | 31 |
| 4.30.1 Detailed Description . . . . .              | 31 |
| 4.31 DEBUG_TYPE Struct Reference . . . . .         | 32 |
| 4.31.1 Detailed Description . . . . .              | 32 |
| 4.32 drop_arguments Struct Reference . . . . .     | 32 |
| 4.33 hash_bucket Struct Reference . . . . .        | 33 |
| 4.33.1 Detailed Description . . . . .              | 33 |
| 4.34 hash_info Struct Reference . . . . .          | 34 |
| 4.34.1 Detailed Description . . . . .              | 34 |
| 4.35 intersect_attr Struct Reference . . . . .     | 34 |
| 4.35.1 Detailed Description . . . . .              | 35 |
| 4.36 list_node Struct Reference . . . . .          | 35 |
| 4.36.1 Detailed Description . . . . .              | 35 |
| 4.37 list_structure_ad Struct Reference . . . . .  | 36 |
| 4.38 list_structure_add Struct Reference . . . . . | 36 |
| 4.38.1 Detailed Description . . . . .              | 36 |
| 4.39 main_bucket Struct Reference . . . . .        | 37 |
| 4.39.1 Detailed Description . . . . .              | 37 |
| 4.40 memoryAddresses Struct Reference . . . . .    | 38 |
| 4.40.1 Detailed Description . . . . .              | 38 |

|        |  |    |
|--------|--|----|
| 4.41   | Observable Struct Reference                    | 38 |
| 4.41.1 | Detailed Description                           | 39 |
| 4.42   | observable_transaction Struct Reference        | 39 |
| 4.42.1 | Detailed Description                           | 39 |
| 4.43   | observable_transaction_struct Struct Reference | 40 |
| 4.44   | Observer Struct Reference                      | 40 |
| 4.44.1 | Detailed Description                           | 41 |
| 4.45   | observer_lock Struct Reference                 | 41 |
| 4.45.1 | Detailed Description                           | 41 |
| 4.46   | root_info Struct Reference                     | 42 |
| 4.47   | search_params Struct Reference                 | 42 |
| 4.47.1 | Detailed Description                           | 42 |
| 4.48   | search_result Struct Reference                 | 43 |
| 4.48.1 | Detailed Description                           | 43 |
| 4.49   | Stack Struct Reference                         | 43 |
| 4.49.1 | Detailed Description                           | 44 |
| 4.50   | struct_add Struct Reference                    | 44 |
| 4.50.1 | Detailed Description                           | 44 |
| 4.51   | Succesor Struct Reference                      | 45 |
| 4.51.1 | Detailed Description                           | 45 |
| 4.52   | table_addresses Struct Reference               | 45 |
| 4.52.1 | Detailed Description                           | 46 |
| 4.53   | threadContainer Struct Reference               | 46 |
| 4.53.1 | Detailed Description                           | 46 |
| 4.54   | transaction_list_elem Struct Reference         | 47 |
| 4.54.1 | Detailed Description                           | 47 |
| 4.55   | transaction_list_head Struct Reference         | 48 |
| 4.55.1 | Detailed Description                           | 48 |
| 4.56   | transaction_locks_list_elem Struct Reference   | 49 |
| 4.56.1 | Detailed Description                           | 49 |
| 4.57   | transactionData Struct Reference               | 49 |
| 4.57.1 | Detailed Description                           | 50 |
| 4.58   | TypeObservable Struct Reference                | 51 |
| 4.59   | TypeObserver Struct Reference                  | 52 |
| 4.60   | Vertex Struct Reference                        | 52 |
| 4.60.1 | Detailed Description                           | 53 |

|          |  |           |
|----------|--|-----------|
| <b>5</b> | <b>File Documentation</b>                | <b>55</b> |
| 5.1      | aux/auxiliary.c File Reference . . . . . | 55        |
| 5.1.1    | Detailed Description . . . . .           | 57        |
| 5.1.2    | Function Documentation . . . . .         | 57        |
| 5.1.2.1  | AK_add_sucesor() . . . . .               | 57        |
| 5.1.2.2  | AK_add_vertex() . . . . .                | 57        |
| 5.1.2.3  | AK_chars_num_from_number() . . . . .     | 58        |
| 5.1.2.4  | Ak_Delete_L3() . . . . .                 | 58        |
| 5.1.2.5  | Ak_DeleteAll_L3() . . . . .              | 59        |
| 5.1.2.6  | AK_destroy_critical_section() . . . . .  | 59        |
| 5.1.2.7  | Ak_End_L2() . . . . .                    | 60        |
| 5.1.2.8  | AK_enter_critical_section() . . . . .    | 60        |
| 5.1.2.9  | Ak_First_L2() . . . . .                  | 60        |
| 5.1.2.10 | AK_get_array_perms() . . . . .           | 61        |
| 5.1.2.11 | Ak_GetNth_L2() . . . . .                 | 61        |
| 5.1.2.12 | AK_init_critical_section() . . . . .     | 62        |
| 5.1.2.13 | Ak_Init_L3() . . . . .                   | 63        |
| 5.1.2.14 | Ak_InsertAfter_L2() . . . . .            | 63        |
| 5.1.2.15 | Ak_InsertAtBegin_L3() . . . . .          | 64        |
| 5.1.2.16 | Ak_InsertAtEnd_L3() . . . . .            | 64        |
| 5.1.2.17 | Ak_InsertBefore_L2() . . . . .           | 65        |
| 5.1.2.18 | Ak_IsEmpty_L2() . . . . .                | 65        |
| 5.1.2.19 | AK_leave_critical_section() . . . . .    | 66        |
| 5.1.2.20 | Ak_Next_L2() . . . . .                   | 66        |
| 5.1.2.21 | AK_pop_from_stack() . . . . .            | 67        |
| 5.1.2.22 | Ak_Previous_L2() . . . . .               | 67        |
| 5.1.2.23 | AK_push_to_stack() . . . . .             | 67        |
| 5.1.2.24 | Ak_Retrieve_L2() . . . . .               | 68        |
| 5.1.2.25 | AK_search_empty_link() . . . . .         | 68        |
| 5.1.2.26 | AK_search_empty_stack_link() . . . . .   | 69        |

|          |  |    |
|----------|--|----|
| 5.1.2.27 | <a href="#">AK_search_in_stack()</a>           | 69 |
| 5.1.2.28 | <a href="#">AK_search_vertex()</a>             | 70 |
| 5.1.2.29 | <a href="#">Ak_Size_L2()</a>                   | 70 |
| 5.1.2.30 | <a href="#">AK_strcmp()</a>                    | 70 |
| 5.1.2.31 | <a href="#">AK_tarjan()</a>                    | 71 |
| 5.1.2.32 | <a href="#">AK_type_size()</a>                 | 71 |
| 5.2      | <a href="#">aux/auxiliary.h File Reference</a> | 72 |
| 5.2.1    | <a href="#">Detailed Description</a>           | 75 |
| 5.2.2    | <a href="#">Function Documentation</a>         | 75 |
| 5.2.2.1  | <a href="#">AK_add_succesor()</a>              | 75 |
| 5.2.2.2  | <a href="#">AK_add_vertex()</a>                | 75 |
| 5.2.2.3  | <a href="#">AK_chars_num_from_number()</a>     | 76 |
| 5.2.2.4  | <a href="#">Ak_Delete_L3()</a>                 | 76 |
| 5.2.2.5  | <a href="#">Ak_DeleteAll_L3()</a>              | 77 |
| 5.2.2.6  | <a href="#">AK_destroy_critical_section()</a>  | 77 |
| 5.2.2.7  | <a href="#">Ak_End_L2()</a>                    | 77 |
| 5.2.2.8  | <a href="#">AK_enter_critical_section()</a>    | 79 |
| 5.2.2.9  | <a href="#">Ak_First_L2()</a>                  | 79 |
| 5.2.2.10 | <a href="#">AK_get_array_perms()</a>           | 80 |
| 5.2.2.11 | <a href="#">Ak_GetNth_L2()</a>                 | 80 |
| 5.2.2.12 | <a href="#">AK_init_critical_section()</a>     | 81 |
| 5.2.2.13 | <a href="#">Ak_Init_L3()</a>                   | 82 |
| 5.2.2.14 | <a href="#">Ak_InsertAfter_L2()</a>            | 82 |
| 5.2.2.15 | <a href="#">Ak_InsertAtBegin_L3()</a>          | 83 |
| 5.2.2.16 | <a href="#">Ak_InsertAtEnd_L3()</a>            | 83 |
| 5.2.2.17 | <a href="#">Ak_InsertBefore_L2()</a>           | 84 |
| 5.2.2.18 | <a href="#">Ak_IsEmpty_L2()</a>                | 84 |
| 5.2.2.19 | <a href="#">AK_leave_critical_section()</a>    | 85 |
| 5.2.2.20 | <a href="#">Ak_Next_L2()</a>                   | 85 |
| 5.2.2.21 | <a href="#">AK_pop_from_stack()</a>            | 85 |



|          |  |     |
|----------|--|-----|
| 5.2.2.22 | <a href="#">Ak_Previous_L2()</a>                 | 86  |
| 5.2.2.23 | <a href="#">Ak_push_to_stack()</a>               | 86  |
| 5.2.2.24 | <a href="#">Ak_Retrieve_L2()</a>                 | 87  |
| 5.2.2.25 | <a href="#">Ak_search_empty_link()</a>           | 87  |
| 5.2.2.26 | <a href="#">Ak_search_empty_stack_link()</a>     | 88  |
| 5.2.2.27 | <a href="#">Ak_search_in_stack()</a>             | 88  |
| 5.2.2.28 | <a href="#">Ak_search_vertex()</a>               | 88  |
| 5.2.2.29 | <a href="#">Ak_Size_L2()</a>                     | 89  |
| 5.2.2.30 | <a href="#">Ak_strcmp()</a>                      | 89  |
| 5.2.2.31 | <a href="#">Ak_tarjan()</a>                      | 90  |
| 5.2.2.32 | <a href="#">Ak_type_size()</a>                   | 90  |
| 5.2.3    | <a href="#">Variable Documentation</a>           | 91  |
| 5.2.3.1  | <a href="#">testMode</a>                         | 91  |
| 5.3      | <a href="#">auxi/constants.h File Reference</a>  | 91  |
| 5.3.1    | <a href="#">Detailed Description</a>             | 95  |
| 5.4      | <a href="#">auxi/debug.c File Reference</a>      | 95  |
| 5.4.1    | <a href="#">Detailed Description</a>             | 96  |
| 5.4.2    | <a href="#">Function Documentation</a>           | 96  |
| 5.4.2.1  | <a href="#">Ak_dbg_messg()</a>                   | 96  |
| 5.5      | <a href="#">auxi/debug.h File Reference</a>      | 96  |
| 5.5.1    | <a href="#">Detailed Description</a>             | 98  |
| 5.5.2    | <a href="#">Macro Definition Documentation</a>   | 98  |
| 5.5.2.1  | <a href="#">DEBUG_ALL</a>                        | 98  |
| 5.5.3    | <a href="#">Function Documentation</a>           | 98  |
| 5.5.3.1  | <a href="#">Ak_dbg_messg()</a>                   | 98  |
| 5.6      | <a href="#">auxi/dictionary.c File Reference</a> | 99  |
| 5.6.1    | <a href="#">Detailed Description</a>             | 100 |
| 5.6.2    | <a href="#">Macro Definition Documentation</a>   | 100 |
| 5.6.2.1  | <a href="#">DICT_INVALID_KEY</a>                 | 100 |
| 5.6.2.2  | <a href="#">DICTMINSZ</a>                        | 100 |

|         |                                 |     |
|---------|---------------------------------|-----|
| 5.6.2.3 | MAXVALSZ                        | 100 |
| 5.6.3   | Function Documentation          | 100 |
| 5.6.3.1 | dictionary_del()                | 100 |
| 5.6.3.2 | dictionary_dump()               | 101 |
| 5.6.3.3 | dictionary_get()                | 101 |
| 5.6.3.4 | dictionary_hash()               | 102 |
| 5.6.3.5 | dictionary_new()                | 102 |
| 5.6.3.6 | dictionary_set()                | 102 |
| 5.6.3.7 | dictionary_unset()              | 103 |
| 5.7     | aux/dictionary.h File Reference | 103 |
| 5.7.1   | Detailed Description            | 105 |
| 5.7.2   | Typedef Documentation           | 105 |
| 5.7.2.1 | dictionary                      | 105 |
| 5.7.3   | Function Documentation          | 105 |
| 5.7.3.1 | dictionary_del()                | 105 |
| 5.7.3.2 | dictionary_dump()               | 106 |
| 5.7.3.3 | dictionary_get()                | 106 |
| 5.7.3.4 | dictionary_hash()               | 107 |
| 5.7.3.5 | dictionary_new()                | 107 |
| 5.7.3.6 | dictionary_set()                | 107 |
| 5.7.3.7 | dictionary_unset()              | 108 |
| 5.8     | aux/iniparser.c File Reference  | 108 |
| 5.8.1   | Detailed Description            | 110 |
| 5.8.2   | Typedef Documentation           | 110 |
| 5.8.2.1 | line_status                     | 110 |
| 5.8.3   | Enumeration Type Documentation  | 110 |
| 5.8.3.1 | _line_status_                   | 111 |
| 5.8.4   | Function Documentation          | 111 |
| 5.8.4.1 | iniparser_AK_freedict()         | 111 |
| 5.8.4.2 | iniparser_dump()                | 111 |

|          |   |     |
|----------|---|-----|
| 5.8.4.3  | <a href="#">iniparser_dump_ini()</a>            | 112 |
| 5.8.4.4  | <a href="#">iniparser_dumpsection_ini()</a>     | 112 |
| 5.8.4.5  | <a href="#">iniparser_find_entry()</a>          | 112 |
| 5.8.4.6  | <a href="#">iniparser_getboolean()</a>          | 113 |
| 5.8.4.7  | <a href="#">iniparser_getdouble()</a>           | 114 |
| 5.8.4.8  | <a href="#">iniparser_getint()</a>              | 114 |
| 5.8.4.9  | <a href="#">iniparser_getnsec()</a>             | 115 |
| 5.8.4.10 | <a href="#">iniparser_getseckey()</a>           | 115 |
| 5.8.4.11 | <a href="#">iniparser_getsecname()</a>          | 115 |
| 5.8.4.12 | <a href="#">iniparser_getsecnkeys()</a>         | 116 |
| 5.8.4.13 | <a href="#">iniparser_getstring()</a>           | 116 |
| 5.8.4.14 | <a href="#">iniparser_load()</a>                | 117 |
| 5.8.4.15 | <a href="#">iniparser_set()</a>                 | 117 |
| 5.8.4.16 | <a href="#">iniparser_unset()</a>               | 118 |
| 5.9      | <a href="#">auxi/iniparser.h File Reference</a> | 118 |
| 5.9.1    | <a href="#">Detailed Description</a>            | 120 |
| 5.9.2    | <a href="#">Function Documentation</a>          | 120 |
| 5.9.2.1  | <a href="#">iniparser_AK_freedit()</a>          | 120 |
| 5.9.2.2  | <a href="#">iniparser_dump()</a>                | 120 |
| 5.9.2.3  | <a href="#">iniparser_dump_ini()</a>            | 121 |
| 5.9.2.4  | <a href="#">iniparser_dumpsection_ini()</a>     | 121 |
| 5.9.2.5  | <a href="#">iniparser_find_entry()</a>          | 122 |
| 5.9.2.6  | <a href="#">iniparser_getboolean()</a>          | 122 |
| 5.9.2.7  | <a href="#">iniparser_getdouble()</a>           | 123 |
| 5.9.2.8  | <a href="#">iniparser_getint()</a>              | 123 |
| 5.9.2.9  | <a href="#">iniparser_getnsec()</a>             | 124 |
| 5.9.2.10 | <a href="#">iniparser_getseckey()</a>           | 125 |
| 5.9.2.11 | <a href="#">iniparser_getsecname()</a>          | 125 |
| 5.9.2.12 | <a href="#">iniparser_getsecnkeys()</a>         | 126 |
| 5.9.2.13 | <a href="#">iniparser_getstring()</a>           | 126 |

|           |  |     |
|-----------|--|-----|
| 5.9.2.14  | <a href="#">iniparser_load()</a>               | 127 |
| 5.9.2.15  | <a href="#">iniparser_set()</a>                | 127 |
| 5.9.2.16  | <a href="#">iniparser_unset()</a>              | 127 |
| 5.10      | <a href="#">auxi/mempro.c File Reference</a>   | 128 |
| 5.10.1    | <a href="#">Detailed Description</a>           | 129 |
| 5.10.2    | <a href="#">Function Documentation</a>         | 130 |
| 5.10.2.1  | <a href="#">AK_malloc()</a>                    | 130 |
| 5.10.2.2  | <a href="#">AK_check_for_writes()</a>          | 130 |
| 5.10.2.3  | <a href="#">AK_debmod_malloc()</a>             | 130 |
| 5.10.2.4  | <a href="#">AK_debmod_d()</a>                  | 131 |
| 5.10.2.5  | <a href="#">AK_debmod_die()</a>                | 131 |
| 5.10.2.6  | <a href="#">AK_debmod_dv()</a>                 | 132 |
| 5.10.2.7  | <a href="#">AK_debmod_enter_critical_sec()</a> | 132 |
| 5.10.2.8  | <a href="#">AK_debmod_free()</a>               | 133 |
| 5.10.2.9  | <a href="#">AK_debmod_fstack_pop()</a>         | 133 |
| 5.10.2.10 | <a href="#">AK_debmod_fstack_push()</a>        | 133 |
| 5.10.2.11 | <a href="#">AK_debmod_func_add()</a>           | 134 |
| 5.10.2.12 | <a href="#">AK_debmod_func_get_name()</a>      | 134 |
| 5.10.2.13 | <a href="#">AK_debmod_func_id()</a>            | 135 |
| 5.10.2.14 | <a href="#">AK_debmod_function_current()</a>   | 135 |
| 5.10.2.15 | <a href="#">AK_debmod_function_epilogue()</a>  | 136 |
| 5.10.2.16 | <a href="#">AK_debmod_function_prologue()</a>  | 136 |
| 5.10.2.17 | <a href="#">AK_debmod_init()</a>               | 137 |
| 5.10.2.18 | <a href="#">AK_debmod_leave_critical_sec()</a> | 137 |
| 5.10.2.19 | <a href="#">AK_debmod_log_memory_alloc()</a>   | 138 |
| 5.10.2.20 | <a href="#">AK_debmod_print_function_use()</a> | 138 |
| 5.10.2.21 | <a href="#">AK_fread()</a>                     | 138 |
| 5.10.2.22 | <a href="#">AK_free()</a>                      | 139 |
| 5.10.2.23 | <a href="#">AK_fwrite()</a>                    | 139 |
| 5.10.2.24 | <a href="#">AK_malloc()</a>                    | 140 |

|  |     |
|--|-----|
| 5.10.2.25 AK_mempro_test()               | 140 |
| 5.10.2.26 AK_print_active_functions()    | 140 |
| 5.10.2.27 AK_print_function_use()        | 141 |
| 5.10.2.28 AK_print_function_uses()       | 141 |
| 5.10.2.29 AK_realloc()                   | 141 |
| 5.10.2.30 AK_write_protect()             | 142 |
| 5.10.2.31 AK_write_unprotect()           | 142 |
| 5.11 auxi/mempro.h File Reference        | 143 |
| 5.11.1 Detailed Description              | 145 |
| 5.11.2 Function Documentation            | 145 |
| 5.11.2.1 AK_calloc()                     | 145 |
| 5.11.2.2 AK_check_for_writes()           | 146 |
| 5.11.2.3 AK_debmod_calloc()              | 146 |
| 5.11.2.4 AK_debmod_d()                   | 146 |
| 5.11.2.5 AK_debmod_die()                 | 147 |
| 5.11.2.6 AK_debmod_dv()                  | 147 |
| 5.11.2.7 AK_debmod_enter_critical_sec()  | 148 |
| 5.11.2.8 AK_debmod_free()                | 148 |
| 5.11.2.9 AK_debmod_fstack_pop()          | 149 |
| 5.11.2.10 AK_debmod_fstack_push()        | 149 |
| 5.11.2.11 AK_debmod_func_add()           | 149 |
| 5.11.2.12 AK_debmod_func_get_name()      | 150 |
| 5.11.2.13 AK_debmod_func_id()            | 150 |
| 5.11.2.14 AK_debmod_function_current()   | 151 |
| 5.11.2.15 AK_debmod_function_epilogue()  | 151 |
| 5.11.2.16 AK_debmod_function_prologue()  | 152 |
| 5.11.2.17 AK_debmod_init()               | 152 |
| 5.11.2.18 AK_debmod_leave_critical_sec() | 153 |
| 5.11.2.19 AK_debmod_log_memory_alloc()   | 153 |
| 5.11.2.20 AK_debmod_print_function_use() | 153 |

|           |  |     |
|-----------|--|-----|
| 5.11.2.21 | <a href="#">AK_free()</a>                          | 154 |
| 5.11.2.22 | <a href="#">AK_malloc()</a>                        | 154 |
| 5.11.2.23 | <a href="#">AK_mempro_test()</a>                   | 155 |
| 5.11.2.24 | <a href="#">AK_print_active_functions()</a>        | 155 |
| 5.11.2.25 | <a href="#">AK_print_function_use()</a>            | 155 |
| 5.11.2.26 | <a href="#">AK_print_function_uses()</a>           | 156 |
| 5.11.2.27 | <a href="#">AK_realloc()</a>                       | 156 |
| 5.11.2.28 | <a href="#">AK_write_protect()</a>                 | 157 |
| 5.11.2.29 | <a href="#">AK_write_unprotect()</a>               | 157 |
| 5.12      | <a href="#">auxi/observable.c File Reference</a>   | 157 |
| 5.12.1    | <a href="#">Detailed Description</a>               | 159 |
| 5.12.2    | <a href="#">Function Documentation</a>             | 159 |
| 5.12.2.1  | <a href="#">AK_init_observable()</a>               | 159 |
| 5.12.2.2  | <a href="#">AK_init_observer()</a>                 | 159 |
| 5.12.2.3  | <a href="#">AK_observable_test()</a>               | 160 |
| 5.13      | <a href="#">auxi/observable.h File Reference</a>   | 160 |
| 5.13.1    | <a href="#">Detailed Description</a>               | 161 |
| 5.13.2    | <a href="#">Function Documentation</a>             | 161 |
| 5.13.2.1  | <a href="#">AK_init_observable()</a>               | 161 |
| 5.13.2.2  | <a href="#">AK_init_observer()</a>                 | 162 |
| 5.13.2.3  | <a href="#">AK_observable_test()</a>               | 162 |
| 5.14      | <a href="#">dm/dbman.c File Reference</a>          | 162 |
| 5.14.1    | <a href="#">Detailed Description</a>               | 165 |
| 5.14.2    | <a href="#">Function Documentation</a>             | 165 |
| 5.14.2.1  | <a href="#">AK_allocate_block_activity_modes()</a> | 165 |
| 5.14.2.2  | <a href="#">AK_allocate_blocks()</a>               | 165 |
| 5.14.2.3  | <a href="#">AK_allocationtable_dump()</a>          | 165 |
| 5.14.2.4  | <a href="#">AK_blocktable_dump()</a>               | 166 |
| 5.14.2.5  | <a href="#">AK_blocktable_flush()</a>              | 166 |
| 5.14.2.6  | <a href="#">AK_blocktable_get()</a>                | 166 |

|           |  |     |
|-----------|--|-----|
| 5.14.2.7  | <a href="#">AK_copy_header()</a>                   | 167 |
| 5.14.2.8  | <a href="#">AK_create_header()</a>                 | 167 |
| 5.14.2.9  | <a href="#">AK_delete_block()</a>                  | 168 |
| 5.14.2.10 | <a href="#">AK_delete_extent()</a>                 | 168 |
| 5.14.2.11 | <a href="#">AK_delete_segment()</a>                | 169 |
| 5.14.2.12 | <a href="#">AK_get_allocation_set()</a>            | 169 |
| 5.14.2.13 | <a href="#">AK_get_extent()</a>                    | 170 |
| 5.14.2.14 | <a href="#">AK_increase_extent()</a>               | 170 |
| 5.14.2.15 | <a href="#">AK_init_allocation_table()</a>         | 171 |
| 5.14.2.16 | <a href="#">AK_init_block()</a>                    | 171 |
| 5.14.2.17 | <a href="#">AK_init_db_file()</a>                  | 172 |
| 5.14.2.18 | <a href="#">AK_init_disk_manager()</a>             | 172 |
| 5.14.2.19 | <a href="#">AK_init_system_catalog()</a>           | 172 |
| 5.14.2.20 | <a href="#">AK_init_system_tables_catalog()</a>    | 173 |
| 5.14.2.21 | <a href="#">AK_insert_entry()</a>                  | 174 |
| 5.14.2.22 | <a href="#">AK_memset_int()</a>                    | 175 |
| 5.14.2.23 | <a href="#">AK_new_extent()</a>                    | 175 |
| 5.14.2.24 | <a href="#">AK_new_segment()</a>                   | 176 |
| 5.14.2.25 | <a href="#">AK_print_block()</a>                   | 176 |
| 5.14.2.26 | <a href="#">AK_read_block()</a>                    | 177 |
| 5.14.2.27 | <a href="#">AK_read_block_for_testing()</a>        | 177 |
| 5.14.2.28 | <a href="#">AK_register_system_tables()</a>        | 177 |
| 5.14.2.29 | <a href="#">AK_thread_safe_block_access_test()</a> | 178 |
| 5.14.2.30 | <a href="#">AK_write_block()</a>                   | 179 |
| 5.14.2.31 | <a href="#">AK_write_block_for_testing()</a>       | 179 |
| 5.14.2.32 | <a href="#">fsize()</a>                            | 179 |
| 5.15      | <a href="#">dm/dbman.h File Reference</a>          | 180 |
| 5.15.1    | <a href="#">Detailed Description</a>               | 183 |
| 5.15.2    | <a href="#">Macro Definition Documentation</a>     | 183 |
| 5.15.2.1  | <a href="#">AK_ALLOCATION_TABLE_SIZE</a>           | 184 |

|           |   |     |
|-----------|---|-----|
| 5.15.2.2  | CHAR_IN_LINE . . . . .                    | 184 |
| 5.15.2.3  | MAX_BLOCK_INIT_NUM . . . . .              | 184 |
| 5.15.3    | Enumeration Type Documentation . . . . .  | 184 |
| 5.15.3.1  | AK_allocation_set_mode . . . . .          | 184 |
| 5.15.4    | Function Documentation . . . . .          | 185 |
| 5.15.4.1  | AK_allocate_blocks() . . . . .            | 185 |
| 5.15.4.2  | AK_allocationtable_dump() . . . . .       | 185 |
| 5.15.4.3  | AK_blocktable_dump() . . . . .            | 185 |
| 5.15.4.4  | AK_blocktable_flush() . . . . .           | 186 |
| 5.15.4.5  | AK_blocktable_get() . . . . .             | 186 |
| 5.15.4.6  | AK_copy_header() . . . . .                | 186 |
| 5.15.4.7  | AK_create_header() . . . . .              | 187 |
| 5.15.4.8  | AK_delete_block() . . . . .               | 187 |
| 5.15.4.9  | AK_delete_extent() . . . . .              | 188 |
| 5.15.4.10 | AK_delete_segment() . . . . .             | 188 |
| 5.15.4.11 | AK_get_allocation_set() . . . . .         | 189 |
| 5.15.4.12 | AK_get_extent() . . . . .                 | 189 |
| 5.15.4.13 | AK_increase_extent() . . . . .            | 190 |
| 5.15.4.14 | AK_init_allocation_table() . . . . .      | 191 |
| 5.15.4.15 | AK_init_block() . . . . .                 | 191 |
| 5.15.4.16 | AK_init_db_file() . . . . .               | 192 |
| 5.15.4.17 | AK_init_disk_manager() . . . . .          | 192 |
| 5.15.4.18 | AK_init_system_catalog() . . . . .        | 192 |
| 5.15.4.19 | AK_init_system_tables_catalog() . . . . . | 193 |
| 5.15.4.20 | AK_insert_entry() . . . . .               | 194 |
| 5.15.4.21 | AK_memset_int() . . . . .                 | 195 |
| 5.15.4.22 | AK_new_extent() . . . . .                 | 195 |
| 5.15.4.23 | AK_new_segment() . . . . .                | 196 |
| 5.15.4.24 | AK_print_block() . . . . .                | 196 |
| 5.15.4.25 | AK_read_block() . . . . .                 | 197 |



|           |  |     |
|-----------|--|-----|
| 5.15.4.26 | <a href="#">AK_read_block_for_testing()</a>        | 197 |
| 5.15.4.27 | <a href="#">AK_register_system_tables()</a>        | 197 |
| 5.15.4.28 | <a href="#">AK_thread_safe_block_access_test()</a> | 198 |
| 5.15.4.29 | <a href="#">AK_write_block()</a>                   | 199 |
| 5.15.4.30 | <a href="#">AK_write_block_for_testing()</a>       | 199 |
| 5.15.4.31 | <a href="#">fsize()</a>                            | 199 |
| 5.15.5    | <a href="#">Variable Documentation</a>             | 200 |
| 5.15.5.1  | <a href="#">AK_allocationbit</a>                   | 200 |
| 5.15.5.2  | <a href="#">db</a>                                 | 200 |
| 5.15.5.3  | <a href="#">db_file_size</a>                       | 200 |
| 5.16      | <a href="#">file/blobs.c File Reference</a>        | 201 |
| 5.16.1    | <a href="#">Detailed Description</a>               | 202 |
| 5.16.2    | <a href="#">Function Documentation</a>             | 202 |
| 5.16.2.1  | <a href="#">AK_check_folder_blobs()</a>            | 202 |
| 5.16.2.2  | <a href="#">AK_concat()</a>                        | 202 |
| 5.16.2.3  | <a href="#">AK_folder_exists()</a>                 | 203 |
| 5.16.2.4  | <a href="#">AK_GUID()</a>                          | 203 |
| 5.16.2.5  | <a href="#">AK_lo_export()</a>                     | 203 |
| 5.16.2.6  | <a href="#">AK_lo_import()</a>                     | 204 |
| 5.16.2.7  | <a href="#">AK_lo_test()</a>                       | 204 |
| 5.16.2.8  | <a href="#">AK_lo_unlink()</a>                     | 204 |
| 5.16.2.9  | <a href="#">AK_mkdir()</a>                         | 205 |
| 5.16.2.10 | <a href="#">AK_split_path_file()</a>               | 205 |
| 5.17      | <a href="#">file/blobs.h File Reference</a>        | 205 |
| 5.17.1    | <a href="#">Detailed Description</a>               | 207 |
| 5.17.2    | <a href="#">Function Documentation</a>             | 207 |
| 5.17.2.1  | <a href="#">AK_check_folder_blobs()</a>            | 207 |
| 5.17.2.2  | <a href="#">AK_concat()</a>                        | 207 |
| 5.17.2.3  | <a href="#">AK_folder_exists()</a>                 | 208 |
| 5.17.2.4  | <a href="#">AK_GUID()</a>                          | 208 |

|           |  |     |
|-----------|--|-----|
| 5.17.2.5  | <a href="#">AK_lo_export()</a>                     | 208 |
| 5.17.2.6  | <a href="#">AK_lo_import()</a>                     | 209 |
| 5.17.2.7  | <a href="#">AK_lo_test()</a>                       | 209 |
| 5.17.2.8  | <a href="#">AK_lo_unlink()</a>                     | 209 |
| 5.17.2.9  | <a href="#">AK_mkdir()</a>                         | 210 |
| 5.17.2.10 | <a href="#">AK_split_path_file()</a>               | 210 |
| 5.18      | <a href="#">file/fileio.c File Reference</a>       | 210 |
| 5.18.1    | <a href="#">Detailed Description</a>               | 211 |
| 5.18.2    | <a href="#">Function Documentation</a>             | 211 |
| 5.18.2.1  | <a href="#">Ak_delete_row()</a>                    | 211 |
| 5.18.2.2  | <a href="#">Ak_delete_row_by_id()</a>              | 212 |
| 5.18.2.3  | <a href="#">Ak_delete_row_from_block()</a>         | 212 |
| 5.18.2.4  | <a href="#">Ak_delete_update_segment()</a>         | 213 |
| 5.18.2.5  | <a href="#">Ak_Insert_New_Element()</a>            | 213 |
| 5.18.2.6  | <a href="#">Ak_Insert_New_Element_For_Update()</a> | 214 |
| 5.18.2.7  | <a href="#">Ak_insert_row()</a>                    | 214 |
| 5.18.2.8  | <a href="#">Ak_insert_row_to_block()</a>           | 215 |
| 5.18.2.9  | <a href="#">Ak_update_row()</a>                    | 215 |
| 5.18.2.10 | <a href="#">Ak_update_row_from_block()</a>         | 216 |
| 5.19      | <a href="#">file/fileio.h File Reference</a>       | 216 |
| 5.19.1    | <a href="#">Detailed Description</a>               | 217 |
| 5.19.2    | <a href="#">Function Documentation</a>             | 217 |
| 5.19.2.1  | <a href="#">Ak_delete_row()</a>                    | 218 |
| 5.19.2.2  | <a href="#">Ak_delete_row_by_id()</a>              | 218 |
| 5.19.2.3  | <a href="#">Ak_delete_row_from_block()</a>         | 218 |
| 5.19.2.4  | <a href="#">Ak_delete_update_segment()</a>         | 219 |
| 5.19.2.5  | <a href="#">Ak_Insert_New_Element()</a>            | 219 |
| 5.19.2.6  | <a href="#">Ak_Insert_New_Element_For_Update()</a> | 220 |
| 5.19.2.7  | <a href="#">Ak_insert_row()</a>                    | 221 |
| 5.19.2.8  | <a href="#">Ak_insert_row_to_block()</a>           | 221 |

|           |   |     |
|-----------|---|-----|
| 5.19.2.9  | <a href="#">Ak_update_row()</a>                   | 222 |
| 5.19.2.10 | <a href="#">Ak_update_row_from_block()</a>        | 222 |
| 5.20      | <a href="#">file/files.c File Reference</a>       | 222 |
| 5.20.1    | <a href="#">Detailed Description</a>              | 223 |
| 5.20.2    | <a href="#">Function Documentation</a>            | 223 |
| 5.20.2.1  | <a href="#">Ak_files_test()</a>                   | 223 |
| 5.20.2.2  | <a href="#">AK_initialize_new_index_segment()</a> | 224 |
| 5.20.2.3  | <a href="#">AK_initialize_new_segment()</a>       | 224 |
| 5.21      | <a href="#">file/files.h File Reference</a>       | 225 |
| 5.21.1    | <a href="#">Detailed Description</a>              | 225 |
| 5.21.2    | <a href="#">Function Documentation</a>            | 226 |
| 5.21.2.1  | <a href="#">Ak_files_test()</a>                   | 226 |
| 5.21.2.2  | <a href="#">AK_initialize_new_index_segment()</a> | 226 |
| 5.21.2.3  | <a href="#">AK_initialize_new_segment()</a>       | 227 |
| 5.22      | <a href="#">file/filesearch.c File Reference</a>  | 227 |
| 5.22.1    | <a href="#">Detailed Description</a>              | 228 |
| 5.22.2    | <a href="#">Function Documentation</a>            | 228 |
| 5.22.2.1  | <a href="#">AK_deallocate_search_result()</a>     | 228 |
| 5.22.2.2  | <a href="#">Ak_filesearch_test()</a>              | 228 |
| 5.22.2.3  | <a href="#">AK_search_unsorted()</a>              | 229 |
| 5.23      | <a href="#">file/filesearch.h File Reference</a>  | 230 |
| 5.23.1    | <a href="#">Detailed Description</a>              | 231 |
| 5.23.2    | <a href="#">Function Documentation</a>            | 231 |
| 5.23.2.1  | <a href="#">AK_deallocate_search_result()</a>     | 231 |
| 5.23.2.2  | <a href="#">Ak_filesearch_test()</a>              | 232 |
| 5.23.2.3  | <a href="#">AK_search_unsorted()</a>              | 232 |
| 5.24      | <a href="#">file/filesort.h File Reference</a>    | 233 |
| 5.24.1    | <a href="#">Detailed Description</a>              | 234 |
| 5.24.2    | <a href="#">Function Documentation</a>            | 234 |
| 5.24.2.1  | <a href="#">AK_block_sort()</a>                   | 234 |

|           |  |     |
|-----------|--|-----|
| 5.24.2.2  | <a href="#">Ak_get_header_number()</a>           | 234 |
| 5.24.2.3  | <a href="#">Ak_get_num_of_tuples()</a>           | 235 |
| 5.24.2.4  | <a href="#">Ak_get_total_headers()</a>           | 235 |
| 5.25      | <a href="#">file/id.c File Reference</a>         | 235 |
| 5.25.1    | <a href="#">Detailed Description</a>             | 236 |
| 5.25.2    | <a href="#">Function Documentation</a>           | 236 |
| 5.25.2.1  | <a href="#">AK_get_id()</a>                      | 236 |
| 5.25.2.2  | <a href="#">AK_get_table_id()</a>                | 236 |
| 5.25.2.3  | <a href="#">Ak_id_test()</a>                     | 237 |
| 5.26      | <a href="#">file/id.h File Reference</a>         | 237 |
| 5.26.1    | <a href="#">Detailed Description</a>             | 238 |
| 5.26.2    | <a href="#">Function Documentation</a>           | 238 |
| 5.26.2.1  | <a href="#">AK_get_id()</a>                      | 238 |
| 5.26.2.2  | <a href="#">Ak_id_test()</a>                     | 238 |
| 5.27      | <a href="#">file/idx/bitmap.c File Reference</a> | 239 |
| 5.27.1    | <a href="#">Detailed Description</a>             | 240 |
| 5.27.2    | <a href="#">Function Documentation</a>           | 240 |
| 5.27.2.1  | <a href="#">AK_add_to_bitmap_index()</a>         | 240 |
| 5.27.2.2  | <a href="#">Ak_bitmap_test()</a>                 | 240 |
| 5.27.2.3  | <a href="#">Ak_create_Index()</a>                | 241 |
| 5.27.2.4  | <a href="#">AK_create_Index_Table()</a>          | 241 |
| 5.27.2.5  | <a href="#">AK_delete_bitmap_index()</a>         | 242 |
| 5.27.2.6  | <a href="#">Ak_get_Attribute()</a>               | 242 |
| 5.27.2.7  | <a href="#">AK_get_Attribute()</a>               | 243 |
| 5.27.2.8  | <a href="#">Ak_If_ExistOp()</a>                  | 243 |
| 5.27.2.9  | <a href="#">Ak_print_Att_Test()</a>              | 244 |
| 5.27.2.10 | <a href="#">Ak_print_Header_Test()</a>           | 244 |
| 5.27.2.11 | <a href="#">AK_update()</a>                      | 245 |
| 5.27.2.12 | <a href="#">Ak_write_block()</a>                 | 245 |
| 5.28      | <a href="#">file/idx/bitmap.h File Reference</a> | 246 |

|  |     |
|--|-----|
| 5.28.1 Detailed Description . . . . .          | 247 |
| 5.28.2 Function Documentation . . . . .        | 247 |
| 5.28.2.1 AK_add_to_bitmap_index() . . . . .    | 247 |
| 5.28.2.2 Ak_bitmap_test() . . . . .            | 248 |
| 5.28.2.3 Ak_create_Index() . . . . .           | 248 |
| 5.28.2.4 AK_create_Index_Table() . . . . .     | 249 |
| 5.28.2.5 AK_delete_bitmap_index() . . . . .    | 249 |
| 5.28.2.6 Ak_get_Attribute() . . . . .          | 250 |
| 5.28.2.7 AK_get_Attribute() . . . . .          | 250 |
| 5.28.2.8 Ak_If_ExistOp() . . . . .             | 251 |
| 5.28.2.9 Ak_print_Att_Test() . . . . .         | 251 |
| 5.28.2.10 Ak_print_Header_Test() . . . . .     | 251 |
| 5.28.2.11 AK_update() . . . . .                | 252 |
| 5.28.2.12 Ak_write_block() . . . . .           | 252 |
| 5.29 file/idx/btree.c File Reference . . . . . | 253 |
| 5.29.1 Detailed Description . . . . .          | 253 |
| 5.29.2 Function Documentation . . . . .        | 254 |
| 5.29.2.1 AK_btree_create() . . . . .           | 254 |
| 5.29.2.2 AK_btree_search_delete() . . . . .    | 254 |
| 5.30 file/idx/btree.h File Reference . . . . . | 255 |
| 5.30.1 Detailed Description . . . . .          | 256 |
| 5.30.2 Function Documentation . . . . .        | 256 |
| 5.30.2.1 AK_btree_create() . . . . .           | 256 |
| 5.30.2.2 AK_btree_search_delete() . . . . .    | 256 |
| 5.31 file/idx/hash.c File Reference . . . . .  | 257 |
| 5.31.1 Detailed Description . . . . .          | 258 |
| 5.31.2 Function Documentation . . . . .        | 258 |
| 5.31.2.1 AK_change_hash_info() . . . . .       | 258 |
| 5.31.2.2 AK_create_hash_index() . . . . .      | 259 |
| 5.31.2.3 AK_delete_in_hash_index() . . . . .   | 259 |

|           |   |     |
|-----------|---|-----|
| 5.31.2.4  | <a href="#">AK_elem_hash_value()</a>            | 260 |
| 5.31.2.5  | <a href="#">AK_find_delete_in_hash_index()</a>  | 260 |
| 5.31.2.6  | <a href="#">AK_find_in_hash_index()</a>         | 260 |
| 5.31.2.7  | <a href="#">AK_get_hash_info()</a>              | 261 |
| 5.31.2.8  | <a href="#">Ak_get_nth_main_bucket_add()</a>    | 261 |
| 5.31.2.9  | <a href="#">Ak_hash_test()</a>                  | 262 |
| 5.31.2.10 | <a href="#">Ak_insert_bucket_to_block()</a>     | 262 |
| 5.31.2.11 | <a href="#">AK_insert_in_hash_index()</a>       | 263 |
| 5.31.2.12 | <a href="#">Ak_update_bucket_in_block()</a>     | 263 |
| 5.32      | <a href="#">file/idx/hash.h File Reference</a>  | 264 |
| 5.32.1    | <a href="#">Detailed Description</a>            | 265 |
| 5.32.2    | <a href="#">Function Documentation</a>          | 265 |
| 5.32.2.1  | <a href="#">AK_change_hash_info()</a>           | 265 |
| 5.32.2.2  | <a href="#">AK_create_hash_index()</a>          | 266 |
| 5.32.2.3  | <a href="#">AK_delete_in_hash_index()</a>       | 266 |
| 5.32.2.4  | <a href="#">AK_elem_hash_value()</a>            | 267 |
| 5.32.2.5  | <a href="#">AK_find_delete_in_hash_index()</a>  | 267 |
| 5.32.2.6  | <a href="#">AK_find_in_hash_index()</a>         | 268 |
| 5.32.2.7  | <a href="#">AK_get_hash_info()</a>              | 268 |
| 5.32.2.8  | <a href="#">Ak_get_nth_main_bucket_add()</a>    | 269 |
| 5.32.2.9  | <a href="#">Ak_hash_test()</a>                  | 269 |
| 5.32.2.10 | <a href="#">Ak_insert_bucket_to_block()</a>     | 269 |
| 5.32.2.11 | <a href="#">AK_insert_in_hash_index()</a>       | 270 |
| 5.32.2.12 | <a href="#">Ak_update_bucket_in_block()</a>     | 270 |
| 5.33      | <a href="#">file/idx/index.c File Reference</a> | 271 |
| 5.33.1    | <a href="#">Detailed Description</a>            | 272 |
| 5.33.2    | <a href="#">Function Documentation</a>          | 272 |
| 5.33.2.1  | <a href="#">Ak_Delete_All_elementsAd()</a>      | 272 |
| 5.33.2.2  | <a href="#">Ak_Delete_elementAd()</a>           | 273 |
| 5.33.2.3  | <a href="#">Ak_Get_First_elementAd()</a>        | 273 |

|           |   |     |
|-----------|---|-----|
| 5.33.2.4  | <a href="#">AK_get_index_header()</a>           | 273 |
| 5.33.2.5  | <a href="#">AK_get_index_num_records()</a>      | 274 |
| 5.33.2.6  | <a href="#">AK_get_index_tuple()</a>            | 275 |
| 5.33.2.7  | <a href="#">Ak_Get_Last_elementAd()</a>         | 275 |
| 5.33.2.8  | <a href="#">Ak_Get_Next_elementAd()</a>         | 275 |
| 5.33.2.9  | <a href="#">Ak_Get_Position_Of_elementAd()</a>  | 276 |
| 5.33.2.10 | <a href="#">Ak_Get_Previous_elementAd()</a>     | 276 |
| 5.33.2.11 | <a href="#">AK_index_table_exist()</a>          | 277 |
| 5.33.2.12 | <a href="#">AK_index_test()</a>                 | 277 |
| 5.33.2.13 | <a href="#">Ak_InitializelistAd()</a>           | 278 |
| 5.33.2.14 | <a href="#">Ak_Insert_NewelementAd()</a>        | 278 |
| 5.33.2.15 | <a href="#">AK_num_index_attr()</a>             | 279 |
| 5.33.2.16 | <a href="#">AK_print_index_table()</a>          | 279 |
| 5.34      | <a href="#">file/idx/index.h File Reference</a> | 279 |
| 5.34.1    | <a href="#">Detailed Description</a>            | 281 |
| 5.34.2    | <a href="#">Function Documentation</a>          | 281 |
| 5.34.2.1  | <a href="#">Ak_Delete_All_elementsAd()</a>      | 281 |
| 5.34.2.2  | <a href="#">Ak_Delete_elementAd()</a>           | 282 |
| 5.34.2.3  | <a href="#">Ak_Get_First_elementAd()</a>        | 282 |
| 5.34.2.4  | <a href="#">AK_get_index_num_records()</a>      | 283 |
| 5.34.2.5  | <a href="#">AK_get_index_tuple()</a>            | 283 |
| 5.34.2.6  | <a href="#">Ak_Get_Last_elementAd()</a>         | 284 |
| 5.34.2.7  | <a href="#">Ak_Get_Next_elementAd()</a>         | 284 |
| 5.34.2.8  | <a href="#">Ak_Get_Position_Of_elementAd()</a>  | 285 |
| 5.34.2.9  | <a href="#">Ak_Get_Previous_elementAd()</a>     | 285 |
| 5.34.2.10 | <a href="#">AK_index_table_exist()</a>          | 286 |
| 5.34.2.11 | <a href="#">AK_index_test()</a>                 | 286 |
| 5.34.2.12 | <a href="#">Ak_InitializelistAd()</a>           | 286 |
| 5.34.2.13 | <a href="#">Ak_Insert_NewelementAd()</a>        | 287 |
| 5.34.2.14 | <a href="#">AK_num_index_attr()</a>             | 287 |

|                                      |     |
|--------------------------------------|-----|
| 5.34.2.15 AK_print_index_table()     | 288 |
| 5.35 file/sequence.c File Reference  | 288 |
| 5.35.1 Detailed Description          | 289 |
| 5.35.2 Function Documentation        | 289 |
| 5.35.2.1 AK_sequence_add()           | 289 |
| 5.35.2.2 AK_sequence_current_value() | 290 |
| 5.35.2.3 AK_sequence_get_id()        | 290 |
| 5.35.2.4 AK_sequence_modify()        | 291 |
| 5.35.2.5 AK_sequence_next_value()    | 291 |
| 5.35.2.6 AK_sequence_remove()        | 292 |
| 5.35.2.7 AK_sequence_rename()        | 292 |
| 5.35.2.8 AK_sequence_test()          | 293 |
| 5.36 file/sequence.h File Reference  | 293 |
| 5.36.1 Detailed Description          | 294 |
| 5.36.2 Function Documentation        | 294 |
| 5.36.2.1 AK_sequence_add()           | 294 |
| 5.36.2.2 AK_sequence_current_value() | 295 |
| 5.36.2.3 AK_sequence_get_id()        | 295 |
| 5.36.2.4 AK_sequence_modify()        | 296 |
| 5.36.2.5 AK_sequence_next_value()    | 296 |
| 5.36.2.6 AK_sequence_remove()        | 297 |
| 5.36.2.7 AK_sequence_rename()        | 297 |
| 5.36.2.8 AK_sequence_test()          | 298 |
| 5.37 file/table.c File Reference     | 298 |
| 5.37.1 Detailed Description          | 299 |
| 5.37.2 Function Documentation        | 299 |
| 5.37.2.1 AK_check_tables_scheme()    | 300 |
| 5.37.2.2 AK_get_attr_index()         | 300 |
| 5.37.2.3 AK_get_attr_name()          | 301 |
| 5.37.2.4 AK_get_column()             | 301 |



|           |   |     |
|-----------|---|-----|
| 5.37.2.5  | <a href="#">AK_get_header()</a>               | 301 |
| 5.37.2.6  | <a href="#">AK_get_num_records()</a>          | 302 |
| 5.37.2.7  | <a href="#">AK_get_row()</a>                  | 303 |
| 5.37.2.8  | <a href="#">AK_get_table_obj_id()</a>         | 303 |
| 5.37.2.9  | <a href="#">AK_get_tuple()</a>                | 304 |
| 5.37.2.10 | <a href="#">AK_num_attr()</a>                 | 304 |
| 5.37.2.11 | <a href="#">AK_op_rename_test()</a>           | 305 |
| 5.37.2.12 | <a href="#">AK_print_row()</a>                | 305 |
| 5.37.2.13 | <a href="#">AK_print_row_spacer()</a>         | 305 |
| 5.37.2.14 | <a href="#">AK_print_row_spacer_to_file()</a> | 306 |
| 5.37.2.15 | <a href="#">AK_print_row_to_file()</a>        | 306 |
| 5.37.2.16 | <a href="#">AK_print_table()</a>              | 307 |
| 5.37.2.17 | <a href="#">AK_print_table_to_file()</a>      | 307 |
| 5.37.2.18 | <a href="#">AK_rename()</a>                   | 308 |
| 5.37.2.19 | <a href="#">AK_table_empty()</a>              | 308 |
| 5.37.2.20 | <a href="#">AK_table_exist()</a>              | 309 |
| 5.37.2.21 | <a href="#">AK_table_test()</a>               | 309 |
| 5.37.2.22 | <a href="#">AK_tuple_to_string()</a>          | 309 |
| 5.37.2.23 | <a href="#">get_row_attr_data()</a>           | 310 |
| 5.38      | <a href="#">file/table.h File Reference</a>   | 310 |
| 5.38.1    | <a href="#">Detailed Description</a>          | 312 |
| 5.38.2    | <a href="#">Function Documentation</a>        | 313 |
| 5.38.2.1  | <a href="#">AK_check_tables_scheme()</a>      | 313 |
| 5.38.2.2  | <a href="#">AK_get_attr_index()</a>           | 313 |
| 5.38.2.3  | <a href="#">AK_get_attr_name()</a>            | 314 |
| 5.38.2.4  | <a href="#">AK_get_column()</a>               | 314 |
| 5.38.2.5  | <a href="#">AK_get_header()</a>               | 315 |
| 5.38.2.6  | <a href="#">AK_get_num_records()</a>          | 315 |
| 5.38.2.7  | <a href="#">AK_get_row()</a>                  | 316 |
| 5.38.2.8  | <a href="#">AK_get_table_obj_id()</a>         | 316 |

|           |  |     |
|-----------|--|-----|
| 5.38.2.9  | <a href="#">AK_get_tuple()</a>                 | 317 |
| 5.38.2.10 | <a href="#">AK_num_attr()</a>                  | 317 |
| 5.38.2.11 | <a href="#">AK_op_rename_test()</a>            | 318 |
| 5.38.2.12 | <a href="#">AK_print_row()</a>                 | 318 |
| 5.38.2.13 | <a href="#">AK_print_row_spacer()</a>          | 318 |
| 5.38.2.14 | <a href="#">AK_print_row_spacer_to_file()</a>  | 319 |
| 5.38.2.15 | <a href="#">AK_print_row_to_file()</a>         | 319 |
| 5.38.2.16 | <a href="#">AK_print_table()</a>               | 320 |
| 5.38.2.17 | <a href="#">AK_print_table_to_file()</a>       | 320 |
| 5.38.2.18 | <a href="#">AK_rename()</a>                    | 321 |
| 5.38.2.19 | <a href="#">AK_table_empty()</a>               | 321 |
| 5.38.2.20 | <a href="#">AK_table_test()</a>                | 322 |
| 5.38.2.21 | <a href="#">AK_tuple_to_string()</a>           | 322 |
| 5.38.2.22 | <a href="#">get_row_attr_data()</a>            | 322 |
| 5.39      | <a href="#">file/test.c File Reference</a>     | 323 |
| 5.39.1    | <a href="#">Detailed Description</a>           | 324 |
| 5.39.2    | <a href="#">Function Documentation</a>         | 324 |
| 5.39.2.1  | <a href="#">AK_create_test_tables()</a>        | 324 |
| 5.39.2.2  | <a href="#">AK_get_table_attribute_types()</a> | 324 |
| 5.39.2.3  | <a href="#">create_header_test()</a>           | 324 |
| 5.39.2.4  | <a href="#">get_column_test()</a>              | 325 |
| 5.39.2.5  | <a href="#">get_row_test()</a>                 | 325 |
| 5.39.2.6  | <a href="#">insert_data_test()</a>             | 326 |
| 5.39.2.7  | <a href="#">selection_test()</a>               | 327 |
| 5.40      | <a href="#">file/test.h File Reference</a>     | 327 |
| 5.40.1    | <a href="#">Detailed Description</a>           | 328 |
| 5.40.2    | <a href="#">Function Documentation</a>         | 328 |
| 5.40.2.1  | <a href="#">AK_create_test_tables()</a>        | 328 |
| 5.40.2.2  | <a href="#">AK_get_table_attribute_types()</a> | 329 |
| 5.40.2.3  | <a href="#">create_header_test()</a>           | 329 |

|           |  |     |
|-----------|--|-----|
| 5.40.2.4  | <a href="#">get_column_test()</a>                | 329 |
| 5.40.2.5  | <a href="#">get_row_test()</a>                   | 330 |
| 5.40.2.6  | <a href="#">insert_data_test()</a>               | 330 |
| 5.40.2.7  | <a href="#">selection_test()</a>                 | 331 |
| 5.41      | <a href="#">mm/memoman.c File Reference</a>      | 332 |
| 5.41.1    | <a href="#">Detailed Description</a>             | 333 |
| 5.41.2    | <a href="#">Function Documentation</a>           | 333 |
| 5.41.2.1  | <a href="#">AK_cache_AK_malloc()</a>             | 333 |
| 5.41.2.2  | <a href="#">AK_cache_block()</a>                 | 333 |
| 5.41.2.3  | <a href="#">AK_cache_result()</a>                | 334 |
| 5.41.2.4  | <a href="#">AK_find_AK_free_space()</a>          | 334 |
| 5.41.2.5  | <a href="#">AK_find_available_result_block()</a> | 335 |
| 5.41.2.6  | <a href="#">AK_flush_cache()</a>                 | 335 |
| 5.41.2.7  | <a href="#">AK_generate_result_id()</a>          | 335 |
| 5.41.2.8  | <a href="#">AK_get_block()</a>                   | 336 |
| 5.41.2.9  | <a href="#">AK_get_index_addresses()</a>         | 336 |
| 5.41.2.10 | <a href="#">AK_get_index_segment_addresses()</a> | 336 |
| 5.41.2.11 | <a href="#">AK_get_segment_addresses()</a>       | 337 |
| 5.41.2.12 | <a href="#">AK_get_table_addresses()</a>         | 337 |
| 5.41.2.13 | <a href="#">AK_init_new_extent()</a>             | 338 |
| 5.41.2.14 | <a href="#">AK_mem_block_modify()</a>            | 338 |
| 5.41.2.15 | <a href="#">AK_memoman_init()</a>                | 339 |
| 5.41.2.16 | <a href="#">AK_query_mem_AK_malloc()</a>         | 339 |
| 5.41.2.17 | <a href="#">AK_redo_log_AK_malloc()</a>          | 339 |
| 5.41.2.18 | <a href="#">AK_refresh_cache()</a>               | 340 |
| 5.42      | <a href="#">mm/memoman.h File Reference</a>      | 340 |
| 5.42.1    | <a href="#">Detailed Description</a>             | 342 |
| 5.42.2    | <a href="#">Function Documentation</a>           | 342 |
| 5.42.2.1  | <a href="#">AK_cache_AK_malloc()</a>             | 342 |
| 5.42.2.2  | <a href="#">AK_cache_block()</a>                 | 342 |

|           |  |     |
|-----------|--|-----|
| 5.42.2.3  | AK_cache_result()                        | 343 |
| 5.42.2.4  | AK_find_AK_free_space()                  | 343 |
| 5.42.2.5  | AK_find_available_result_block()         | 344 |
| 5.42.2.6  | AK_flush_cache()                         | 344 |
| 5.42.2.7  | AK_generate_result_id()                  | 344 |
| 5.42.2.8  | AK_get_block()                           | 345 |
| 5.42.2.9  | AK_get_index_addresses()                 | 345 |
| 5.42.2.10 | AK_get_index_segment_addresses()         | 345 |
| 5.42.2.11 | AK_get_segment_addresses()               | 346 |
| 5.42.2.12 | AK_get_table_addresses()                 | 346 |
| 5.42.2.13 | AK_init_new_extent()                     | 347 |
| 5.42.2.14 | AK_mem_block_modify()                    | 347 |
| 5.42.2.15 | AK_memoman_init()                        | 348 |
| 5.42.2.16 | AK_query_mem_AK_malloc()                 | 348 |
| 5.42.2.17 | AK_redo_log_AK_malloc()                  | 348 |
| 5.42.2.18 | AK_refresh_cache()                       | 349 |
| 5.43      | opti/query_optimization.c File Reference | 349 |
| 5.43.1    | Detailed Description                     | 349 |
| 5.43.2    | Function Documentation                   | 350 |
| 5.43.2.1  | AK_execute_rel_eq()                      | 350 |
| 5.43.2.2  | AK_print_optimized_query()               | 350 |
| 5.43.2.3  | AK_query_optimization()                  | 351 |
| 5.43.2.4  | AK_query_optimization_test()             | 351 |
| 5.44      | opti/query_optimization.h File Reference | 352 |
| 5.44.1    | Detailed Description                     | 353 |
| 5.44.2    | Function Documentation                   | 353 |
| 5.44.2.1  | AK_execute_rel_eq()                      | 353 |
| 5.44.2.2  | AK_print_optimized_query()               | 353 |
| 5.44.2.3  | AK_query_optimization()                  | 354 |
| 5.44.2.4  | AK_query_optimization_test()             | 354 |

|          |   |     |
|----------|---|-----|
| 5.45     | <a href="#">opti/rel_eq_assoc.c File Reference</a>      | 355 |
| 5.45.1   | <a href="#">Detailed Description</a>                    | 355 |
| 5.45.2   | <a href="#">Function Documentation</a>                  | 355 |
| 5.45.2.1 | <a href="#">AK_compare()</a>                            | 356 |
| 5.45.2.2 | <a href="#">AK_print_rel_eq_assoc()</a>                 | 356 |
| 5.45.2.3 | <a href="#">AK_rel_eq_assoc()</a>                       | 356 |
| 5.45.2.4 | <a href="#">AK_rel_eq_assoc_test()</a>                  | 357 |
| 5.46     | <a href="#">opti/rel_eq_assoc.h File Reference</a>      | 357 |
| 5.46.1   | <a href="#">Detailed Description</a>                    | 358 |
| 5.46.2   | <a href="#">Function Documentation</a>                  | 358 |
| 5.46.2.1 | <a href="#">AK_compare()</a>                            | 359 |
| 5.46.2.2 | <a href="#">AK_print_rel_eq_assoc()</a>                 | 359 |
| 5.46.2.3 | <a href="#">AK_rel_eq_assoc()</a>                       | 359 |
| 5.46.2.4 | <a href="#">AK_rel_eq_assoc_test()</a>                  | 360 |
| 5.47     | <a href="#">opti/rel_eq_comut.c File Reference</a>      | 360 |
| 5.47.1   | <a href="#">Detailed Description</a>                    | 361 |
| 5.47.2   | <a href="#">Function Documentation</a>                  | 361 |
| 5.47.2.1 | <a href="#">AK_print_rel_eq_comut()</a>                 | 361 |
| 5.47.2.2 | <a href="#">AK_rel_eq_commute_with_theta_join()</a>     | 361 |
| 5.47.2.3 | <a href="#">AK_rel_eq_comut()</a>                       | 362 |
| 5.47.2.4 | <a href="#">AK_rel_eq_comut_test()</a>                  | 362 |
| 5.48     | <a href="#">opti/rel_eq_comut.h File Reference</a>      | 363 |
| 5.48.1   | <a href="#">Detailed Description</a>                    | 364 |
| 5.48.2   | <a href="#">Function Documentation</a>                  | 364 |
| 5.48.2.1 | <a href="#">AK_print_rel_eq_comut()</a>                 | 364 |
| 5.48.2.2 | <a href="#">AK_rel_eq_commute_with_theta_join()</a>     | 364 |
| 5.48.2.3 | <a href="#">AK_rel_eq_comut()</a>                       | 365 |
| 5.48.2.4 | <a href="#">AK_rel_eq_comut_test()</a>                  | 365 |
| 5.49     | <a href="#">opti/rel_eq_projection.c File Reference</a> | 366 |
| 5.49.1   | <a href="#">Detailed Description</a>                    | 367 |

|          |   |     |
|----------|---|-----|
| 5.49.2   | Function Documentation                  | 367 |
| 5.49.2.1 | AK_print_rel_eq_projection()            | 367 |
| 5.49.2.2 | AK_rel_eq_can_commute()                 | 367 |
| 5.49.2.3 | AK_rel_eq_collect_cond_attributes()     | 368 |
| 5.49.2.4 | AK_rel_eq_get_attributes()              | 368 |
| 5.49.2.5 | AK_rel_eq_is_subset()                   | 369 |
| 5.49.2.6 | AK_rel_eq_projection()                  | 370 |
| 5.49.2.7 | AK_rel_eq_projection_attributes()       | 370 |
| 5.49.2.8 | AK_rel_eq_projection_test()             | 371 |
| 5.49.2.9 | AK_rel_eq_remove_duplicates()           | 371 |
| 5.50     | opti/rel_eq_projection.h File Reference | 371 |
| 5.50.1   | Detailed Description                    | 373 |
| 5.50.2   | Function Documentation                  | 373 |
| 5.50.2.1 | AK_print_rel_eq_projection()            | 373 |
| 5.50.2.2 | AK_rel_eq_can_commute()                 | 373 |
| 5.50.2.3 | AK_rel_eq_collect_cond_attributes()     | 374 |
| 5.50.2.4 | AK_rel_eq_get_attributes()              | 374 |
| 5.50.2.5 | AK_rel_eq_is_subset()                   | 375 |
| 5.50.2.6 | AK_rel_eq_projection()                  | 376 |
| 5.50.2.7 | AK_rel_eq_projection_attributes()       | 376 |
| 5.50.2.8 | AK_rel_eq_projection_test()             | 377 |
| 5.50.2.9 | AK_rel_eq_remove_duplicates()           | 377 |
| 5.51     | opti/rel_eq_selection.c File Reference  | 377 |
| 5.51.1   | Detailed Description                    | 378 |
| 5.51.2   | Function Documentation                  | 378 |
| 5.51.2.1 | AK_print_rel_eq_selection()             | 379 |
| 5.51.2.2 | AK_rel_eq_cond_attributes()             | 379 |
| 5.51.2.3 | AK_rel_eq_get_attrributes_char()        | 379 |
| 5.51.2.4 | AK_rel_eq_is_attr_subset()              | 380 |
| 5.51.2.5 | AK_rel_eq_selection()                   | 381 |

|          |  |     |
|----------|--|-----|
| 5.51.2.6 | <a href="#">AK_rel_eq_selection_test()</a>             | 381 |
| 5.51.2.7 | <a href="#">AK_rel_eq_share_attributes()</a>           | 382 |
| 5.51.2.8 | <a href="#">AK_rel_eq_split_condition()</a>            | 382 |
| 5.52     | <a href="#">opti/rel_eq_selection.h File Reference</a> | 383 |
| 5.52.1   | <a href="#">Detailed Description</a>                   | 384 |
| 5.52.2   | <a href="#">Function Documentation</a>                 | 384 |
| 5.52.2.1 | <a href="#">AK_print_rel_eq_selection()</a>            | 384 |
| 5.52.2.2 | <a href="#">AK_rel_eq_cond_attributes()</a>            | 385 |
| 5.52.2.3 | <a href="#">AK_rel_eq_get_attributes_char()</a>        | 385 |
| 5.52.2.4 | <a href="#">AK_rel_eq_is_attr_subset()</a>             | 386 |
| 5.52.2.5 | <a href="#">AK_rel_eq_selection()</a>                  | 386 |
| 5.52.2.6 | <a href="#">AK_rel_eq_selection_test()</a>             | 387 |
| 5.52.2.7 | <a href="#">AK_rel_eq_share_attributes()</a>           | 387 |
| 5.52.2.8 | <a href="#">AK_rel_eq_split_condition()</a>            | 388 |
| 5.53     | <a href="#">rec/archive_log.h File Reference</a>       | 389 |
| 5.53.1   | <a href="#">Detailed Description</a>                   | 390 |
| 5.53.2   | <a href="#">Function Documentation</a>                 | 390 |
| 5.53.2.1 | <a href="#">AK_archive_log()</a>                       | 390 |
| 5.53.2.2 | <a href="#">AK_empty_archive_log()</a>                 | 390 |
| 5.53.2.3 | <a href="#">AK_get_timestamp()</a>                     | 391 |
| 5.54     | <a href="#">rec/recovery.c File Reference</a>          | 391 |
| 5.54.1   | <a href="#">Detailed Description</a>                   | 392 |
| 5.54.2   | <a href="#">Function Documentation</a>                 | 392 |
| 5.54.2.1 | <a href="#">AK_recover_archive_log()</a>               | 392 |
| 5.54.2.2 | <a href="#">AK_recover_operation()</a>                 | 392 |
| 5.54.2.3 | <a href="#">AK_recovery_insert_row()</a>               | 393 |
| 5.54.2.4 | <a href="#">AK_recovery_test()</a>                     | 393 |
| 5.54.2.5 | <a href="#">AK_recovery_tokenize()</a>                 | 394 |
| 5.54.3   | <a href="#">Variable Documentation</a>                 | 394 |
| 5.54.3.1 | <a href="#">grandfailure</a>                           | 394 |

|          |                                  |     |
|----------|----------------------------------|-----|
| 5.55     | rec/redo_log.c File Reference    | 394 |
| 5.55.1   | Detailed Description             | 395 |
| 5.55.2   | Function Documentation           | 395 |
| 5.55.2.1 | AK_add_to_redolog()              | 395 |
| 5.55.2.2 | AK_check_attributes()            | 396 |
| 5.55.2.3 | AK_printout_redolog()            | 396 |
| 5.56     | rel/aggregation.c File Reference | 396 |
| 5.56.1   | Detailed Description             | 397 |
| 5.56.2   | Function Documentation           | 397 |
| 5.56.2.1 | AK_agg_input_add()               | 397 |
| 5.56.2.2 | AK_agg_input_add_to_beginning()  | 398 |
| 5.56.2.3 | AK_agg_input_fix()               | 398 |
| 5.56.2.4 | AK_agg_input_init()              | 399 |
| 5.56.2.5 | AK_aggregation()                 | 399 |
| 5.56.2.6 | Ak_aggregation_test()            | 400 |
| 5.56.2.7 | AK_header_size()                 | 400 |
| 5.56.2.8 | AK_search_unsorted()             | 401 |
| 5.57     | rel/aggregation.h File Reference | 402 |
| 5.57.1   | Detailed Description             | 403 |
| 5.57.2   | Function Documentation           | 403 |
| 5.57.2.1 | AK_agg_input_add()               | 403 |
| 5.57.2.2 | AK_agg_input_add_to_beginning()  | 404 |
| 5.57.2.3 | AK_agg_input_fix()               | 404 |
| 5.57.2.4 | AK_agg_input_init()              | 405 |
| 5.57.2.5 | AK_aggregation()                 | 405 |
| 5.57.2.6 | Ak_aggregation_test()            | 406 |
| 5.57.2.7 | AK_header_size()                 | 406 |
| 5.58     | rel/difference.c File Reference  | 407 |
| 5.58.1   | Detailed Description             | 407 |
| 5.58.2   | Function Documentation           | 407 |



|          |  |     |
|----------|--|-----|
| 5.58.2.1 | <a href="#">AK_difference()</a>                        | 408 |
| 5.58.2.2 | <a href="#">Ak_op_difference_test()</a>                | 408 |
| 5.59     | <a href="#">rel/difference.h File Reference</a>        | 408 |
| 5.59.1   | <a href="#">Detailed Description</a>                   | 409 |
| 5.59.2   | <a href="#">Function Documentation</a>                 | 409 |
| 5.59.2.1 | <a href="#">AK_difference()</a>                        | 410 |
| 5.59.2.2 | <a href="#">Ak_op_difference_test()</a>                | 410 |
| 5.60     | <a href="#">rel/expression_check.c File Reference</a>  | 410 |
| 5.60.1   | <a href="#">Detailed Description</a>                   | 411 |
| 5.60.2   | <a href="#">Function Documentation</a>                 | 411 |
| 5.60.2.1 | <a href="#">AK_check_arithmetic_statement()</a>        | 412 |
| 5.60.2.2 | <a href="#">AK_check_if_row_satisfies_expression()</a> | 412 |
| 5.60.2.3 | <a href="#">Ak_check_regex_expression()</a>            | 413 |
| 5.60.2.4 | <a href="#">Ak_check_regex_operator_expression()</a>   | 413 |
| 5.60.2.5 | <a href="#">AK_replace_wild_card()</a>                 | 414 |
| 5.61     | <a href="#">rel/expression_check.h File Reference</a>  | 414 |
| 5.61.1   | <a href="#">Detailed Description</a>                   | 415 |
| 5.61.2   | <a href="#">Function Documentation</a>                 | 415 |
| 5.61.2.1 | <a href="#">AK_check_arithmetic_statement()</a>        | 415 |
| 5.61.2.2 | <a href="#">AK_check_if_row_satisfies_expression()</a> | 416 |
| 5.61.2.3 | <a href="#">Ak_check_regex_expression()</a>            | 416 |
| 5.61.2.4 | <a href="#">Ak_check_regex_operator_expression()</a>   | 417 |
| 5.62     | <a href="#">rel/intersect.c File Reference</a>         | 417 |
| 5.62.1   | <a href="#">Detailed Description</a>                   | 418 |
| 5.62.2   | <a href="#">Function Documentation</a>                 | 418 |
| 5.62.2.1 | <a href="#">AK_intersect()</a>                         | 418 |
| 5.62.2.2 | <a href="#">Ak_op_intersect_test()</a>                 | 418 |
| 5.63     | <a href="#">rel/intersect.h File Reference</a>         | 419 |
| 5.63.1   | <a href="#">Detailed Description</a>                   | 420 |
| 5.63.2   | <a href="#">Function Documentation</a>                 | 420 |

|          |   |     |
|----------|---|-----|
| 5.63.2.1 | <a href="#">AK_intersect()</a> . . . . .                  | 420 |
| 5.63.2.2 | <a href="#">Ak_op_intersect_test()</a> . . . . .          | 420 |
| 5.64     | <a href="#">rel/nat_join.c File Reference</a> . . . . .   | 421 |
| 5.64.1   | <a href="#">Detailed Description</a> . . . . .            | 421 |
| 5.64.2   | <a href="#">Function Documentation</a> . . . . .          | 422 |
| 5.64.2.1 | <a href="#">AK_copy_blocks_join()</a> . . . . .           | 422 |
| 5.64.2.2 | <a href="#">AK_create_join_block_header()</a> . . . . .   | 422 |
| 5.64.2.3 | <a href="#">AK_join()</a> . . . . .                       | 423 |
| 5.64.2.4 | <a href="#">AK_merge_block_join()</a> . . . . .           | 423 |
| 5.64.2.5 | <a href="#">AK_op_join_test()</a> . . . . .               | 424 |
| 5.65     | <a href="#">rel/nat_join.h File Reference</a> . . . . .   | 424 |
| 5.65.1   | <a href="#">Detailed Description</a> . . . . .            | 425 |
| 5.65.2   | <a href="#">Function Documentation</a> . . . . .          | 425 |
| 5.65.2.1 | <a href="#">AK_copy_blocks_join()</a> . . . . .           | 426 |
| 5.65.2.2 | <a href="#">AK_create_join_block_header()</a> . . . . .   | 426 |
| 5.65.2.3 | <a href="#">AK_join()</a> . . . . .                       | 427 |
| 5.65.2.4 | <a href="#">AK_merge_block_join()</a> . . . . .           | 427 |
| 5.65.2.5 | <a href="#">AK_op_join_test()</a> . . . . .               | 428 |
| 5.66     | <a href="#">rel/product.c File Reference</a> . . . . .    | 428 |
| 5.66.1   | <a href="#">Detailed Description</a> . . . . .            | 429 |
| 5.66.2   | <a href="#">Function Documentation</a> . . . . .          | 429 |
| 5.66.2.1 | <a href="#">AK_op_product_test()</a> . . . . .            | 429 |
| 5.66.2.2 | <a href="#">AK_product()</a> . . . . .                    | 429 |
| 5.67     | <a href="#">rel/product.h File Reference</a> . . . . .    | 430 |
| 5.67.1   | <a href="#">Detailed Description</a> . . . . .            | 431 |
| 5.67.2   | <a href="#">Function Documentation</a> . . . . .          | 431 |
| 5.67.2.1 | <a href="#">AK_op_product_test()</a> . . . . .            | 431 |
| 5.67.2.2 | <a href="#">AK_product()</a> . . . . .                    | 431 |
| 5.68     | <a href="#">rel/projection.c File Reference</a> . . . . . | 432 |
| 5.68.1   | <a href="#">Detailed Description</a> . . . . .            | 433 |

|           |                                 |     |
|-----------|---------------------------------|-----|
| 5.68.2    | Function Documentation          | 433 |
| 5.68.2.1  | AK_copy_block_projection()      | 433 |
| 5.68.2.2  | AK_create_block_header()        | 433 |
| 5.68.2.3  | AK_create_header_name()         | 434 |
| 5.68.2.4  | AK_determine_header_type()      | 434 |
| 5.68.2.5  | AK_get_operator()               | 435 |
| 5.68.2.6  | AK_op_projection_test()         | 435 |
| 5.68.2.7  | AK_perform_operatrion()         | 436 |
| 5.68.2.8  | AK_projection()                 | 437 |
| 5.68.2.9  | AK_temp_create_table()          | 437 |
| 5.68.2.10 | removeSubstring()               | 438 |
| 5.69      | rel/projection.h File Reference | 438 |
| 5.69.1    | Detailed Description            | 440 |
| 5.69.2    | Function Documentation          | 440 |
| 5.69.2.1  | AK_copy_block_projection()      | 440 |
| 5.69.2.2  | AK_create_block_header()        | 440 |
| 5.69.2.3  | AK_create_header_name()         | 441 |
| 5.69.2.4  | AK_determine_header_type()      | 441 |
| 5.69.2.5  | AK_get_operator()               | 443 |
| 5.69.2.6  | AK_op_projection_test()         | 443 |
| 5.69.2.7  | AK_perform_operatrion()         | 444 |
| 5.69.2.8  | AK_projection()                 | 445 |
| 5.69.2.9  | AK_temp_create_table()          | 445 |
| 5.69.2.10 | removeSubstring()               | 446 |
| 5.70      | rel/selection.c File Reference  | 446 |
| 5.70.1    | Detailed Description            | 447 |
| 5.70.2    | Function Documentation          | 447 |
| 5.70.2.1  | AK_op_selection_test()          | 447 |
| 5.70.2.2  | AK_op_selection_test_pattern()  | 448 |
| 5.70.2.3  | AK_op_selection_test_redolog()  | 448 |

|          |   |     |
|----------|---|-----|
| 5.70.2.4 | <a href="#">AK_selection()</a>                  | 448 |
| 5.71     | <a href="#">rel/selection.h File Reference</a>  | 449 |
| 5.71.1   | <a href="#">Detailed Description</a>            | 449 |
| 5.71.2   | <a href="#">Function Documentation</a>          | 450 |
| 5.71.2.1 | <a href="#">AK_op_selection_test()</a>          | 450 |
| 5.71.2.2 | <a href="#">AK_op_selection_test_pattern()</a>  | 450 |
| 5.71.2.3 | <a href="#">AK_op_selection_test_redolog()</a>  | 450 |
| 5.71.2.4 | <a href="#">AK_selection()</a>                  | 450 |
| 5.72     | <a href="#">rel/theta_join.c File Reference</a> | 451 |
| 5.72.1   | <a href="#">Detailed Description</a>            | 451 |
| 5.72.2   | <a href="#">Function Documentation</a>          | 452 |
| 5.72.2.1 | <a href="#">AK_check_constraints()</a>          | 452 |
| 5.72.2.2 | <a href="#">AK_create_theta_join_header()</a>   | 452 |
| 5.72.2.3 | <a href="#">AK_op_theta_join_test()</a>         | 453 |
| 5.72.2.4 | <a href="#">AK_theta_join()</a>                 | 453 |
| 5.73     | <a href="#">rel/theta_join.h File Reference</a> | 454 |
| 5.73.1   | <a href="#">Detailed Description</a>            | 455 |
| 5.73.2   | <a href="#">Function Documentation</a>          | 455 |
| 5.73.2.1 | <a href="#">AK_check_constraints()</a>          | 455 |
| 5.73.2.2 | <a href="#">AK_create_theta_join_header()</a>   | 456 |
| 5.73.2.3 | <a href="#">AK_op_theta_join_test()</a>         | 456 |
| 5.73.2.4 | <a href="#">AK_theta_join()</a>                 | 456 |
| 5.74     | <a href="#">rel/union.c File Reference</a>      | 457 |
| 5.74.1   | <a href="#">Detailed Description</a>            | 457 |
| 5.74.2   | <a href="#">Function Documentation</a>          | 457 |
| 5.74.2.1 | <a href="#">AK_op_union_test()</a>              | 458 |
| 5.74.2.2 | <a href="#">AK_union()</a>                      | 458 |
| 5.75     | <a href="#">rel/union.h File Reference</a>      | 458 |
| 5.75.1   | <a href="#">Detailed Description</a>            | 459 |
| 5.75.2   | <a href="#">Function Documentation</a>          | 459 |

|          |  |     |
|----------|--|-----|
| 5.75.2.1 | <a href="#">AK_op_union_test()</a>                       | 460 |
| 5.75.2.2 | <a href="#">AK_union()</a>                               | 460 |
| 5.76     | <a href="#">sql/cs/between.c File Reference</a>          | 460 |
| 5.76.1   | <a href="#">Detailed Description</a>                     | 461 |
| 5.76.2   | <a href="#">Function Documentation</a>                   | 461 |
| 5.76.2.1 | <a href="#">Ak_constraint_between_test()</a>             | 462 |
| 5.76.2.2 | <a href="#">AK_delete_constraint_between()</a>           | 462 |
| 5.76.2.3 | <a href="#">AK_find_table_address()</a>                  | 462 |
| 5.76.2.4 | <a href="#">AK_print_constraints()</a>                   | 463 |
| 5.76.2.5 | <a href="#">AK_read_constraint_between()</a>             | 463 |
| 5.76.2.6 | <a href="#">AK_set_constraint_between()</a>              | 464 |
| 5.77     | <a href="#">sql/cs/between.h File Reference</a>          | 464 |
| 5.77.1   | <a href="#">Detailed Description</a>                     | 465 |
| 5.77.2   | <a href="#">Function Documentation</a>                   | 466 |
| 5.77.2.1 | <a href="#">Ak_constraint_between_test()</a>             | 466 |
| 5.77.2.2 | <a href="#">AK_find_table_address()</a>                  | 466 |
| 5.77.2.3 | <a href="#">AK_read_constraint_between()</a>             | 466 |
| 5.77.2.4 | <a href="#">AK_set_constraint_between()</a>              | 467 |
| 5.78     | <a href="#">sql/cs/check_constraint.c File Reference</a> | 467 |
| 5.78.1   | <a href="#">Detailed Description</a>                     | 468 |
| 5.78.2   | <a href="#">Function Documentation</a>                   | 468 |
| 5.78.2.1 | <a href="#">AK_check_constraint()</a>                    | 468 |
| 5.78.2.2 | <a href="#">AK_check_constraint_test()</a>               | 469 |
| 5.78.2.3 | <a href="#">AK_set_check_constraint()</a>                | 469 |
| 5.78.2.4 | <a href="#">condition_passed()</a>                       | 470 |
| 5.79     | <a href="#">sql/cs/check_constraint.h File Reference</a> | 470 |
| 5.79.1   | <a href="#">Detailed Description</a>                     | 471 |
| 5.79.2   | <a href="#">Function Documentation</a>                   | 472 |
| 5.79.2.1 | <a href="#">AK_check_constraint()</a>                    | 472 |
| 5.79.2.2 | <a href="#">AK_check_constraint_test()</a>               | 472 |

|          |  |     |
|----------|--|-----|
| 5.79.2.3 | <a href="#">AK_set_check_constraint()</a>                | 473 |
| 5.79.2.4 | <a href="#">condition_passed()</a>                       | 473 |
| 5.80     | <a href="#">sql/cs/constraint_names.c File Reference</a> | 474 |
| 5.80.1   | <a href="#">Detailed Description</a>                     | 474 |
| 5.80.2   | <a href="#">Function Documentation</a>                   | 474 |
| 5.80.2.1 | <a href="#">Ak_check_constraint_name()</a>               | 474 |
| 5.80.2.2 | <a href="#">AK_constraint_names_test()</a>               | 475 |
| 5.81     | <a href="#">sql/cs/constraint_names.h File Reference</a> | 475 |
| 5.81.1   | <a href="#">Detailed Description</a>                     | 476 |
| 5.81.2   | <a href="#">Function Documentation</a>                   | 476 |
| 5.81.2.1 | <a href="#">Ak_check_constraint_name()</a>               | 476 |
| 5.81.2.2 | <a href="#">AK_constraint_names_test()</a>               | 477 |
| 5.82     | <a href="#">sql/cs/nnnull.c File Reference</a>           | 477 |
| 5.82.1   | <a href="#">Detailed Description</a>                     | 478 |
| 5.82.2   | <a href="#">Function Documentation</a>                   | 478 |
| 5.82.2.1 | <a href="#">AK_delete_constraint_not_null()</a>          | 478 |
| 5.82.2.2 | <a href="#">AK_null_test()</a>                           | 478 |
| 5.82.2.3 | <a href="#">AK_read_constraint_not_null()</a>            | 479 |
| 5.82.2.4 | <a href="#">AK_set_constraint_not_null()</a>             | 479 |
| 5.83     | <a href="#">sql/cs/nnnull.h File Reference</a>           | 480 |
| 5.83.1   | <a href="#">Detailed Description</a>                     | 481 |
| 5.83.2   | <a href="#">Function Documentation</a>                   | 481 |
| 5.83.2.1 | <a href="#">AK_delete_constraint_not_null()</a>          | 481 |
| 5.83.2.2 | <a href="#">AK_null_test()</a>                           | 481 |
| 5.83.2.3 | <a href="#">AK_read_constraint_not_null()</a>            | 482 |
| 5.83.2.4 | <a href="#">AK_set_constraint_not_null()</a>             | 482 |
| 5.84     | <a href="#">sql/cs/reference.c File Reference</a>        | 483 |
| 5.84.1   | <a href="#">Detailed Description</a>                     | 483 |
| 5.84.2   | <a href="#">Function Documentation</a>                   | 484 |
| 5.84.2.1 | <a href="#">AK_add_reference()</a>                       | 484 |

|           |   |     |
|-----------|---|-----|
| 5.84.2.2  | <a href="#">AK_get_reference()</a>                    | 484 |
| 5.84.2.3  | <a href="#">AK_reference_check_attribute()</a>        | 485 |
| 5.84.2.4  | <a href="#">AK_reference_check_entry()</a>            | 485 |
| 5.84.2.5  | <a href="#">AK_reference_check_if_update_needed()</a> | 486 |
| 5.84.2.6  | <a href="#">AK_reference_check_restricion()</a>       | 486 |
| 5.84.2.7  | <a href="#">AK_reference_test()</a>                   | 487 |
| 5.84.2.8  | <a href="#">AK_reference_update()</a>                 | 487 |
| 5.85      | <a href="#">sql/cs/reference.h File Reference</a>     | 488 |
| 5.85.1    | <a href="#">Detailed Description</a>                  | 490 |
| 5.85.2    | <a href="#">Macro Definition Documentation</a>        | 490 |
| 5.85.2.1  | <a href="#">REF_TYPE_NO_ACTION</a>                    | 490 |
| 5.85.3    | <a href="#">Function Documentation</a>                | 490 |
| 5.85.3.1  | <a href="#">AK_add_reference()</a>                    | 490 |
| 5.85.3.2  | <a href="#">Ak_delete_row()</a>                       | 491 |
| 5.85.3.3  | <a href="#">AK_get_reference()</a>                    | 491 |
| 5.85.3.4  | <a href="#">AK_initialize_new_segment()</a>           | 491 |
| 5.85.3.5  | <a href="#">Ak_Insert_New_Element()</a>               | 492 |
| 5.85.3.6  | <a href="#">Ak_Insert_New_Element_For_Update()</a>    | 493 |
| 5.85.3.7  | <a href="#">Ak_insert_row()</a>                       | 493 |
| 5.85.3.8  | <a href="#">AK_reference_check_attribute()</a>        | 494 |
| 5.85.3.9  | <a href="#">AK_reference_check_entry()</a>            | 494 |
| 5.85.3.10 | <a href="#">AK_reference_check_if_update_needed()</a> | 495 |
| 5.85.3.11 | <a href="#">AK_reference_check_restricion()</a>       | 495 |
| 5.85.3.12 | <a href="#">AK_reference_test()</a>                   | 496 |
| 5.85.3.13 | <a href="#">AK_reference_update()</a>                 | 496 |
| 5.85.3.14 | <a href="#">AK_selection()</a>                        | 497 |
| 5.85.3.15 | <a href="#">Ak_update_row()</a>                       | 497 |
| 5.86      | <a href="#">sql/cs/unique.c File Reference</a>        | 497 |
| 5.86.1    | <a href="#">Detailed Description</a>                  | 498 |
| 5.86.2    | <a href="#">Function Documentation</a>                | 498 |

|          |  |     |
|----------|--|-----|
| 5.86.2.1 | <a href="#">AK_delete_constraint_unique()</a>      | 498 |
| 5.86.2.2 | <a href="#">AK_read_constraint_unique()</a>        | 499 |
| 5.86.2.3 | <a href="#">Ak_set_constraint_unique()</a>         | 499 |
| 5.86.2.4 | <a href="#">AK_unique_test()</a>                   | 500 |
| 5.87     | <a href="#">sql/cs/unique.h File Reference</a>     | 500 |
| 5.87.1   | <a href="#">Detailed Description</a>               | 501 |
| 5.87.2   | <a href="#">Function Documentation</a>             | 501 |
| 5.87.2.1 | <a href="#">AK_delete_constraint_unique()</a>      | 501 |
| 5.87.2.2 | <a href="#">AK_read_constraint_unique()</a>        | 502 |
| 5.87.2.3 | <a href="#">Ak_set_constraint_unique()</a>         | 502 |
| 5.87.2.4 | <a href="#">AK_unique_test()</a>                   | 503 |
| 5.88     | <a href="#">sql/drop.c File Reference</a>          | 503 |
| 5.88.1   | <a href="#">Detailed Description</a>               | 504 |
| 5.88.2   | <a href="#">Function Documentation</a>             | 504 |
| 5.88.2.1 | <a href="#">AK_drop()</a>                          | 504 |
| 5.88.2.2 | <a href="#">AK_drop_help_function()</a>            | 505 |
| 5.88.2.3 | <a href="#">AK_drop_test()</a>                     | 505 |
| 5.88.2.4 | <a href="#">AK_if_exist()</a>                      | 505 |
| 5.88.3   | <a href="#">Variable Documentation</a>             | 506 |
| 5.88.3.1 | <a href="#">system_catalog</a>                     | 506 |
| 5.89     | <a href="#">sql/drop.h File Reference</a>          | 506 |
| 5.89.1   | <a href="#">Function Documentation</a>             | 507 |
| 5.89.1.1 | <a href="#">AK_drop()</a>                          | 508 |
| 5.89.1.2 | <a href="#">AK_drop_test()</a>                     | 508 |
| 5.89.1.3 | <a href="#">AK_if_exist()</a>                      | 508 |
| 5.90     | <a href="#">sql/function.c File Reference</a>      | 509 |
| 5.90.1   | <a href="#">Detailed Description</a>               | 510 |
| 5.90.2   | <a href="#">Function Documentation</a>             | 510 |
| 5.90.2.1 | <a href="#">AK_check_function_arguments()</a>      | 510 |
| 5.90.2.2 | <a href="#">AK_check_function_arguments_type()</a> | 510 |



|           |  |     |
|-----------|--|-----|
| 5.90.2.3  | <a href="#">AK_function_add()</a>                        | 511 |
| 5.90.2.4  | <a href="#">AK_function_arguments_add()</a>              | 511 |
| 5.90.2.5  | <a href="#">AK_function_arguments_remove_by_obj_id()</a> | 512 |
| 5.90.2.6  | <a href="#">AK_function_change_return_type()</a>         | 512 |
| 5.90.2.7  | <a href="#">AK_function_remove_by_name()</a>             | 513 |
| 5.90.2.8  | <a href="#">AK_function_remove_by_obj_id()</a>           | 513 |
| 5.90.2.9  | <a href="#">AK_function_rename()</a>                     | 514 |
| 5.90.2.10 | <a href="#">AK_function_test()</a>                       | 514 |
| 5.90.2.11 | <a href="#">AK_get_function_obj_id()</a>                 | 514 |
| 5.91      | <a href="#">sql/function.h File Reference</a>            | 515 |
| 5.91.1    | <a href="#">Detailed Description</a>                     | 516 |
| 5.91.2    | <a href="#">Function Documentation</a>                   | 516 |
| 5.91.2.1  | <a href="#">AK_check_function_arguments()</a>            | 516 |
| 5.91.2.2  | <a href="#">AK_check_function_arguments_type()</a>       | 517 |
| 5.91.2.3  | <a href="#">AK_function_add()</a>                        | 517 |
| 5.91.2.4  | <a href="#">AK_function_arguments_add()</a>              | 518 |
| 5.91.2.5  | <a href="#">AK_function_arguments_remove_by_obj_id()</a> | 518 |
| 5.91.2.6  | <a href="#">AK_function_change_return_type()</a>         | 519 |
| 5.91.2.7  | <a href="#">AK_function_remove_by_name()</a>             | 519 |
| 5.91.2.8  | <a href="#">AK_function_remove_by_obj_id()</a>           | 520 |
| 5.91.2.9  | <a href="#">AK_function_rename()</a>                     | 520 |
| 5.91.2.10 | <a href="#">AK_function_test()</a>                       | 521 |
| 5.91.2.11 | <a href="#">AK_get_function_obj_id()</a>                 | 521 |
| 5.92      | <a href="#">sql/privileges.c File Reference</a>          | 521 |
| 5.92.1    | <a href="#">Detailed Description</a>                     | 523 |
| 5.92.2    | <a href="#">Function Documentation</a>                   | 523 |
| 5.92.2.1  | <a href="#">AK_add_user_to_group()</a>                   | 523 |
| 5.92.2.2  | <a href="#">AK_check_group_privilege()</a>               | 523 |
| 5.92.2.3  | <a href="#">AK_check_privilege()</a>                     | 524 |
| 5.92.2.4  | <a href="#">AK_check_user_privilege()</a>                | 524 |

|           |                                  |     |
|-----------|----------------------------------|-----|
| 5.92.2.5  | AK_grant_privilege_group()       | 525 |
| 5.92.2.6  | AK_grant_privilege_user()        | 525 |
| 5.92.2.7  | AK_group_add()                   | 526 |
| 5.92.2.8  | AK_group_get_id()                | 526 |
| 5.92.2.9  | AK_group_remove_by_name()        | 527 |
| 5.92.2.10 | AK_group_rename()                | 527 |
| 5.92.2.11 | AK_privileges_test()             | 528 |
| 5.92.2.12 | AK_remove_all_users_from_group() | 528 |
| 5.92.2.13 | AK_remove_user_from_all_groups() | 528 |
| 5.92.2.14 | AK_revoke_all_privileges_group() | 529 |
| 5.92.2.15 | AK_revoke_all_privileges_user()  | 529 |
| 5.92.2.16 | AK_revoke_privilege_group()      | 530 |
| 5.92.2.17 | AK_revoke_privilege_user()       | 530 |
| 5.92.2.18 | AK_user_add()                    | 531 |
| 5.92.2.19 | AK_user_get_id()                 | 531 |
| 5.92.2.20 | AK_user_remove_by_name()         | 532 |
| 5.92.2.21 | AK_user_rename()                 | 532 |
| 5.93      | sql/select.c File Reference      | 533 |
| 5.93.1    | Detailed Description             | 533 |
| 5.93.2    | Function Documentation           | 533 |
| 5.93.2.1  | AK_select()                      | 533 |
| 5.93.2.2  | AK_select_test()                 | 534 |
| 5.94      | sql/trigger.c File Reference     | 534 |
| 5.94.1    | Detailed Description             | 535 |
| 5.94.2    | Function Documentation           | 535 |
| 5.94.2.1  | AK_trigger_add()                 | 535 |
| 5.94.2.2  | AK_trigger_edit()                | 536 |
| 5.94.2.3  | AK_trigger_get_conditions()      | 536 |
| 5.94.2.4  | AK_trigger_get_id()              | 537 |
| 5.94.2.5  | AK_trigger_remove_by_name()      | 537 |

|          |  |     |
|----------|--|-----|
| 5.94.2.6 | <a href="#">AK_trigger_remove_by_obj_id()</a>    | 538 |
| 5.94.2.7 | <a href="#">AK_trigger_rename()</a>              | 538 |
| 5.94.2.8 | <a href="#">AK_trigger_save_conditions()</a>     | 539 |
| 5.94.2.9 | <a href="#">AK_trigger_test()</a>                | 539 |
| 5.95     | <a href="#">sql/trigger.h File Reference</a>     | 539 |
| 5.95.1   | <a href="#">Detailed Description</a>             | 541 |
| 5.95.2   | <a href="#">Function Documentation</a>           | 541 |
| 5.95.2.1 | <a href="#">AK_trigger_add()</a>                 | 541 |
| 5.95.2.2 | <a href="#">AK_trigger_edit()</a>                | 541 |
| 5.95.2.3 | <a href="#">AK_trigger_get_conditions()</a>      | 542 |
| 5.95.2.4 | <a href="#">AK_trigger_get_id()</a>              | 542 |
| 5.95.2.5 | <a href="#">AK_trigger_remove_by_name()</a>      | 543 |
| 5.95.2.6 | <a href="#">AK_trigger_remove_by_obj_id()</a>    | 543 |
| 5.95.2.7 | <a href="#">AK_trigger_rename()</a>              | 544 |
| 5.95.2.8 | <a href="#">AK_trigger_save_conditions()</a>     | 544 |
| 5.95.2.9 | <a href="#">AK_trigger_test()</a>                | 545 |
| 5.96     | <a href="#">sql/view.c File Reference</a>        | 545 |
| 5.96.1   | <a href="#">Detailed Description</a>             | 546 |
| 5.96.2   | <a href="#">Function Documentation</a>           | 546 |
| 5.96.2.1 | <a href="#">AK_get_rel_exp()</a>                 | 546 |
| 5.96.2.2 | <a href="#">AK_get_view_obj_id()</a>             | 546 |
| 5.96.2.3 | <a href="#">AK_get_view_query()</a>              | 547 |
| 5.96.2.4 | <a href="#">AK_view_add()</a>                    | 547 |
| 5.96.2.5 | <a href="#">AK_view_change_query()</a>           | 548 |
| 5.96.2.6 | <a href="#">AK_view_remove_by_name()</a>         | 548 |
| 5.96.2.7 | <a href="#">AK_view_remove_by_obj_id()</a>       | 549 |
| 5.96.2.8 | <a href="#">AK_view_rename()</a>                 | 549 |
| 5.96.2.9 | <a href="#">AK_view_test()</a>                   | 550 |
| 5.97     | <a href="#">tools/comments.py File Reference</a> | 550 |
| 5.97.1   | <a href="#">Detailed Description</a>             | 550 |

|  |     |
|--|-----|
| 5.98 tools/getFiles.sh File Reference . . . . .                | 550 |
| 5.98.1 Detailed Description . . . . .                          | 550 |
| 5.99 tools/parseC.sh File Reference . . . . .                  | 551 |
| 5.99.1 Detailed Description . . . . .                          | 551 |
| 5.100 tools/parsePy.sh File Reference . . . . .                | 551 |
| 5.100.1 Detailed Description . . . . .                         | 551 |
| 5.101 tools/updateVersion.sh File Reference . . . . .          | 551 |
| 5.101.1 Detailed Description . . . . .                         | 551 |
| 5.102 trans/transaction.c File Reference . . . . .             | 551 |
| 5.102.1 Detailed Description . . . . .                         | 553 |
| 5.102.2 Function Documentation . . . . .                       | 553 |
| 5.102.2.1 AK_acquire_lock() . . . . .                          | 554 |
| 5.102.2.2 AK_add_hash_entry_list() . . . . .                   | 554 |
| 5.102.2.3 AK_add_lock() . . . . .                              | 555 |
| 5.102.2.4 AK_all_transactions_finished() . . . . .             | 555 |
| 5.102.2.5 AK_create_lock() . . . . .                           | 555 |
| 5.102.2.6 AK_create_new_transaction_thread() . . . . .         | 556 |
| 5.102.2.7 AK_delete_hash_entry_list() . . . . .                | 556 |
| 5.102.2.8 AK_delete_lock_entry_list() . . . . .                | 557 |
| 5.102.2.9 AK_execute_commands() . . . . .                      | 557 |
| 5.102.2.10 AK_execute_transaction() . . . . .                  | 558 |
| 5.102.2.11 AK_get_memory_blocks() . . . . .                    | 558 |
| 5.102.2.12 AK_handle_observable_transaction_action() . . . . . | 559 |
| 5.102.2.13 AK_init_observable_transaction() . . . . .          | 559 |
| 5.102.2.14 AK_init_observer_lock() . . . . .                   | 559 |
| 5.102.2.15 AK_isLock_waiting() . . . . .                       | 560 |
| 5.102.2.16 AK_lock_released() . . . . .                        | 560 |
| 5.102.2.17 AK_memory_block_hash() . . . . .                    | 560 |
| 5.102.2.18 AK_on_all_transactions_end() . . . . .              | 561 |
| 5.102.2.19 AK_on_lock_release() . . . . .                      | 561 |

|   |     |
|---|-----|
| 5.102.2.20AK_on_observable_notify()                 | 561 |
| 5.102.2.21AK_on_transaction_end()                   | 562 |
| 5.102.2.22AK_release_locks()                        | 562 |
| 5.102.2.23AK_remove_transaction_thread()            | 563 |
| 5.102.2.24AK_search_empty_link_for_hook()           | 563 |
| 5.102.2.25AK_search_existing_link_for_hook()        | 563 |
| 5.102.2.26AK_search_lock_entry_list_by_key()        | 564 |
| 5.102.2.27AK_transaction_finished()                 | 564 |
| 5.102.2.28AK_transaction_manager()                  | 565 |
| 5.102.2.29AK_transaction_register_observer()        | 565 |
| 5.102.2.30AK_transaction_unregister_observer()      | 565 |
| 5.102.2.31handle_transaction_notify()               | 566 |
| 5.103trans/transaction.h File Reference             | 566 |
| 5.103.1 Detailed Description                        | 569 |
| 5.103.2 Enumeration Type Documentation              | 569 |
| 5.103.2.1 NoticeType                                | 570 |
| 5.103.3 Function Documentation                      | 570 |
| 5.103.3.1 AK_acquire_lock()                         | 570 |
| 5.103.3.2 AK_add_hash_entry_list()                  | 571 |
| 5.103.3.3 AK_add_lock()                             | 571 |
| 5.103.3.4 AK_all_transactions_finished()            | 572 |
| 5.103.3.5 AK_create_lock()                          | 572 |
| 5.103.3.6 AK_create_new_transaction_thread()        | 572 |
| 5.103.3.7 AK_delete_hash_entry_list()               | 573 |
| 5.103.3.8 AK_delete_lock_entry_list()               | 573 |
| 5.103.3.9 AK_execute_commands()                     | 574 |
| 5.103.3.10AK_execute_transaction()                  | 574 |
| 5.103.3.11AK_get_memory_blocks()                    | 575 |
| 5.103.3.12AK_handle_observable_transaction_action() | 575 |
| 5.103.3.13AK_init_observable_transaction()          | 575 |

|  |     |
|--|-----|
| 5.103.3.14AK_init_observer_lock()              | 576 |
| 5.103.3.15AK_isLock_waiting()                  | 576 |
| 5.103.3.16AK_lock_released()                   | 577 |
| 5.103.3.17AK_memory_block_hash()               | 577 |
| 5.103.3.18AK_on_all_transactions_end()         | 577 |
| 5.103.3.19AK_on_lock_release()                 | 578 |
| 5.103.3.20AK_on_observable_notify()            | 578 |
| 5.103.3.21AK_on_transaction_end()              | 578 |
| 5.103.3.22AK_release_locks()                   | 579 |
| 5.103.3.23AK_remove_transaction_thread()       | 579 |
| 5.103.3.24AK_search_empty_link_for_hook()      | 580 |
| 5.103.3.25AK_search_existing_link_for_hook()   | 580 |
| 5.103.3.26AK_search_lock_entry_list_by_key()   | 580 |
| 5.103.3.27AK_transaction_finished()            | 581 |
| 5.103.3.28AK_transaction_manager()             | 581 |
| 5.103.3.29AK_transaction_register_observer()   | 582 |
| 5.103.3.30AK_transaction_unregister_observer() | 582 |
| 5.103.3.31handle_transaction_notify()          | 582 |

# Chapter 1

## Todo List

### Member [AK\\_acquire\\_lock](#) (int, int, pthread\_t)

Implement a better deadlock detection. This method uses a very simple approach. It waits for 60sec before it restarts a transaction.

### Member [AK\\_acquire\\_lock](#) (int, int, pthread\_t)

Implement a better deadlock detection. This method uses a very simple approach. It waits for 60sec before it restarts a transaction.

### Member [AK\\_archive\\_log](#) (int sig)

this function takes static filename to store the failed commands, create certain logic that would make the function to use dynamic filename (this is partly implemented inside [AK\\_get\\_timestamp](#), but there is no logic that uses the last file when recovering - [recovery.c](#)) {link} [recovery.c](#) function test

### Member [AK\\_execute\\_commands](#) (command \*, int)

Check multithreading, check if it's working correctly

### Member [AK\\_execute\\_commands](#) (command \*, int)

Check multithreading, check if it's working correctly

### Member [AK\\_get\\_timestamp](#) ()

Think about this in the future when creating multiple binary recovery files. Implementation gives the timestamp, but is not used anywhere for now.

### Member [AK\\_memory\\_block\\_hash](#) (int)

The current implementation is very limited it doesn't cope well with collision. recommendation use some better version of hash calculation. Maybe Knuth's memory address hashing function.

### Member [AK\\_memory\\_block\\_hash](#) (int)

The current implementation is very limited it doesn't cope well with collision. recommendation use some better version of hash calculation. Maybe Knuth's memory address hashing function.





## Chapter 2

# Class Index

### 2.1 Class List

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

|   |    |  |
|---|----|--|
| <a href="#">_dictionary_</a>  |    |  |
| Dictionary object   | 11 |  |
| <a href="#">_file_metadata</a>  | 12 |  |
| <a href="#">_notifyDetails</a>  | 12 |  |
| <a href="#">AK_agg_input</a>  |    |  |
| Structure that contains attributes from table header, tasks for this table and counter value  | 13 |  |
| <a href="#">AK_agg_value</a>  |    |  |
| Structure that contains attribute name, date and aggregation task associated  | 13 |  |
| <a href="#">AK_block</a>  |    |  |
| Structure that defines a block of data inside a DB file. It contains address, type, chained_with, AK_free space, last_tuple_dict_id, header and tuple_dict and data   | 14 |  |
| <a href="#">AK_block_activity</a>   |    |  |
| Structure which holds information about each block, whether it is locked for reading or writing. It is important to note such information, to enable quick and thread-safe reading from or writing to disk. Structure contains of: locked_for_reading - thread which locks particular block for reading will set this value locked_for_writing - thread which locks particular block for writing will set this value block_lock - each reading and writing operation will be done atomically and un-interruptable, using this mutex block lock reading_done - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block writing_done - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it thread_holding_lock - the only thread which can unlock locked "block_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it | 15 |  |
| <a href="#">AK_blocktable</a>   | 16 |  |
| <a href="#">AK_command_recovery_struct</a>  |    |  |
| Recovery structure used to recover commands from binary file  | 16 |  |
| <a href="#">AK_command_struct</a>   | 17 |  |
| <a href="#">AK_create_table_struct</a>  | 17 |  |
| <a href="#">AK_db_cache</a>   |    |  |
| Structure that defines global cache memory  | 18 |  |
| <a href="#">AK_debmod_state</a>   |    |  |
| Global structure that holds all relevant information for the debug mode and related functionality   | 19 |  |
| <a href="#">AK_header</a>   |    |  |
| Structure that represents header structure of blocks (describes an attribute inside an object). It contains type, attribute name, integrity, constraint name and constraint code  | 19 |  |

|   |   |    |
|---|---|----|
| <a href="#">AK_mem_block</a>            | Structure that defines a block of data in memory . . . . .  | 20 |
| <a href="#">AK_operand</a>              | . . . . .   | 21 |
| <a href="#">AK_query_mem</a>            | Structure that defines global query memory . . . . .  | 22 |
| <a href="#">AK_query_mem_dict</a>       | Structure that defines global query memory for data dictionaries . . . . .  | 23 |
| <a href="#">AK_query_mem_lib</a>        | Structure that defines global query memory for libraries . . . . .  | 23 |
| <a href="#">AK_query_mem_result</a>     | Structure that defines global query memory for results . . . . .  | 24 |
| <a href="#">AK_redo_log</a>             | Structure that defines global redo log . . . . .  | 25 |
| <a href="#">AK_ref_item</a>             | Structure that represents reference item. It contains of table, attributes, parent table and it's attributes, number of attributes, constraint and type of reference . . . . .  | 26 |
| <a href="#">AK_results</a>              | Structure used for in-memory result caching . . . . .   | 27 |
| <a href="#">AK_synchronization_info</a> | Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex) . . . . .  | 28 |
| <a href="#">AK_tuple_dict</a>           | Structure that defines a mapping in a header of an object to the actual entries (data). It contains type, address and size . . . . .  | 28 |
| <a href="#">blocktable</a>              | Structure that defines bit status of blocks, last initialized and last allocated index . . . . .  | 29 |
| <a href="#">btree_node</a>              | . . . . .   | 29 |
| <a href="#">bucket_elem</a>             | Structure for defining a single bucket element . . . . .  | 30 |
| <a href="#">cost_eval_t</a>             | Structure for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name) . . . . .   | 31 |
| <a href="#">DEBUG_LEVEL</a>             | Structure for setting debug level. Divide debug information according to their importance. More levels can be defined in the enum if needed. Each debug level can be easily excluded from output by setting corresponding enum element to 0 . . . . .           | 31 |
| <a href="#">DEBUG_TYPE</a>              | Structure for setting debug type. Divide debug information according to their type (e.g. DB modules). More modules can be additional added to the enum. Each debug type can be easily excluded from output by setting corresponding enum element to 0 . . . . . | 32 |
| <a href="#">drop_arguments</a>          | . . . . .   | 32 |
| <a href="#">hash_bucket</a>             | Structure for hash bucket for table hashing . . . . .   | 33 |
| <a href="#">hash_info</a>               | Structure for defining a hash info element . . . . .  | 34 |
| <a href="#">intersect_attr</a>          | Structure defines intersect attribute . . . . .   | 34 |
| <a href="#">list_node</a>               | Structure defines a list node . . . . .   | 35 |
| <a href="#">list_structure_ad</a>       | . . . . .   | 36 |
| <a href="#">list_structure_add</a>      | Structure that defines linked list node for index . . . . .   | 36 |
| <a href="#">main_bucket</a>             | Structure for defining main bucket for table hashing . . . . .  | 37 |
| <a href="#">memoryAddresses</a>         | Structure that represents a linked list of locked addresses . . . . .   | 38 |
| <a href="#">Observable</a>              | Structure defines functions for observable object . . . . .   | 38 |

|  |    |
|--|----|
| <a href="#">observable_transaction</a>   |    |
| Structure which defines transaction observable type . . . . .  | 39 |
| <a href="#">observable_transaction_struct</a> . . . . .  | 40 |
| <a href="#">Observer</a>   |    |
| Structure defines functions for observer object . . . . .  | 40 |
| <a href="#">observer_lock</a>  |    |
| Structure which defines transaction lock observer type . . . . .   | 41 |
| <a href="#">root_info</a> . . . . .  | 42 |
| <a href="#">search_params</a>  |    |
| Structure that contains attribute name, lower and upper data value, special(NULL or *) which is input for AK_equisearch_unsorted and AK_rangesearch_unsorted . . . . .                                   | 42 |
| <a href="#">search_result</a>  |    |
| Structure which represents search result of AK_equisearch_unsorted and AK_rangesearch_unsorted . . . . .   | 43 |
| <a href="#">Stack</a>  |    |
| Structure defines a <a href="#">Stack</a> element. Every <a href="#">Stack</a> has its <a href="#">Vertex</a> pointer and pointer to next <a href="#">Stack</a> in the linked list . . . . .             | 43 |
| <a href="#">struct_add</a>   |    |
| Structure defining node address . . . . .  | 44 |
| <a href="#">Successor</a>  |    |
| Structure defines a <a href="#">Successor</a> element. Every <a href="#">Successor</a> has its <a href="#">Vertex</a> pointer and pointer to next <a href="#">Successor</a> in the linked list . . . . . | 45 |
| <a href="#">table_addresses</a>  |    |
| Structure that defines start and end address of extent . . . . .   | 45 |
| <a href="#">threadContainer</a>  |    |
| Structure that represents a linked list of threads . . . . .   | 46 |
| <a href="#">transaction_list_elem</a>  |    |
| Structure that represents LockTable entry about transaction lock holder.Element indexed by Hash table . . . . .  | 47 |
| <a href="#">transaction_list_head</a>  |    |
| Structure that represents LockTable entry about doubly linked list of collision in Hash table . . . . .  | 48 |
| <a href="#">transaction_locks_list_elem</a>  |    |
| Structure that represents LockTable entry about transaction resource lock . . . . .  | 49 |
| <a href="#">transactionData</a>  |    |
| Structure used to transport transaction data to the thread . . . . .   | 49 |
| <a href="#">TypeObservable</a> . . . . .   | 51 |
| <a href="#">TypeObserver</a> . . . . .   | 52 |
| <a href="#">Vertex</a>   |    |
| Structure defines a <a href="#">Vertex</a> node element. Every <a href="#">Vertex</a> has its VertexId, index, lowLink and pointer to next edge and vertex . . . . .                                     | 52 |



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

|  |     |
|--|-----|
| auxiliary/auxiliary.c                        | 55  |
| auxiliary/auxiliary.h                        | 72  |
| auxiliary/configuration.h                    | ??  |
| auxiliary/constants.h                        | 91  |
| auxiliary/debug.c                            | 95  |
| auxiliary/debug.h                            | 96  |
| auxiliary/dictionary.c                       |     |
| Implements a dictionary for string variables | 99  |
| auxiliary/dictionary.h                       |     |
| Implements a dictionary for string variables | 103 |
| auxiliary/iniparser.c                        |     |
| Parser for ini files                         | 108 |
| auxiliary/iniparser.h                        |     |
| Parser for ini files                         | 118 |
| auxiliary/mempro.c                           | 128 |
| auxiliary/mempro.h                           | 143 |
| auxiliary/observable.c                       | 157 |
| auxiliary/observable.h                       | 160 |
| dm/dbman.c                                   | 162 |
| dm/dbman.h                                   | 180 |
| file/blobs.c                                 | 201 |
| file/blobs.h                                 | 205 |
| file/fileio.c                                | 210 |
| file/fileio.h                                | 216 |
| file/files.c                                 | 222 |
| file/files.h                                 | 225 |
| file/filesearch.c                            | 227 |
| file/filesearch.h                            | 230 |
| file/filesort.h                              | 233 |
| file/id.c                                    | 235 |
| file/id.h                                    | 237 |
| file/sequence.c                              | 288 |
| file/sequence.h                              | 293 |
| file/table.c                                 | 298 |
| file/table.h                                 | 310 |

|                           |     |
|---------------------------|-----|
| file/test.c               | 323 |
| file/test.h               | 327 |
| file/idx/bitmap.c         | 239 |
| file/idx/bitmap.h         | 246 |
| file/idx/btree.c          | 253 |
| file/idx/btree.h          | 255 |
| file/idx/hash.c           | 257 |
| file/idx/hash.h           | 264 |
| file/idx/index.c          | 271 |
| file/idx/index.h          | 279 |
| mm/memoman.c              | 332 |
| mm/memoman.h              | 340 |
| opti/query_optimization.c | 349 |
| opti/query_optimization.h | 352 |
| opti/rel_eq_assoc.c       | 355 |
| opti/rel_eq_assoc.h       | 357 |
| opti/rel_eq_comut.c       | 360 |
| opti/rel_eq_comut.h       | 363 |
| opti/rel_eq_projection.c  | 366 |
| opti/rel_eq_projection.h  | 371 |
| opti/rel_eq_selection.c   | 377 |
| opti/rel_eq_selection.h   | 383 |
| rec/archive_log.h         | 389 |
| rec/recovery.c            | 391 |
| rec/recovery.h            | ??  |
| rec/redo_log.c            | 394 |
| rec/redo_log.h            | ??  |
| rel/aggregation.c         | 396 |
| rel/aggregation.h         | 402 |
| rel/difference.c          | 407 |
| rel/difference.h          | 408 |
| rel/expression_check.c    | 410 |
| rel/expression_check.h    | 414 |
| rel/intersect.c           | 417 |
| rel/intersect.h           | 419 |
| rel/nat_join.c            | 421 |
| rel/nat_join.h            | 424 |
| rel/product.c             | 428 |
| rel/product.h             | 430 |
| rel/projection.c          | 432 |
| rel/projection.h          | 438 |
| rel/selection.c           | 446 |
| rel/selection.h           | 449 |
| rel/theta_join.c          | 451 |
| rel/theta_join.h          | 454 |
| rel/union.c               | 457 |
| rel/union.h               | 458 |
| sql/command.h             | ??  |
| sql/drop.c                | 503 |
| sql/drop.h                | 506 |
| sql/function.c            | 509 |
| sql/function.h            | 515 |
| sql/privileges.c          | 521 |
| sql/privileges.h          | ??  |
| sql/select.c              | 533 |
| sql/select.h              | ??  |
| sql/trigger.c             | 534 |
| sql/trigger.h             | 539 |

|   |     |
|---|-----|
| sql/ <a href="#">view.c</a>   | 545 |
| sql/ <a href="#">view.h</a>   | ??  |
| sql/cs/ <a href="#">between.c</a>   | 460 |
| sql/cs/ <a href="#">between.h</a>   | 464 |
| sql/cs/ <a href="#">check_constraint.c</a>  | 467 |
| sql/cs/ <a href="#">check_constraint.h</a>  | 470 |
| sql/cs/ <a href="#">constraint_names.c</a>  | 474 |
| sql/cs/ <a href="#">constraint_names.h</a>  | 475 |
| sql/cs/ <a href="#">nnull.c</a>   | 477 |
| sql/cs/ <a href="#">nnull.h</a>   | 480 |
| sql/cs/ <a href="#">reference.c</a>   | 483 |
| sql/cs/ <a href="#">reference.h</a>   | 488 |
| sql/cs/ <a href="#">unique.c</a>  | 497 |
| sql/cs/ <a href="#">unique.h</a>  | 500 |
| tools/ <a href="#">comments.py</a>  |     |
| //!   | 550 |
| tools/ <a href="#">getFiles.sh</a>  |     |
| Finding all files that ends with extension .py or .c and storing them into file.txt | 550 |
| tools/ <a href="#">parseC.sh</a>  |     |
| Parsing every C file  | 551 |
| tools/ <a href="#">parsePy.sh</a>   |     |
| Parsing every Py file   | 551 |
| tools/ <a href="#">updateVersion.sh</a>   |     |
| Updating project version  | 551 |
| trans/ <a href="#">transaction.c</a>  | 551 |
| trans/ <a href="#">transaction.h</a>  | 566 |





## Chapter 4

# Class Documentation

### 4.1 `_dictionary_` Struct Reference

Dictionary object.

```
#include <dictionary.h>
```

#### Public Attributes

- `int n`
- `int size`
- `char ** val`
- `char ** key`
- `unsigned * hash`

#### 4.1.1 Detailed Description

Dictionary object.

This object contains a list of string/string associations. Each association is identified by a unique string key. Looking up values in the dictionary is speeded up by the use of a (hopefully collision-AK\_free) hash function.

#### 4.1.2 Member Data Documentation

##### 4.1.2.1 `hash`

```
unsigned* _dictionary_::hash
```

List of string keys

#### 4.1.2.2 key

```
char** _dictionary_::key
```

List of string values

#### 4.1.2.3 size

```
int _dictionary_::size
```

Number of entries in dictionary

#### 4.1.2.4 val

```
char** _dictionary_::val
```

Storage size

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

- [auxi/dictionary.h](#)

## 4.2 \_file\_metadata Struct Reference

### Public Attributes

- char \* **new\_path**
- char \* **new\_name**
- char \* **old\_path**
- char \* **old\_name**
- char \* **checksum**

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

- [file/blobs.h](#)

## 4.3 \_notifyDetails Struct Reference

### Public Attributes

- char \* **message**
- NotifyType **type**

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

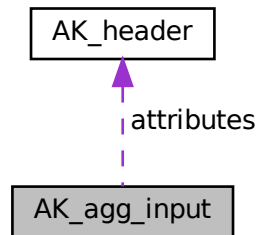
- [auxi/observable.c](#)

## 4.4 AK\_agg\_input Struct Reference

Structure that contains attributes from table header, tasks for this table and counter value.

```
#include <aggregation.h>
```

Collaboration diagram for AK\_agg\_input:



### Public Attributes

- [AK\\_header](#) **attributes** [[MAX\\_ATTRIBUTES](#)]
- int **tasks** [[MAX\\_ATTRIBUTES](#)]
- int **counter**

### 4.4.1 Detailed Description

Structure that contains attributes from table header, tasks for this table and counter value.

#### Author

Unknown

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

- [rel/aggregation.h](#)

## 4.5 AK\_agg\_value Struct Reference

Structure that contains attribute name, date and aggregation task associated.

```
#include <aggregation.h>
```

## Public Attributes

- char **att\_name** [[MAX\\_ATT\\_NAME](#)]
- char **data** [[MAX\\_VARCHAR\\_LENGTH](#)]
- int **agg\_task**

### 4.5.1 Detailed Description

Structure that contains attribute name, date and aggregation task associated.

#### Author

Unknown

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

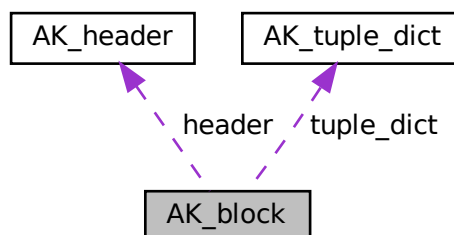
- [rel/aggregation.h](#)

## 4.6 AK\_block Struct Reference

Structure that defines a block of data inside a DB file. It contains address, type, chained\_with, AK\_free space, last\_tuple\_dict\_id, header and tuple\_dict and data.

```
#include <dbman.h>
```

Collaboration diagram for AK\_block:



## Public Attributes

- int [address](#)  
*block number (address) in DB file*
- int [type](#)  
*block type (can be BLOCK\_TYPE\_FREE, BLOCK\_TYPE\_NORMAL or BLOCK\_TYPE\_CHAINED)*
- int [chained\\_with](#)  
*address of chained block; NOT\_CHAINED otherwise*
- int [AK\\_free\\_space](#)  
*AK\_free space in block.*
- int [last\\_tuple\\_dict\\_id](#)
- [AK\\_header](#) [header](#) [[MAX\\_ATTRIBUTES](#)]  
*attribute definitions*
- [AK\\_tuple\\_dict](#) [tuple\\_dict](#) [[DATA\\_BLOCK\\_SIZE](#)]  
*dictionary of data entries*
- unsigned char [data](#) [[DATA\\_BLOCK\\_SIZE](#) \* [DATA\\_ENTRY\\_SIZE](#)]  
*actual data entries*

### 4.6.1 Detailed Description

Structure that defines a block of data inside a DB file. It contains address, type, chained\_with, AK\_free space, last\_tuple\_dict\_id, header and tuple\_dict and data.

#### Author

Markus Schatten

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

- [dm/dbman.h](#)

## 4.7 AK\_block\_activity Struct Reference

Structure which holds information about each block, whether it is locked for reading or writing. It is important to note such information, to enable quick and thread-safe reading from or writing to disk. Structure contains of: `locked_for_reading` - thread which locks particular block for reading will set this value `locked_for_writing` - thread which locks particular block for writing will set this value `block_lock` - each reading and writing operation will be done atomically and uninterruptable, using this mutex `block_lock` `reading_done` - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block `writing_done` - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it `thread_holding_lock` - the only thread which can unlock `locked_block_lock` is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it.

```
#include <dbman.h>
```

## Public Attributes

- short **locked\_for\_reading**
- short **locked\_for\_writing**
- pthread\_mutex\_t **block\_lock**
- pthread\_cond\_t **writing\_done**
- pthread\_cond\_t **reading\_done**
- int \* **thread\_holding\_lock**

### 4.7.1 Detailed Description

Structure which holds information about each block, whether it is locked for reading or writing. It is important to note such information, to enable quick and thread-safe reading from or writing to disk. Structure contains of: `locked_for_reading` - thread which locks particular block for reading will set this value `locked_for_writing` - thread which locks particular block for writing will set this value `block_lock` - each reading and writing operation will be done atomically and uninterruptable, using this mutex `block_lock` `reading_done` - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block `writing_done` - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it `thread_holding_lock` - the only thread which can unlock `locked_block_lock` is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it.

#### Author

Domagoj Šitum

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

- [dm/dbman.h](#)

## 4.8 AK\_blocktable Struct Reference

### Public Attributes

- unsigned int **allocationtable** [DB\_FILE\_BLOCKS\_NUM\_EX]
- unsigned char **bittable** [BITNSLOTS(DB\_FILE\_BLOCKS\_NUM\_EX)]
- int **last\_allocated**
- int **last\_initialized**
- int **prepared**
- time\_t **ltime**

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

- [dm/dbman.h](#)

## 4.9 AK\_command\_recovery\_struct Struct Reference

recovery structure used to recover commands from binary file

```
#include <memoman.h>
```

### Public Attributes

- int **operation**
- char **table\_name** [[MAX\\_VARCHAR\\_LENGTH](#)]
- char **arguments** [[MAX\\_ATTRIBUTES](#)][[MAX\\_VARCHAR\\_LENGTH](#)]
- int **finished**

#### 4.9.1 Detailed Description

recovery structure used to recover commands from binary file

Structure that contains all vital information for the command that is about to execute. It is defined by the operation (INSERT, UPDATE, DELETE that are defined inside the const.c file), table where the data is stored, and certain data that will be stored.

#### Author

Tomislav Turek

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

- mm/[memoman.h](#)

## 4.10 AK\_command\_struct Struct Reference

### Public Attributes

- int **id\_command**
- char \* **tblName**
- void \* **parameters**

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

- sql/command.h

## 4.11 AK\_create\_table\_struct Struct Reference

### Public Attributes

- char **name** [[MAX\\_ATT\\_NAME](#)]
- int **type**

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

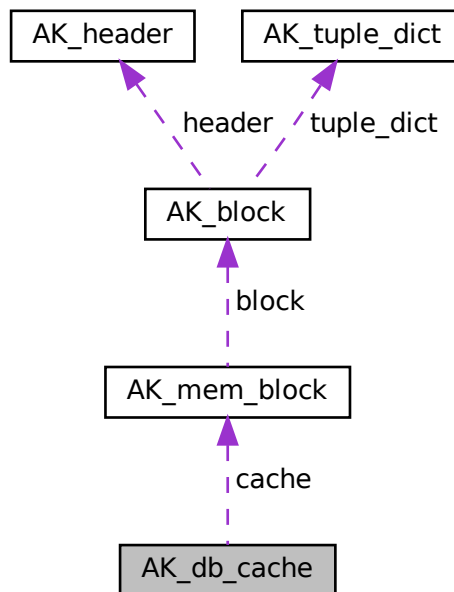
- file/[table.h](#)

## 4.12 AK\_db\_cache Struct Reference

Structure that defines global cache memory.

```
#include <memoman.h>
```

Collaboration diagram for AK\_db\_cache:



### Public Attributes

- `AK_mem_block * cache [MAX_CACHE_MEMORY]`  
*last recently read blocks*
- `int next_replace`  
*next cached block to be replaced (0 - MAX\_CACHE\_MEMORY-1); depends on caching algorithm*

### 4.12.1 Detailed Description

Structure that defines global cache memory.

Author

Unknown

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

- `mm/memoman.h`



## 4.13 AK\_debmod\_state Struct Reference

Global structure that holds all relevant information for the debug mode and related functionality.

```
#include <mempro.h>
```

### Public Attributes

- uint8\_t **init**
- uint32\_t **page\_size**
- uint8\_t **ready**
- void \* **page** [[AK\\_DEBMOD\\_PAGES\\_NUM](#)]
- uint8\_t **used** [[AK\\_DEBMOD\\_PAGES\\_NUM](#)]
- uint32\_t **nomi** [[AK\\_DEBMOD\\_PAGES\\_NUM](#)]
- uint32\_t **real** [[AK\\_DEBMOD\\_PAGES\\_NUM](#)]
- uint8\_t **dirty** [[AK\\_DEBMOD\\_PAGES\\_NUM](#)]
- char **function** [[AK\\_DEBMOD\\_MAX\\_FUNCTIONS](#)][[AK\\_DEBMOD\\_MAX\\_FUNC\\_NAME](#)]
- int32\_t **last\_function\_id**
- int32\_t **alloc\_owner** [[AK\\_DEBMOD\\_PAGES\\_NUM](#)]
- int32\_t **free\_owner** [[AK\\_DEBMOD\\_PAGES\\_NUM](#)]
- int8\_t **func\_used\_by** [[AK\\_DEBMOD\\_MAX\\_FUNCTIONS](#)][[AK\\_DEBMOD\\_MAX\\_FUNCTIONS](#)]
- uint8\_t **print**
- int32\_t **fstack\_size**
- int32\_t **fstack\_items** [[AK\\_DEBMOD\\_STACKSIZE](#)]

### 4.13.1 Detailed Description

Global structure that holds all relevant information for the debug mode and related functionality.

#### Author

Marin Rukavina, Mislav Bozicevic

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

- [auxi/mempro.h](#)

## 4.14 AK\_header Struct Reference

Structure that represents header structure of blocks (describes an attribute inside an object). It contains type, attribute name, integrity, constraint name and constraint code.

```
#include <dbman.h>
```

## Public Attributes

- int [type](#)  
*type of attribute*
- char [att\\_name](#) [[MAX\\_ATT\\_NAME](#)]  
*attribute name*
- int [integrity](#) [[MAX\\_CONSTRAINTS](#)]  
*standard integrity constraints*
- char [constr\\_name](#) [[MAX\\_CONSTRAINTS](#)][[MAX\\_CONSTR\\_NAME](#)]  
*extra integrity constraint names*
- char [constr\\_code](#) [[MAX\\_CONSTRAINTS](#)][[MAX\\_CONSTR\\_CODE](#)]  
*extra integrity constraint codes*

### 4.14.1 Detailed Description

Structure that represents header structure of blocks (describes an attribute inside an object). It contains type, attribute name, integrity, constraint name and constraint code.

#### Author

Markus Schatten

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

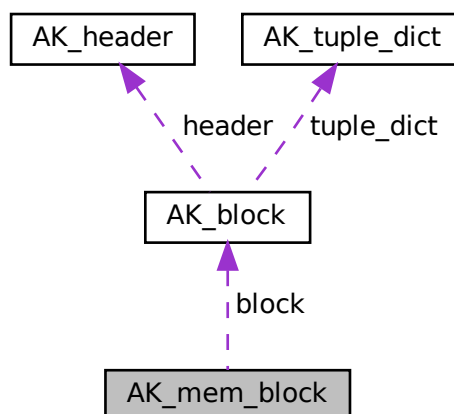
- [dm/dbman.h](#)

## 4.15 AK\_mem\_block Struct Reference

Structure that defines a block of data in memory.

```
#include <memoman.h>
```

Collaboration diagram for AK\_mem\_block:



### Public Attributes

- [AK\\_block](#) \* [block](#)  
*pointer to block from DB file*
- int [dirty](#)  
*dirty bit (BLOCK\_CLEAN if unchanged; BLOCK\_DIRTY if changed but not yet written to file)*
- unsigned long [timestamp\\_read](#)  
*timestamp when the block has lastly been read*
- unsigned long [timestamp\\_last\\_change](#)  
*timestamp when the block has lastly been changed*

#### 4.15.1 Detailed Description

Structure that defines a block of data in memory.

#### Author

Unknown

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

- [mm/memoman.h](#)

## 4.16 AK\_operand Struct Reference

### Public Attributes

- char **value** [[MAX\\_VARCHAR\\_LENGTH](#)]
- int **type**

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

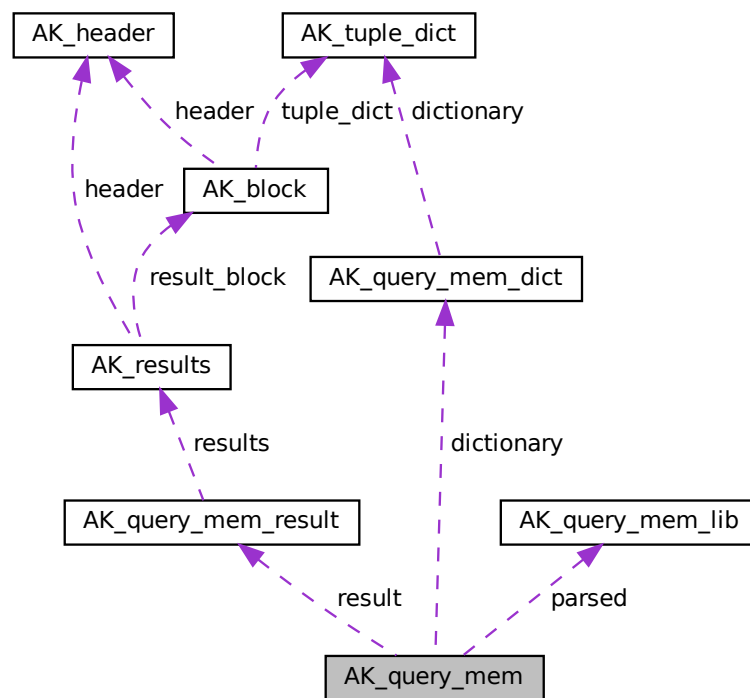
- [rel/projection.h](#)

## 4.17 AK\_query\_mem Struct Reference

Structure that defines global query memory.

```
#include <memoman.h>
```

Collaboration diagram for AK\_query\_mem:



### Public Attributes

- `AK_query_mem_lib * parsed`  
*parsed queries*
- `AK_query_mem_dict * dictionary`  
*obtained data dictionaries*
- `AK_query_mem_result * result`  
*obtained query results*

### 4.17.1 Detailed Description

Structure that defines global query memory.

Author

Unknown

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

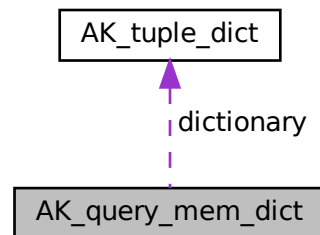
- `mm/memoman.h`

## 4.18 AK\_query\_mem\_dict Struct Reference

Structure that defines global query memory for data dictionaries.

```
#include <memoman.h>
```

Collaboration diagram for AK\_query\_mem\_dict:



### Public Attributes

- `AK_tuple_dict * dictionary` [`MAX_QUERY_DICT_MEMORY`]  
*last used data dictionaries*
- `int next_replace`  
*next dictionary to be replaced (0 - `MAX_QUERY_DICT_MEMORY-1`); field pointer (LIFO)*

### 4.18.1 Detailed Description

Structure that defines global query memory for data dictionaries.

#### Author

Unkown

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

- `mm/memoman.h`

## 4.19 AK\_query\_mem\_lib Struct Reference

Structure that defines global query memory for libraries.

```
#include <memoman.h>
```

## Public Attributes

- char [parsed](#) [[MAX\\_QUERY\\_LIB\\_MEMORY](#)]  
*last parsed queries; to be changed to more adequate data structure*
- int [next\\_replace](#)  
*next query to be replaced (0 - MAX\_QUERY\_LIB\_MEMORY-1); field pointer (LIFO)*

### 4.19.1 Detailed Description

Structure that defines global query memory for libraries.

#### Author

Unkown

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

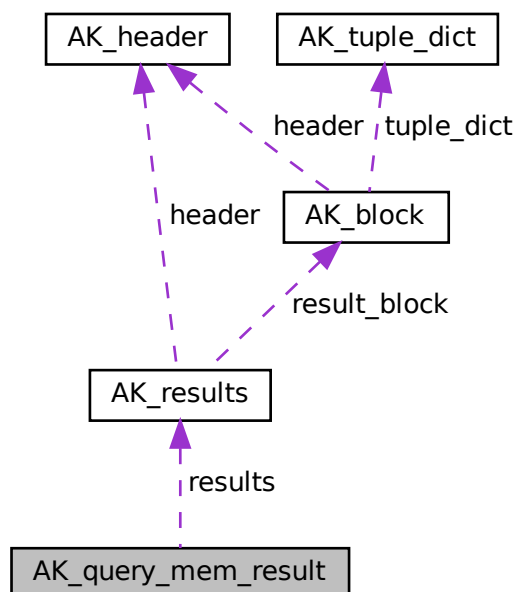
- [mm/memoman.h](#)

## 4.20 AK\_query\_mem\_result Struct Reference

Structure that defines global query memory for results.

```
#include <memoman.h>
```

Collaboration diagram for AK\_query\_mem\_result:



## Public Attributes

- [AK\\_results](#) \* **results**
- int [next\\_replace](#)  
*next result to be replaced (0 - MAX\_QUERY\_RESULT\_MEMORY-1); field pointer (LIFO)*

### 4.20.1 Detailed Description

Structure that defines global query memory for results.

#### Author

Unknown

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

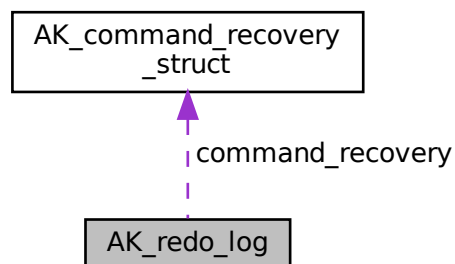
- mm/[memoman.h](#)

## 4.21 AK\_redo\_log Struct Reference

Structure that defines global redo log.

```
#include <memoman.h>
```

Collaboration diagram for AK\_redo\_log:



## Public Attributes

- [AK\\_command\\_recovery\\_struct](#) **command\_recovery** [MAX\_REDO\_LOG\_ENTRIES]
- int **number**

### 4.21.1 Detailed Description

Structure that defines global redo log.

The structure defines an array of commands being executed at the moment. If and when commands fail to execute, the rest of the commands that did not execute will be stored inside a binary file and the system will try recovery and execution for those commands. With the array, we also store a number that defines the number of commands that failed to execute (length of `command_recovery` array).

#### Author

Dražen Bandić, updated by Tomislav Turek

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

- [mm/memoman.h](#)

## 4.22 AK\_ref\_item Struct Reference

Structure that represents reference item. It contains of table, attributes, parent table and it's attributes, number of attributes, constraint and type of reference.

```
#include <reference.h>
```

### Public Attributes

- char **table** [[MAX\\_ATT\\_NAME](#)]
- char **attributes** [[MAX\\_REFERENCE\\_ATTRIBUTES](#)][[MAX\\_ATT\\_NAME](#)]
- char **parent** [[MAX\\_ATT\\_NAME](#)]
- char **parent\_attributes** [[MAX\\_REFERENCE\\_ATTRIBUTES](#)][[MAX\\_ATT\\_NAME](#)]
- int **attributes\_number**
- char **constraint** [[MAX\\_VARCHAR\\_LENGTH](#)]
- int **type**

### 4.22.1 Detailed Description

Structure that represents reference item. It contains of table, attributes, parent table and it's attributes, number of attributes, constraint and type of reference.

#### Author

Dejan Franković

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

- [sql/cs/reference.h](#)

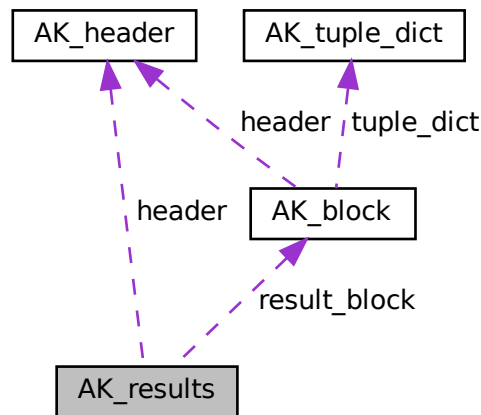


## 4.23 AK\_results Struct Reference

Structure used for in-memory result caching.

```
#include <memoman.h>
```

Collaboration diagram for AK\_results:



### Public Attributes

- unsigned long **result\_id**
- int **result\_size**
- char **date\_created** [80]
- short **free**
- char \* **source\_table**
- [AK\\_block](#) \* **result\_block**
- [AK\\_header](#) **header** [[MAX\\_ATTRIBUTES](#)]

#### 4.23.1 Detailed Description

Structure used for in-memory result caching.

Author

Mario Novoselec

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

- [mm/memoman.h](#)

## 4.24 AK\_synchronization\_info Struct Reference

Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex).

```
#include <auxiliary.h>
```

### Public Attributes

- int **init**
- int **ready**

### 4.24.1 Detailed Description

Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex).

#### Author

Marko Sinko

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

- auxi/[auxiliary.h](#)

## 4.25 AK\_tuple\_dict Struct Reference

Structure that defines a mapping in a header of an object to the actual entries (data). It contains type, address and size.

```
#include <dbman.h>
```

### Public Attributes

- int [type](#)  
*data entry type*
- int [address](#)  
*data entry address (in AK\_block->data)*
- int [size](#)  
*data entry size (using sizeof( \*\*\* ) )*

### 4.25.1 Detailed Description

Structure that defines a mapping in a header of an object to the actual entries (data). It contains type, address and size.

Author

Markus Schatten

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

- [dm/dbman.h](#)

## 4.26 blocktable Struct Reference

Structure that defines bit status of blocks, last initialized and last allocated index.

```
#include <dbman.h>
```

### 4.26.1 Detailed Description

Structure that defines bit status of blocks, last initialized and last allocated index.

Author

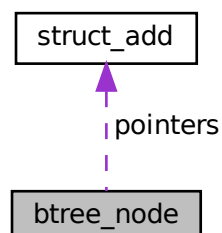
dv

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

- [dm/dbman.h](#)

## 4.27 btree\_node Struct Reference

Collaboration diagram for btree\_node:



## Public Attributes

- int **values** [B]
- [struct\\_add](#) **pointers** [B+1]

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

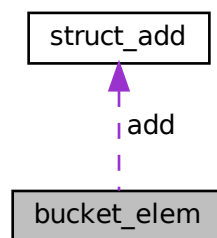
- file/idx/[btree.h](#)

## 4.28 bucket\_elem Struct Reference

Structure for defining a single bucket element.

```
#include <hash.h>
```

Collaboration diagram for bucket\_elem:



## Public Attributes

- unsigned int [value](#)  
*bucket element hash value*
- [struct\\_add](#) [add](#)  
*bucket element address values*

### 4.28.1 Detailed Description

Structure for defining a single bucket element.

#### Author

Unknown

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

- file/idx/[hash.h](#)

## 4.29 cost\_eval\_t Struct Reference

Structure for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name)

```
#include <rel_eq_assoc.h>
```

### Public Attributes

- int **value**
- char **data** [[MAX\\_VARCHAR\\_LENGTH](#)]

#### 4.29.1 Detailed Description

Structure for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name)

##### Author

Dino Laktašić

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

- [opti/rel\\_eq\\_assoc.h](#)

## 4.30 DEBUG\_LEVEL Struct Reference

Structure for setting debug level. Divide debug information according to their importance. More levels can be defined in the enum if needed. Each debug level can be easily excluded from output by setting corresponding enum element to 0.

```
#include <debug.h>
```

#### 4.30.1 Detailed Description

Structure for setting debug level. Divide debug information according to their importance. More levels can be defined in the enum if needed. Each debug level can be easily excluded from output by setting corresponding enum element to 0.

##### Author

Dino Laktašić

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

- [auxi/debug.h](#)

## 4.31 DEBUG\_TYPE Struct Reference

Structure for setting debug type. Divide debug information according to their type (e.g. DB modules). More modules can be additional added to the enum. Each debug type can be easily excluded from output by setting corresponding enum element to 0.

```
#include <debug.h>
```

### 4.31.1 Detailed Description

Structure for setting debug type. Divide debug information according to their type (e.g. DB modules). More modules can be additional added to the enum. Each debug type can be easily excluded from output by setting corresponding enum element to 0.

#### Author

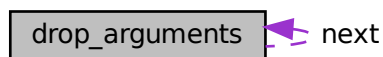
Dino Laktašić

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

- [auxi/debug.h](#)

## 4.32 drop\_arguments Struct Reference

Collaboration diagram for drop\_arguments:



#### Public Attributes

- void \* **value**
- struct [drop\\_arguments](#) \* **next**

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

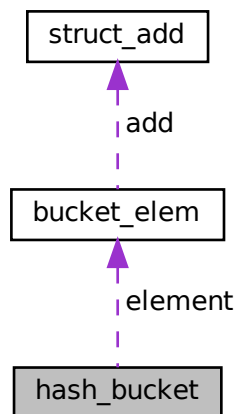
- [sql/drop.h](#)

## 4.33 hash\_bucket Struct Reference

Structure for hash bucket for table hashing.

```
#include <hash.h>
```

Collaboration diagram for hash\_bucket:



### Public Attributes

- `int bucket_level`  
*hash bucket level*
- `bucket_elem element [HASH_BUCKET_SIZE]`  
*hash bucket array of `bucket_elem` elements*

#### 4.33.1 Detailed Description

Structure for hash bucket for table hashing.

#### Author

Unknown

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

- `file/idx/hash.h`

## 4.34 hash\_info Struct Reference

Structure for defining a hash info element.

```
#include <hash.h>
```

### Public Attributes

- int [modulo](#)  
*modulo value for hash function*
- int [main\\_bucket\\_num](#)  
*bucket number*
- int [hash\\_bucket\\_num](#)  
*hash bucket number*

### 4.34.1 Detailed Description

Structure for defining a hash info element.

Author

Unknown

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

- file/idx/[hash.h](#)

## 4.35 intersect\_attr Struct Reference

Structure defines intersect attribute.

```
#include <intersect.h>
```

### Public Attributes

- int [type](#)  
*type of attribute*
- char [att\\_name](#) [MAX\_ATT\_NAME]  
*attribute name*



### 4.35.1 Detailed Description

Structure defines intersect attribute.

Author

Dino Laktašić

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

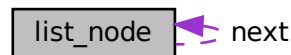
- [rel/intersect.h](#)

## 4.36 list\_node Struct Reference

Structure defines a list node.

```
#include <auxiliary.h>
```

Collaboration diagram for list\_node:



### Public Attributes

- int [type](#)  
*TODO - type, attribute name, table staviti na početak polja data data type.*
- int **size**
- char [data](#) [[MAX\\_VARCHAR\\_LENGTH](#)]  
*loaded data*
- char **table** [[MAX\\_ATT\\_NAME](#)]
- char **attribute\_name** [[MAX\\_ATT\\_NAME](#)]
- int **constraint**
- struct [list\\_node](#) \* [next](#)  
*pointer to next element*

### 4.36.1 Detailed Description

Structure defines a list node.

Author

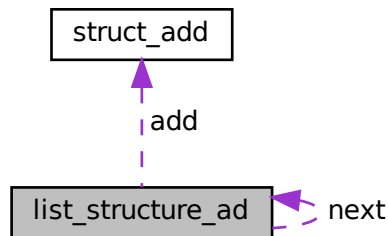
Ljiljana Pintarić

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

- [auxi/auxiliary.h](#)

## 4.37 list\_structure\_ad Struct Reference

Collaboration diagram for list\_structure\_ad:



### Public Attributes

- char \* [attName](#)  
*attribute name*
- [struct\\_add](#) [add](#)  
*addresses*
- struct [list\\_structure\\_ad](#) \* [next](#)  
*next node pointer*

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

- file/idx/[index.h](#)

## 4.38 list\_structure\_add Struct Reference

Structure that defines linked list node for index.

```
#include <index.h>
```

### 4.38.1 Detailed Description

Structure that defines linked list node for index.

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

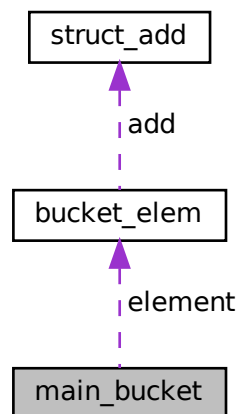
- file/idx/[index.h](#)

## 4.39 main\_bucket Struct Reference

Structure for defining main bucket for table hashing.

```
#include <hash.h>
```

Collaboration diagram for main\_bucket:



### Public Attributes

- `bucket_elem element [MAIN_BUCKET_SIZE]`  
*main bucket array of `bucket_elem` elements*

#### 4.39.1 Detailed Description

Structure for defining main bucket for table hashing.

#### Author

Unknown

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

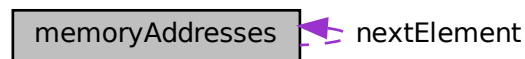
- file/idx/[hash.h](#)

## 4.40 memoryAddresses Struct Reference

Structure that represents a linked list of locked addresses.

```
#include <transaction.h>
```

Collaboration diagram for memoryAddresses:



### Public Attributes

- int **adresa**
- struct [memoryAddresses](#) \* **nextElement**

### 4.40.1 Detailed Description

Structure that represents a linked list of locked addresses.

Author

Frane Jakelić

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

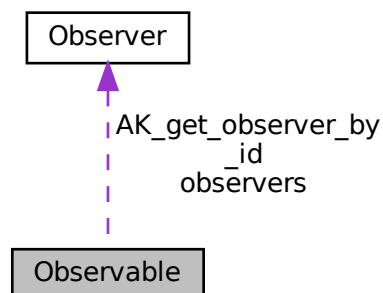
- trans/[transaction.h](#)

## 4.41 Observable Struct Reference

Structure defines functions for observable object.

```
#include <observable.h>
```

Collaboration diagram for Observable:



## Public Attributes

- [AK\\_observer](#) \* **observers** [[MAX\\_OBSERVABLE\\_OBSERVERS](#)]
- int **observer\_id\_counter**
- void \* **AK\_observable\_type**
- int **AK\_ObservableType\_Def**
- int(\* **AK\_destroy\_observable** )(struct [Observable](#) \*)
- int(\* **AK\_register\_observer** )(struct [Observable](#) \*, [AK\\_observer](#) \*)
- int(\* **AK\_unregister\_observer** )(struct [Observable](#) \*, [AK\\_observer](#) \*)
- int(\* **AK\_notify\_observer** )(struct [Observable](#) \*, [AK\\_observer](#) \*)
- int(\* **AK\_notify\_observers** )(struct [Observable](#) \*)
- int(\* **AK\_run\_custom\_action** )(void \*)
- [AK\\_observer](#) \*(\* **AK\_get\_observer\_by\_id** )(struct [Observable](#) \*, int id)

### 4.41.1 Detailed Description

Structure defines functions for observable object.

#### Author

Ivan Pusic

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

- auxi/[observable.h](#)

## 4.42 observable\_transaction Struct Reference

Structure which defines transaction observable type.

```
#include <transaction.h>
```

### 4.42.1 Detailed Description

Structure which defines transaction observable type.

#### Author

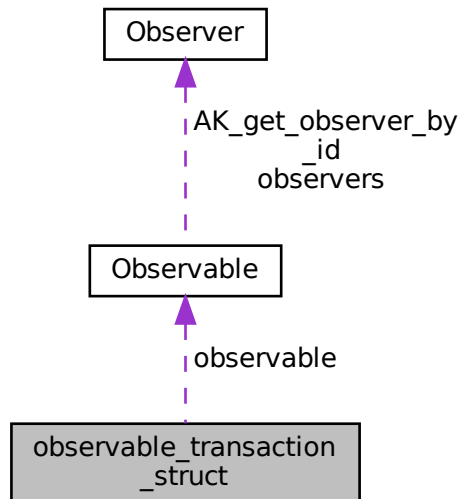
Ivan Pusic

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

- trans/[transaction.h](#)

### 4.43 observable\_transaction\_struct Struct Reference

Collaboration diagram for observable\_transaction\_struct:



#### Public Attributes

- `int(* AK_transaction_register_observer )(struct observable\_transaction\_struct *, AK\_observer *)`
- `int(* AK_transaction_unregister_observer )(struct observable\_transaction\_struct *, AK\_observer *)`
- `void(* AK_lock_released )()`
- `void(* AK_transaction_finished )()`
- `void(* AK_all_transactions_finished )()`
- `AK\_observable * observable`

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

- `trans/transaction.h`

### 4.44 Observer Struct Reference

Structure defines functions for observer object.

```
#include <observable.h>
```

#### Public Attributes

- `int observer_id`
- `void * AK_observer_type`
- `void(* AK_observer_type_event_handler )(void *, void *, AK\_ObservableType\_Enum)`
- `int(* AK_notify )(struct Observer *, void *observable_type, AK\_ObservableType\_Enum)`
- `int(* AK_destroy_observer )(struct Observer *)`

### 4.44.1 Detailed Description

Structure defines functions for observer object.

#### Author

Ivan Pusic

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

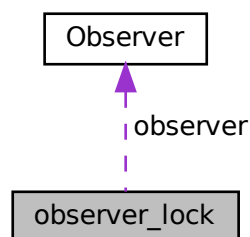
- auxi/[observable.h](#)

## 4.45 observer\_lock Struct Reference

Structure which defines transaction lock observer type.

```
#include <transaction.h>
```

Collaboration diagram for observer\_lock:



#### Public Attributes

- [AK\\_observer](#) \* **observer**

### 4.45.1 Detailed Description

Structure which defines transaction lock observer type.

#### Author

Ivan Pusic

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

- trans/[transaction.h](#)

## 4.46 root\_info Struct Reference

### Public Attributes

- int **root**
- int **level** [ORDER]

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

- file/idx/[btree.h](#)

## 4.47 search\_params Struct Reference

Structure that contains attribute name, lower and upper data value, special(NULL or \*) which is input for AK\_↔  
equisearch\_unsorted and AK\_rangearch\_unsorted.

```
#include <filesearch.h>
```

### Public Attributes

- char \* [szAttribute](#)  
*name of attribute*
- void \* [pData\\_lower](#)  
*pointer to lower value of search range*
- void \* [pData\\_upper](#)  
*pointer to upper value of search range*
- int [iSearchType](#)  
*if searching for NULL values, set to SEARCH\_NULL, all values -> SEARCH\_ALL, particular value -> SEARCH\_↔  
PARTICULAR, range of values -> SEARCH\_RANGE*

### 4.47.1 Detailed Description

Structure that contains attribute name, lower and upper data value, special(NULL or \*) which is input for AK\_↔  
equisearch\_unsorted and AK\_rangearch\_unsorted.

#### Author

Unknown

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

- file/[filesearch.h](#)



## 4.48 search\_result Struct Reference

Structure which represents search result of AK\_equisearch\_unsorted and AK\_rangesearch\_unsorted.

```
#include <filesearch.h>
```

### Public Attributes

- int \* [aiTuple\\_addresses](#)  
*array of tuple addresses*
- int \* [aiBlocks](#)  
*array of blocks to which the tuple addresses are relative*
- int [iNum\\_tuple\\_addresses](#)  
*number of tuple addresses/blocks in corresponding arrays*
- int \* [aiSearch\\_attributes](#)  
*array of indexes of searched-for attributes*
- int [iNum\\_search\\_attributes](#)  
*number of searched-for attributes in array*
- int [iNum\\_tuple\\_attributes](#)  
*number of attributes in tuple*

### 4.48.1 Detailed Description

Structure which represents search result of AK\_equisearch\_unsorted and AK\_rangesearch\_unsorted.

Author

Unknown

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

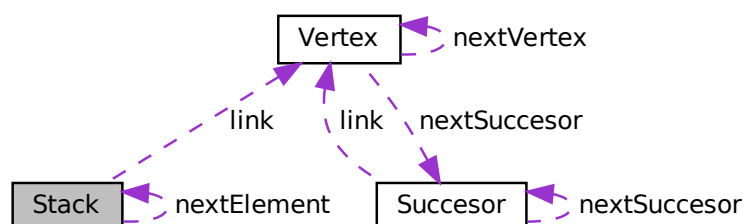
- file/[filesearch.h](#)

## 4.49 Stack Struct Reference

Structure defines a [Stack](#) element. Every [Stack](#) has its [Vertex](#) pointer and pointer to next [Stack](#) in the linked list.

```
#include <auxiliary.h>
```

Collaboration diagram for Stack:



## Public Attributes

- struct [Vertex](#) \* **link**
- struct [Stack](#) \* **nextElement**

### 4.49.1 Detailed Description

Structure defines a [Stack](#) element. Every [Stack](#) has its [Vertex](#) pointer and pointer to next [Stack](#) in the linked list.

#### Author

Frane Jakelić

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

- auxi/[auxiliary.h](#)

## 4.50 struct\_add Struct Reference

Structure defining node address.

```
#include <index.h>
```

## Public Attributes

- int [addBlock](#)  
*block address*
- int [indexTd](#)  
*index table destination*

### 4.50.1 Detailed Description

Structure defining node address.

#### Author

Unknown

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

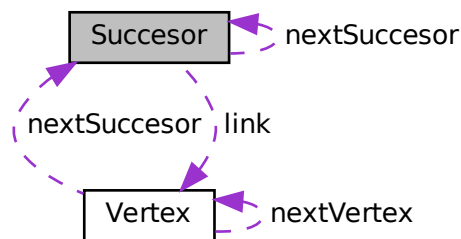
- file/idx/[index.h](#)

## 4.51 Succesor Struct Reference

Structure defines a [Succesor](#) element. Every [Succesor](#) has its [Vertex](#) pointer and pointer to next [Succesor](#) in the linked list.

```
#include <auxiliary.h>
```

Collaboration diagram for Succesor:



### Public Attributes

- struct [Vertex](#) \* `link`
- struct [Succesor](#) \* `nextSuccesor`

#### 4.51.1 Detailed Description

Structure defines a [Succesor](#) element. Every [Succesor](#) has its [Vertex](#) pointer and pointer to next [Succesor](#) in the linked list.

#### Author

Frane Jakelić

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

- [auxi/auxiliary.h](#)

## 4.52 table\_addresses Struct Reference

Structure that defines start and end address of extent.

```
#include <dbman.h>
```

## Public Attributes

- int [address\\_from](#) [MAX\_EXTENTS\_IN\_SEGMENT]  
*sturcture for extents start end stop adresses*
- int **address\_to** [MAX\_EXTENTS\_IN\_SEGMENT]

### 4.52.1 Detailed Description

Structure that defines start and end address of extent.

#### Author

Matija Novak

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

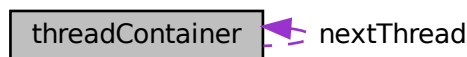
- [dm/dbman.h](#)

## 4.53 threadContainer Struct Reference

Structure that represents a linked list of threads.

```
#include <transaction.h>
```

Collaboration diagram for threadContainer:



## Public Attributes

- pthread\_t **thread**
- struct [threadContainer](#) \* **nextThread**

### 4.53.1 Detailed Description

Structure that represents a linked list of threads.

#### Author

Frane Jakelić

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

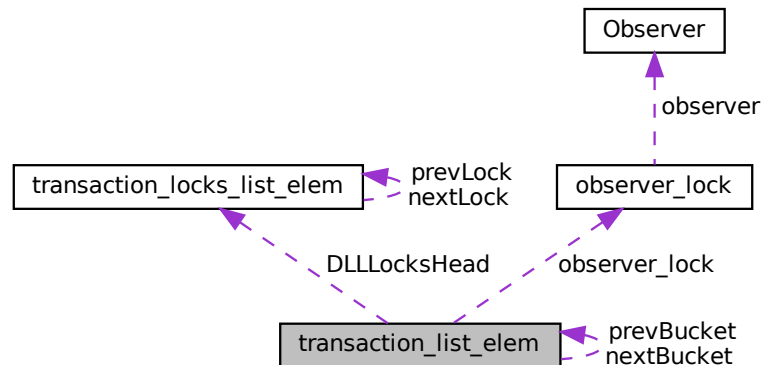
- [trans/transaction.h](#)

## 4.54 transaction\_list\_elem Struct Reference

Structure that represents LockTable entry about transaction lock holder. Element indexed by Hash table.

```
#include <transaction.h>
```

Collaboration diagram for transaction\_list\_elem:



### Public Attributes

- int **address**
- int **lock\_type**
- int **isWaiting**
- struct [transaction\\_locks\\_list\\_elem](#) \* **DLLLocksHead**
- struct [transaction\\_list\\_elem](#) \* **nextBucket**
- struct [transaction\\_list\\_elem](#) \* **prevBucket**
- [AK\\_observer\\_lock](#) \* **observer\_lock**

#### 4.54.1 Detailed Description

Structure that represents LockTable entry about transaction lock holder. Element indexed by Hash table.

#### Author

Frane Jakelić

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

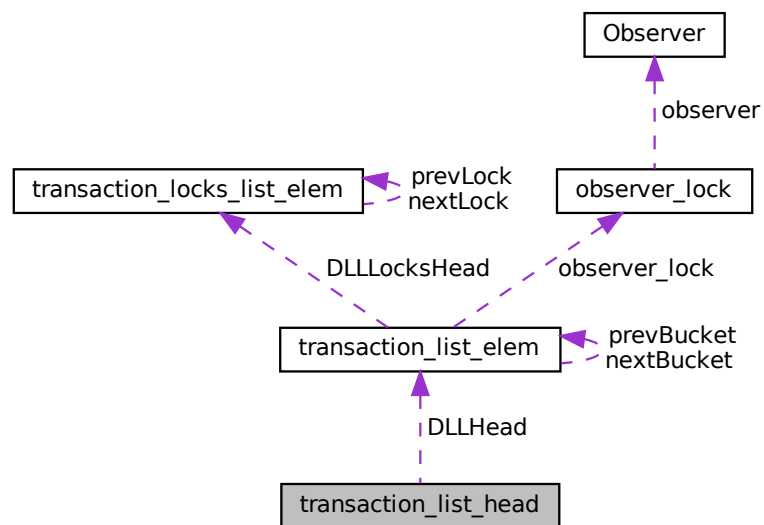
- [trans/transaction.h](#)

## 4.55 transaction\_list\_head Struct Reference

Structure that represents LockTable entry about doubly linked list of collision in Hash table.

```
#include <transaction.h>
```

Collaboration diagram for transaction\_list\_head:



### Public Attributes

- struct [transaction\\_list\\_elem](#) \* **DLLHead**

### 4.55.1 Detailed Description

Structure that represents LockTable entry about doubly linked list of collision in Hash table.

#### Author

Frane Jakelić

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

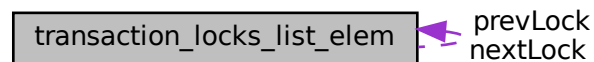
- trans/[transaction.h](#)

## 4.56 transaction\_locks\_list\_elem Struct Reference

Structure that represents LockTable entry about transaction resource lock.

```
#include <transaction.h>
```

Collaboration diagram for transaction\_locks\_list\_elem:



### Public Attributes

- pthread\_t **TransactionId**
- int **lock\_type**
- int **isWaiting**
- struct [transaction\\_locks\\_list\\_elem](#) \* **nextLock**
- struct [transaction\\_locks\\_list\\_elem](#) \* **prevLock**

### 4.56.1 Detailed Description

Structure that represents LockTable entry about transaction resource lock.

#### Author

Frane Jakelić

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

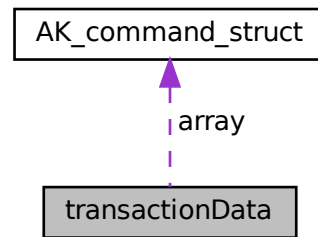
- trans/[transaction.h](#)

## 4.57 transactionData Struct Reference

Structure used to transport transaction data to the thread.

```
#include <transaction.h>
```

Collaboration diagram for transactionData:



### Public Attributes

- int **lengthOfArray**
- `command` \* **array**

#### 4.57.1 Detailed Description

Structure used to transport transaction data to the thread.

#### Author

Frane Jakelić

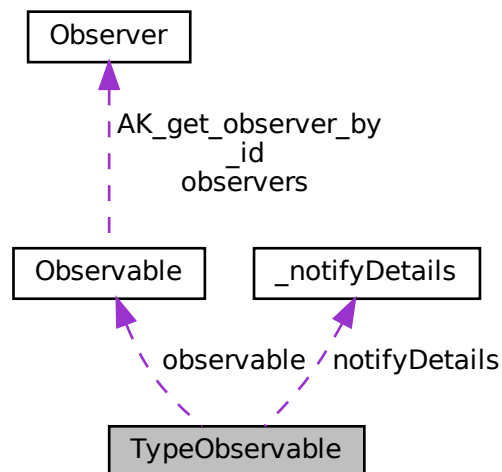
The documentation for this struct was generated from the following file:

- `trans/transaction.h`



## 4.58 TypeObservable Struct Reference

Collaboration diagram for TypeObservable:



### Public Attributes

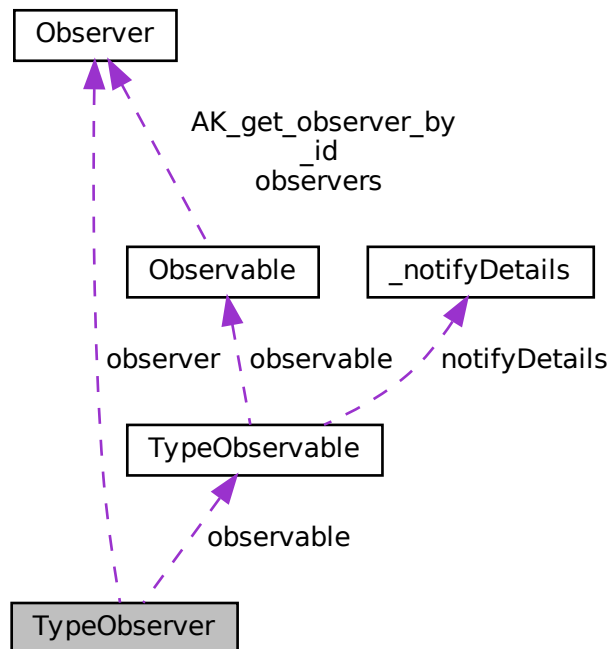
- [NotifyDetails](#) \* `notifyDetails`
- `char *(* AK_get_message )(struct TypeObservable *)`
- `int(* AK_custom_register_observer )(struct TypeObservable *, AK\_observer *)`
- `int(* AK_custom_unregister_observer )(struct TypeObservable *, AK\_observer *)`
- `void(* AK_set_notify_info_details )(struct TypeObservable *, NotifyType type, char *message)`
- [AK\\_observable](#) \* `observable`

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

- [auxi/observable.c](#)

## 4.59 TypeObserver Struct Reference

Collaboration diagram for TypeObserver:



### Public Attributes

- [AK\\_TypeObservable](#) \* **observable**
- [AK\\_observer](#) \* **observer**

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

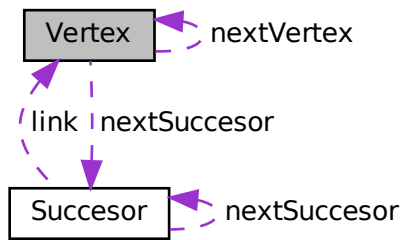
- [auxi/observable.c](#)

## 4.60 Vertex Struct Reference

Structure defines a [Vertex](#) node element. Every [Vertex](#) has its VertexId, index, lowLink and pointer to next edge and vertex.

```
#include <auxiliary.h>
```

Collaboration diagram for Vertex:



### Public Attributes

- int **vertexId**
- int **index**
- int **lowLink**
- struct `Succesor` \* **nextSuccesor**
- struct `Vertex` \* **nextVertex**

#### 4.60.1 Detailed Description

Structure defines a `Vertex` node element. Every `Vertex` has its `VertexId`, `index`, `lowLink` and pointer to next edge and vertex.

#### Author

Frane Jakelić

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

- `auxi/auxiliary.h`



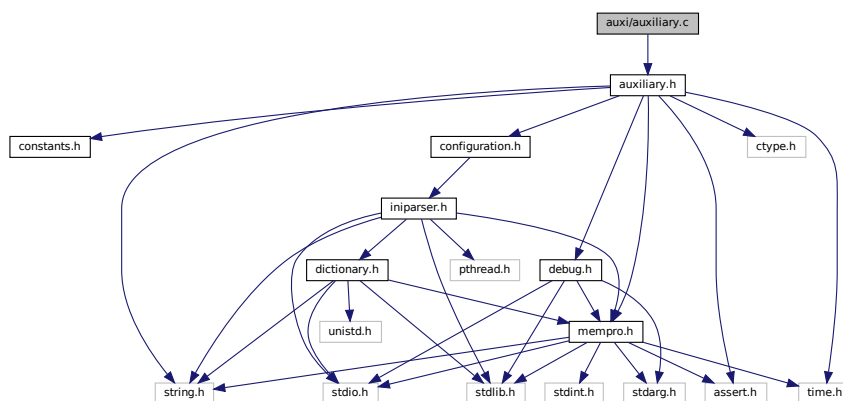
## Chapter 5

# File Documentation

### 5.1 auxi/auxiliary.c File Reference

```
#include "auxiliary.h"
```

Include dependency graph for auxiliary.c:



### Functions

- int [AK\\_chars\\_num\\_from\\_number](#) (int number, int base)  
*Function gets number of digits for given number.*
- size\_t [AK\\_type\\_size](#) (int iDB\_type, char \*szVarchar)  
*Function returns size in bytes of the provided database type.*
- int [AK\\_strcmp](#) (const void \*a, const void \*b)  
*Function compares to Strings.*
- void [Ak\\_Init\\_L3](#) (struct [list\\_node](#) \*\*L)  
*Function initializes empty list.*
- struct [list\\_node](#) \* [Ak\\_First\\_L2](#) (struct [list\\_node](#) \*L)  
*Function gets the first element of the list.*
- struct [list\\_node](#) \* [Ak\\_End\\_L2](#) (struct [list\\_node](#) \*L)  
*Function gets the last element of the list.*

- struct [list\\_node](#) \* [Ak\\_Next\\_L2](#) (struct [list\\_node](#) \*current)  
*Function gets the next element of the list.*
- struct [list\\_node](#) \* [Ak\\_Previous\\_L2](#) (struct [list\\_node](#) \*current, struct [list\\_node](#) \*L)  
*Function gets the previous element of the list.*
- unsigned int [Ak\\_IsEmpty\\_L2](#) (struct [list\\_node](#) \*L)  
*Function tests whether the list is empty.*
- void [Ak\\_InsertBefore\\_L2](#) (int type, char \*data, int size, struct [list\\_node](#) \*\*current, struct [list\\_node](#) \*\*L)  
*Function inserts new element before the current element in the list.*
- void [Ak\\_InsertAfter\\_L2](#) (int type, char \*data, int size, struct [list\\_node](#) \*\*current, struct [list\\_node](#) \*\*L)  
*Function inserts new element after the current element in the list.*
- void [Ak\\_InsertAtBegin\\_L3](#) (int type, char \*data, int size, struct [list\\_node](#) \*L)  
*Function inserts new element at the begin of the list. It uses function [AK\\_InsertBefore\\_L2](#).*
- void [Ak\\_InsertAtEnd\\_L3](#) (int type, char \*data, int size, struct [list\\_node](#) \*L)  
*Function inserts new element at the end of the list. It uses function [Ak\\_InsertAfter\\_L2](#).*
- void [Ak\\_Delete\\_L3](#) (struct [list\\_node](#) \*\*current, struct [list\\_node](#) \*\*L)  
*Function deletes the current element in the list.*
- void [Ak\\_DeleteAll\\_L3](#) (struct [list\\_node](#) \*\*L)  
*Function empties list.*
- int [Ak\\_Size\\_L2](#) (struct [list\\_node](#) \*L)  
*Function gets a number of the elements in the list.*
- char \* [Ak\\_Retrieve\\_L2](#) (struct [list\\_node](#) \*current, struct [list\\_node](#) \*L)  
*Function retrieves data from the current element in the list.*
- struct [list\\_node](#) \* [Ak\\_GetNth\\_L2](#) (int pos, struct [list\\_node](#) \*row)  
*Function gets a type of the current list element.*
- char \* [AK\\_get\\_array\\_perms](#) (char \*arr)  
*Function gets the position of given element.*
- [AK\\_vertex](#) [AK\\_search\\_vertex](#) (int id)
- [AK\\_vertex](#) [AK\\_search\\_empty\\_link](#) ()
- [AK\\_vertex](#) [AK\\_add\\_vertex](#) (int id)  
*Function that adds a new graph node.*
- [AK\\_succesor](#) [AK\\_add\\_succesor](#) (int succesorId, int succesorOf)  
*Creates a edge between two nodes.*
- [AK\\_stack](#) [AK\\_search\\_empty\\_stack\\_link](#) ([AK\\_stack](#) stackRoot)  
*Returns a empty link for the stack.*
- [AK\\_stack](#) [AK\\_push\\_to\\_stack](#) (int id)  
*Adds a entry to the stack.*
- [AK\\_stack](#) [AK\\_pop\\_from\\_stack](#) ()  
*Pops a entry to the stack.*
- [AK\\_stack](#) [AK\\_search\\_in\\_stack](#) (int id)  
*Finds an element in the stack.*
- int [MIN](#) (int X, int Y)
- void [AK\\_tarjan](#) (int id)  
*Tarjan algorithm that looks for a strongly connected component inside all subgraphs; using DFS.*
- void [AK\\_tarjan\\_test](#) ()
- [AK\\_synchronization\\_info](#) \* [AK\\_init\\_critical\\_section](#) ()  
*Initializes an [AK\\_synchronization\\_info](#) structure and returns an owned pointer that must later be passed to [AK\\_destroy\\_critical\\_section](#).*
- void [AK\\_destroy\\_critical\\_section](#) ([AK\\_synchronization\\_info](#) \*info)  
*Destroys a synchronization object when it is no longer necessary and frees the pointer.*
- void [AK\\_enter\\_critical\\_section](#) ([AK\\_synchronization\\_info](#) \*info)  
*Enters a critical section.*
- void [AK\\_leave\\_critical\\_section](#) ([AK\\_synchronization\\_info](#) \*info)  
*Leaves a critical section.*

## Variables

- [AK\\_graph](#) **G**
- [AK\\_stackHead](#) **S**
- int **indexCounter** = 0

### 5.1.1 Detailed Description

Provides auxiliary functions

### 5.1.2 Function Documentation

#### 5.1.2.1 AK\_add\_succesor()

```
AK_succesor AK_add_succesor (  
    int succesorId,  
    int succesorOf )
```

Creates a edge between two nodes.

#### Author

Frane Jakelić

#### Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>succesorId</i> | id of a newly created edge       |
| <i>succesorOf</i> | source of the newly created edge |

#### Returns

pointer to the newly created edge

#### 5.1.2.2 AK\_add\_vertex()

```
AK_vertex AK_add_vertex (  
    int id )
```

Function that adds a new graph node.

#### Author

Frane Jakelić

**Parameters**

|                  |                                      |
|------------------|--------------------------------------|
| <i>id</i>        | of the vertex that needs to be added |
| <i>graphRoot</i> | root node of the graph structure     |

**Returns**

pointer to the newly created node

**5.1.2.3 AK\_chars\_num\_from\_number()**

```
int AK_chars_num_from_number (
    int number,
    int base )
```

Function gets number of digits for given number.

**Author**

Dino Laktašić.

**Parameters**

|               |  |
|---------------|--|
| <i>number</i> | number to evaluate                     |
| <i>int</i>    | base mathematic base (e.g. 2, 10 etc.) |

**Returns**

number of digits for given number

**5.1.2.4 Ak\_Delete\_L3()**

```
void Ak_Delete_L3 (
    struct list_node ** current,
    struct list_node ** L )
```

Function deletes the current element in the list.

**Author**

Ljiljana Pintarić.



## Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>current</i> | current element in the list      |
| <i>L</i>       | root of the list No return value |

## 5.1.2.5 Ak\_DeleteAll\_L3()

```
void Ak_DeleteAll_L3 (
    struct list_node ** L )
```

Function empties list.

## Author

Ljiljana Pintarić.

## Parameters

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

## Returns

No return value

## 5.1.2.6 AK\_destroy\_critical\_section()

```
void AK_destroy_critical_section (
    AK_synchronization_info * info )
```

Destroys a synchronization object when it is no longer necessary and frees the pointer.

## Author

Marko Sinko

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>info</i> | Synchronization info structure |
|-------------|--------------------------------|

## Returns

void

#### 5.1.2.7 Ak\_End\_L2()

```
struct list_node* Ak_End_L2 (
    struct list_node * L )
```

Function gets the last element of the list.

##### Author

Ljiljana Pintarić.

##### Parameters

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

##### Returns

last element of the list

#### 5.1.2.8 AK\_enter\_critical\_section()

```
void AK_enter_critical_section (
    AK_synchronization_info * info )
```

Enters a critical section.

##### Author

Marko Sinko

##### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>info</i> | Synchronization info structure |
|-------------|--------------------------------|

##### Returns

void

#### 5.1.2.9 Ak\_First\_L2()

```
struct list_node* Ak_First_L2 (
    struct list_node * L )
```

Function gets the first element of the list.

##### Author

Ljiljana Pintarić.

**Parameters**

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

**Returns**

first element of the list

**5.1.2.10 AK\_get\_array\_perms()**

```
char* AK_get_array_perms (
    char * arr )
```

Function gets the position of given element.

**Author**

Matija Novak

**Parameters**

|                      |                                       |
|----------------------|---------------------------------------|
| <i>SearchElement</i> | element whose posititon we search for |
| <i>L</i>             | root of the list                      |

**Returns**

returns the posititon number of some elelemnt

**Author**

Dino Laktašić. Get all permutations without repetition (currently not used, but it can be helpful)

**Parameters**

|            |                           |
|------------|---------------------------|
| <i>arr</i> | array of chars to permute |
|------------|---------------------------|

**Returns**

char pointer to array of pointers to permuted char arrays

**5.1.2.11 Ak\_GetNth\_L2()**

```
struct list_node* Ak_GetNth_L2 (
    int pos,
    struct list_node * row )
```

Function gets a type of the current list element.

#### Author

Matija Šestak.

#### Parameters

|                |                      |
|----------------|----------------------|
| <i>current</i> | current list element |
| <i>L</i>       | root of the list     |

#### Returns

data type of the current list element

#### Author

Matija Šestak. Function gets a data size of the element

#### Parameters

|                |                      |
|----------------|----------------------|
| <i>current</i> | current list element |
| <i>L</i>       | - root of the list   |

#### Returns

data size of the current list element

#### Author

Ljiljana Pintarić Function for fetching nth element in row

#### Parameters

|            |                                  |
|------------|----------------------------------|
| <i>pos</i> | position of element in row       |
| <i>row</i> | list of elements of row in table |

#### Returns

element of list of elements of row in table

#### 5.1.2.12 AK\_init\_critical\_section()

```
AK_synchronization_info* AK_init_critical_section ( )
```

Initializes an [AK\\_synchronization\\_info](#) structure and returns an owned pointer that must later be passed to [AK\\_destroy\\_critical\\_section](#).

**Author**

Marko Sinko

**Returns**

Initialized synchronization object

**5.1.2.13 Ak\_Init\_L3()**

```
void Ak_Init_L3 (
    struct list_node ** L )
```

Function initializes empty list.

**Author**

Ljiljana Pintarić

**Parameters**

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

**Returns**

NO return value

**5.1.2.14 Ak\_InsertAfter\_L2()**

```
void Ak_InsertAfter_L2 (
    int type,
    char * data,
    int size,
    struct list_node ** current,
    struct list_node ** L )
```

Function inserts new element after the current element in the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|                |                             |
|----------------|-----------------------------|
| <i>data</i>    | new data                    |
| <i>current</i> | current element in the list |
| <i>L</i>       | root of the list            |

**Returns**

No return value.

**5.1.2.15 Ak\_InsertAtBegin\_L3()**

```
void Ak_InsertAtBegin_L3 (
    int type,
    char * data,
    int size,
    struct list_node * L )
```

Function inserts new element at the begin of the list. It uses function AK\_InsertBefore\_L.

**Author**

Ljiljana Pintarić.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>data</i> | new data         |
| <i>L</i>    | root of the list |

**Returns**

No return value

**5.1.2.16 Ak\_InsertAtEnd\_L3()**

```
void Ak_InsertAtEnd_L3 (
    int type,
    char * data,
    int size,
    struct list_node * L )
```

Function inserts new element at the end of the list. It uses function Ak\_InsertAfter\_L2.

**Author**

Ljiljana Pintarić.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>data</i> | new data         |
| <i>L</i>    | root of the list |

**Returns**

No return value.

**5.1.2.17 Ak\_InsertBefore\_L2()**

```
void Ak_InsertBefore_L2 (
    int type,
    char * data,
    int size,
    struct list_node ** current,
    struct list_node ** L )
```

Function inserts new element before the current element in the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|                |                             |
|----------------|-----------------------------|
| <i>data</i>    | new data                    |
| <i>current</i> | current element in the list |
| <i>L</i>       | root of the list            |

**Returns**

No return value

**5.1.2.18 Ak\_IsEmpty\_L2()**

```
unsigned int Ak_IsEmpty_L2 (
    struct list_node * L )
```

Function tests whether the list is empty.

**Author**

Ljiljana Pintarić.

**Parameters**

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

**Returns**

1 if the list is empty, otherwise returns 0

**5.1.2.19 AK\_leave\_critical\_section()**

```
void AK_leave_critical_section (
    AK_synchronization_info * info )
```

Leaves a critical section.

**Author**

Marko Sinko

**Parameters**

|             |                                |
|-------------|--------------------------------|
| <i>info</i> | Synchronization info structure |
|-------------|--------------------------------|

**Returns**

void

**5.1.2.20 Ak\_Next\_L2()**

```
struct list_node* Ak_Next_L2 (
    struct list_node * current )
```

Function gets the next element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|                |                             |
|----------------|-----------------------------|
| <i>current</i> | current element in the list |
|----------------|-----------------------------|

**Returns**

next element in the list



#### 5.1.2.21 AK\_pop\_from\_stack()

```
AK_stack AK_pop_from_stack ( )
```

Pops a entry to the stack.

##### Author

Frane Jakelić

##### Returns

pointer to the popped stack node

#### 5.1.2.22 Ak\_Previous\_L2()

```
struct list_node* Ak_Previous_L2 (
    struct list_node * current,
    struct list_node * L )
```

Function gets the previous element of the list.

##### Author

Ljiljana Pintarić.

##### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>current</i> | current element in the list |
| <i>L</i>       | root of the list            |

##### Returns

previous element in the list

#### 5.1.2.23 AK\_push\_to\_stack()

```
AK_stack AK_push_to_stack (
    int id )
```

Adds a entry to the stack.

##### Author

Frane Jakelić

**Parameters**

|           |   |
|-----------|---|
| <i>id</i> | of the element that is being added to the stack |
|-----------|---|

**Returns**

pointer to the newly added stack node

**5.1.2.24 Ak\_Retrieve\_L2()**

```
char* Ak_Retrieve_L2 (
    struct list_node * current,
    struct list_node * L )
```

Function retrieves data from the current element in the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|                |                             |
|----------------|-----------------------------|
| <i>current</i> | current element in the list |
| <i>L</i>       | root of the list            |

**Returns**

data from the list element

**5.1.2.25 AK\_search\_empty\_link()**

```
AK_vertex AK_search_empty_link ( )
```

**Author**

Frane Jakelić for empty link for a new graph node

**Parameters**

|                  |                                 |
|------------------|---------------------------------|
| <i>graphRoot</i> | oot node of the graph structure |
|------------------|---------------------------------|

**Returns**

empty link for new graph node

**5.1.2.26 AK\_search\_empty\_stack\_link()**

```
AK_stack AK_search_empty_stack_link (
    AK_stack stackRoot )
```

Returns a empty link for the stack.

**Author**

Frane Jakelić

**Parameters**

|                  |                                 |
|------------------|---------------------------------|
| <i>stackRoot</i> | root node of the selected stack |
|------------------|---------------------------------|

**Returns**

pointer to the empty link

**5.1.2.27 AK\_search\_in\_stack()**

```
AK_stack AK_search_in_stack (
    int id )
```

Finds an element in the stack.

**Author**

Frane Jakelić

**Parameters**

|           |  |
|-----------|--|
| <i>id</i> | of the node that needs to be found in the stac |
|-----------|--|

**Returns**

pointer to the found stack node

#### 5.1.2.28 AK\_search\_vertex()

```
AK_vertex AK_search_vertex (
    int id )
```

##### Author

Frane Jakelić that searches for a specific graph node by its ID

##### Parameters

|                  |                                      |
|------------------|--------------------------------------|
| <i>id</i>        | of the vertex that needs to be found |
| <i>graphRoot</i> | root node of the graph structure     |

##### Returns

found graph nod or null

#### 5.1.2.29 Ak\_Size\_L2()

```
int Ak_Size_L2 (
    struct list_node * L )
```

Function gets a number of the elements in the list.

##### Author

Ljiljana Pintarić.

##### Parameters

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

##### Returns

Size of the list

#### 5.1.2.30 AK\_strcmp()

```
int AK_strcmp (
    const void * a,
    const void * b )
```

Function compares to Strings.

**Author**

Dino Laktašić

**Parameters**

|           |                             |
|-----------|-----------------------------|
| <i>*a</i> | pointer of value to compare |
| <i>*b</i> | pointer of value to compare |

**Returns**

result of comparison according to strcmp function

**5.1.2.31 AK\_tarjan()**

```
void AK_tarjan (
    int id )
```

Tarjan algorithm that looks for a strongly connected component inside all subgraphs; using DFS.

**Author**

Frane Jakelić

**Parameters**

|           |  |
|-----------|--|
| <i>id</i> | of the element on which the algorithm looks for a id of a strongly connected component |
|-----------|--|

```
#### connected component. Edges:");
```

```
####");
```

**5.1.2.32 AK\_type\_size()**

```
size_t AK_type_size (
    int iDB_type,
    char * szVarchar )
```

Function returns size in bytes of the provided database type.

**Author**

Miroslav Policki



Structure defines a [Vertex](#) node element. Every [Vertex](#) has its [VertexId](#), [index](#), [lowLink](#) and pointer to next edge and vertex.

- struct [Successor](#)

Structure defines a [Successor](#) element. Every [Successor](#) has its [Vertex](#) pointer and pointer to next [Successor](#) in the linked list.

- struct [Stack](#)

Structure defines a [Stack](#) element. Every [Stack](#) has its [Vertex](#) pointer and pointer to next [Stack](#) in the linked list.

- struct [AK\\_synchronization\\_info](#)

Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex).

## Macros

- #define **MAX\_LOOP\_ITERATIONS** 1000
- #define **TBL\_BOX\_OFFSET** 1

## Typedefs

- typedef struct [list\\_node](#) **AK\_list**
- typedef struct [list\\_node](#) \* **AK\_list\_elem**
- typedef struct [Vertex](#) **AK\_graph**
- typedef struct [Successor](#) \* **AK\_successor**
- typedef struct [Vertex](#) \* **AK\_vertex**
- typedef struct [Stack](#) \* **AK\_stack**
- typedef struct [Stack](#) **AK\_stackHead**

## Functions

- int [AK\\_strcmp](#) (const void \*a, const void \*b)  
*Function compares to Strings.*
- int [AK\\_chars\\_num\\_from\\_number](#) (int number, int base)  
*Function gets number of digits for given number.*
- size\_t [AK\\_type\\_size](#) (int iDB\_type, char \*szVarchar)  
*Function returns size in bytes of the provided database type.*
- void [Ak\\_Init\\_L3](#) (struct [list\\_node](#) \*\*L)  
*Function initializes empty list.*
- struct [list\\_node](#) \* [Ak\\_First\\_L2](#) (struct [list\\_node](#) \*L)  
*Function gets the first element of the list.*
- struct [list\\_node](#) \* [Ak\\_End\\_L2](#) (struct [list\\_node](#) \*L)  
*Function gets the last element of the list.*
- struct [list\\_node](#) \* [Ak\\_Next\\_L2](#) (struct [list\\_node](#) \*current)  
*Function gets the next element of the list.*
- struct [list\\_node](#) \* [Ak\\_Previous\\_L2](#) (struct [list\\_node](#) \*current, struct [list\\_node](#) \*L)  
*Function gets the previous element of the list.*
- unsigned int [Ak\\_IsEmpty\\_L2](#) (struct [list\\_node](#) \*L)  
*Function tests whether the list is empty.*
- void [Ak\\_InsertBefore\\_L2](#) (int type, char \*data, int size, struct [list\\_node](#) \*\*current, struct [list\\_node](#) \*\*L)  
*Function inserts new element before the current element in the list.*
- void [Ak\\_InsertAfter\\_L2](#) (int type, char \*data, int size, struct [list\\_node](#) \*\*current, struct [list\\_node](#) \*\*L)  
*Function inserts new element after the current element in the list.*

- void [Ak\\_InsertAtBegin\\_L3](#) (int type, char \*data, int size, struct [list\\_node](#) \*L)  
*Function inserts new element at the begin of the list. It uses function [AK\\_InsertBefore\\_L](#).*
- void [Ak\\_InsertAtEnd\\_L3](#) (int type, char \*data, int size, struct [list\\_node](#) \*L)  
*Function inserts new element at the end of the list. It uses function [Ak\\_InsertAfter\\_L2](#).*
- void [Ak\\_Delete\\_L3](#) (struct [list\\_node](#) \*\*current, struct [list\\_node](#) \*\*L)  
*Function deletes the current element in the list.*
- void [Ak\\_DeleteAll\\_L3](#) (struct [list\\_node](#) \*\*L)  
*Function empties list.*
- int [Ak\\_Size\\_L2](#) (struct [list\\_node](#) \*L)  
*Function gets a number of the elements in the list.*
- char \* [Ak\\_Retrieve\\_L2](#) (struct [list\\_node](#) \*current, struct [list\\_node](#) \*L)  
*Function retrieves data from the current element in the list.*
- struct [list\\_node](#) \* [Ak\\_GetNth\\_L2](#) (int pos, struct [list\\_node](#) \*row)  
*Function gets a type of the current list element.*
- char \* [AK\\_get\\_array\\_perms](#) (char \*arr)  
*Function gets the position of given element.*
- [AK\\_vertex AK\\_search\\_vertex](#) (int id)
- [AK\\_vertex AK\\_search\\_empty\\_link](#) ()
- [AK\\_vertex AK\\_add\\_vertex](#) (int id)  
*Function that adds a new graph node.*
- [AK\\_succesor AK\\_add\\_succesor](#) (int succesorId, int succesorOf)  
*Creates a edge between two nodes.*
- [AK\\_stack AK\\_search\\_empty\\_stack\\_link](#) ([AK\\_stack](#) stackRoot)  
*Returns a empty link for the stack.*
- [AK\\_stack AK\\_push\\_to\\_stack](#) (int id)  
*Adds a entry to the stack.*
- [AK\\_stack AK\\_pop\\_from\\_stack](#) ()  
*Pops a entry to the stack.*
- [AK\\_stack AK\\_search\\_in\\_stack](#) (int id)  
*Finds an element in the stack.*
- int [MIN](#) (int X, int Y)
- void [AK\\_tarjan](#) (int id)  
*Tarjan algorithm that looks for a strongly connected component inside all subgraphs; using DFS.*
- void [AK\\_tarjan\\_test](#) ()
- [AK\\_synchronization\\_info](#) \* [AK\\_init\\_critical\\_section](#) ()  
*Initializes an [AK\\_synchronization\\_info](#) structure and returns an owned pointer that must later be passed to [AK\\_↔destroy\\_critical\\_section](#).*
- void [AK\\_destroy\\_critical\\_section](#) ([AK\\_synchronization\\_info](#) \*info)  
*Destroys a synchronization object when it is no longer necessary and frees the pointer.*
- void [AK\\_enter\\_critical\\_section](#) ([AK\\_synchronization\\_info](#) \*info)  
*Enters a critical section.*
- void [AK\\_leave\\_critical\\_section](#) ([AK\\_synchronization\\_info](#) \*info)  
*Leaves a critical section.*

## Variables

- int [testMode](#)  
*You can turn testMode on or off with [TEST\\_MODE\\_ON](#) and [TEST\\_MODE\\_OFF](#). To do this, simply enable or disable it in YOUR function (not in any other!) Test mode can be used when you need some special cases in your functions (i.e., when you are testing some functionality, which doesn't apply in normal conditions). But don't forget to turn this mode off, after you are done (within test function for example)!*



### 5.2.1 Detailed Description

Header file zhat provides data structures for the auxiliary functions

### 5.2.2 Function Documentation

#### 5.2.2.1 AK\_add\_succesor()

```
AK_succesor AK_add_succesor (
    int succesorId,
    int succesorOf )
```

Creates a edge between two nodes.

##### Author

Frane Jakelić

##### Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>succesorId</i> | id of a newly created edge       |
| <i>succesorOf</i> | source of the newly created edge |

##### Returns

pointer to the newly created edge

#### 5.2.2.2 AK\_add\_vertex()

```
AK_vertex AK_add_vertex (
    int id )
```

Function that adds a new graph node.

##### Author

Frane Jakelić

##### Parameters

|                  |                                      |
|------------------|--------------------------------------|
| <i>id</i>        | of the vertex that needs to be added |
| <i>graphRoot</i> | root node of the graph structure     |

**Returns**

pointer to the newly created node

**5.2.2.3 AK\_chars\_num\_from\_number()**

```
int AK_chars_num_from_number (
    int number,
    int base )
```

Function gets number of digits for given number.

**Author**

Dino Laktašić.

**Parameters**

|               |  |
|---------------|--|
| <i>number</i> | number to evaluate                     |
| <i>int</i>    | base mathematic base (e.g. 2, 10 etc.) |

**Returns**

number of digits for given number

**5.2.2.4 Ak\_Delete\_L3()**

```
void Ak_Delete_L3 (
    struct list_node ** current,
    struct list_node ** L )
```

Function deletes the current element in the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|                |                                  |
|----------------|----------------------------------|
| <i>current</i> | current element in the list      |
| <i>L</i>       | root of the list No return value |

### 5.2.2.5 Ak\_DeleteAll\_L3()

```
void Ak_DeleteAll_L3 (
    struct list_node ** L )
```

Function empties list.

#### Author

Ljiljana Pintarić.

#### Parameters

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

#### Returns

No return value

### 5.2.2.6 AK\_destroy\_critical\_section()

```
void AK_destroy_critical_section (
    AK_synchronization_info * info )
```

Destroys a synchronization object when it is no longer necessary and frees the pointer.

#### Author

Marko Sinko

#### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>info</i> | Synchronization info structure |
|-------------|--------------------------------|

#### Returns

void

### 5.2.2.7 Ak\_End\_L2()

```
struct list_node* Ak_End_L2 (
    struct list_node * L )
```

Function gets the last element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

**Returns**

last element of the list

**5.2.2.8 AK\_enter\_critical\_section()**

```
void AK_enter_critical_section (
    AK_synchronization_info * info )
```

Enters a critical section.

**Author**

Marko Sinko

**Parameters**

|             |                                |
|-------------|--------------------------------|
| <i>info</i> | Synchronization info structure |
|-------------|--------------------------------|

**Returns**

void

**5.2.2.9 Ak\_First\_L2()**

```
struct list_node* Ak_First_L2 (
    struct list_node * L )
```

Function gets the first element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

**Returns**

first element of the list

**5.2.2.10 AK\_get\_array\_perms()**

```
char* AK_get_array_perms (
    char * arr )
```

Function gets the position of given element.

**Author**

Matija Novak

**Parameters**

|                      |                                     |
|----------------------|-------------------------------------|
| <i>SearchElement</i> | element whose positon we search for |
| <i>L</i>             | root of the list                    |

**Returns**

returns the posititon number of some elelemnt

**Author**

Dino Laktašić. Get all permutations without repetition (currently not used, but it can be helpful)

**Parameters**

|            |                           |
|------------|---------------------------|
| <i>arr</i> | array of chars to permute |
|------------|---------------------------|

**Returns**

char pointer to array of pointers to permuted char arrays

**5.2.2.11 Ak\_GetNth\_L2()**

```
struct list_node* Ak_GetNth_L2 (
    int pos,
    struct list_node * row )
```

Function gets a type of the current list element.

**Author**

Matija Šestak.

## Parameters

|                |                      |
|----------------|----------------------|
| <i>current</i> | current list element |
| <i>L</i>       | root of the list     |

## Returns

data type of the current list element

## Author

Matija Šestak. Function gets a data size of the element

## Parameters

|                |                      |
|----------------|----------------------|
| <i>current</i> | current list element |
| <i>L</i>       | - root of the list   |

## Returns

data size of the current list element

## Author

Ljiljana Pintarić Function for fetching nth element in row

## Parameters

|            |                                  |
|------------|----------------------------------|
| <i>pos</i> | position of element in row       |
| <i>row</i> | list of elements of row in table |

## Returns

element of list of elements of row in table

## 5.2.2.12 AK\_init\_critical\_section()

```
AK_synchronization_info* AK_init_critical_section ( )
```

Initializes an [AK\\_synchronization\\_info](#) structure and returns an owned pointer that must later be passed to [AK\\_destroy\\_critical\\_section](#).

## Author

Marko Sinko

## Returns

Initialized synchronization object

### 5.2.2.13 Ak\_Init\_L3()

```
void Ak_Init_L3 (
    struct list_node ** L )
```

Function initializes empty list.

#### Author

Ljiljana Pintarić

#### Parameters

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

#### Returns

NO return value

### 5.2.2.14 Ak\_InsertAfter\_L2()

```
void Ak_InsertAfter_L2 (
    int type,
    char * data,
    int size,
    struct list_node ** current,
    struct list_node ** L )
```

Function inserts new element after the current element in the list.

#### Author

Ljiljana Pintarić.

#### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>data</i>    | new data                    |
| <i>current</i> | current element in the list |
| <i>L</i>       | root of the list            |

#### Returns

No return value.



### 5.2.2.15 Ak\_InsertAtBegin\_L3()

```
void Ak_InsertAtBegin_L3 (
    int type,
    char * data,
    int size,
    struct list_node * L )
```

Function inserts new element at the begin of the list. It uses function AK\_InsertBefore\_L.

#### Author

Ljiljana Pintarić.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>data</i> | new data         |
| <i>L</i>    | root of the list |

#### Returns

No return value

### 5.2.2.16 Ak\_InsertAtEnd\_L3()

```
void Ak_InsertAtEnd_L3 (
    int type,
    char * data,
    int size,
    struct list_node * L )
```

Function inserts new element at the end of the list. It uses function Ak\_InsertAfter\_L2.

#### Author

Ljiljana Pintarić.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>data</i> | new data         |
| <i>L</i>    | root of the list |

#### Returns

No return value.

### 5.2.2.17 Ak\_InsertBefore\_L2()

```
void Ak_InsertBefore_L2 (
    int type,
    char * data,
    int size,
    struct list_node ** current,
    struct list_node ** L )
```

Function inserts new element before the current element in the list.

#### Author

Ljiljana Pintarić.

#### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>data</i>    | new data                    |
| <i>current</i> | current element in the list |
| <i>L</i>       | root of the list            |

#### Returns

No return value

### 5.2.2.18 Ak\_IsEmpty\_L2()

```
unsigned int Ak_IsEmpty_L2 (
    struct list_node * L )
```

Function tests whether the list is empty.

#### Author

Ljiljana Pintarić.

#### Parameters

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

#### Returns

1 if the list is empty, otherwise returns 0

### 5.2.2.19 AK\_leave\_critical\_section()

```
void AK_leave_critical_section (
    AK_synchronization_info * info )
```

Leaves a critical section.

#### Author

Marko Sinko

#### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>info</i> | Synchronization info structure |
|-------------|--------------------------------|

#### Returns

void

### 5.2.2.20 Ak\_Next\_L2()

```
struct list_node* Ak_Next_L2 (
    struct list_node * current )
```

Function gets the next element of the list.

#### Author

Ljiljana Pintarić.

#### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>current</i> | current element in the list |
|----------------|-----------------------------|

#### Returns

next element in the list

### 5.2.2.21 AK\_pop\_from\_stack()

```
AK_stack AK_pop_from_stack ( )
```

Pops a entry to the stack.

**Author**

Frane Jakelić

**Returns**

pointer to the popped stack node

**5.2.2.22 Ak\_Previous\_L2()**

```
struct list_node* Ak_Previous_L2 (
    struct list_node * current,
    struct list_node * L )
```

Function gets the previous element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|                |                             |
|----------------|-----------------------------|
| <i>current</i> | current element in the list |
| <i>L</i>       | root of the list            |

**Returns**

previous element in the list

**5.2.2.23 AK\_push\_to\_stack()**

```
AK_stack AK_push_to_stack (
    int id )
```

Adds a entry to the stack.

**Author**

Frane Jakelić

**Parameters**

|           |   |
|-----------|---|
| <i>id</i> | of the element that is being added to the stack |
|-----------|---|

**Returns**

pointer to the newly added stack node

**5.2.2.24 Ak\_Retrieve\_L2()**

```
char* Ak_Retrieve_L2 (
    struct list_node * current,
    struct list_node * L )
```

Function retrieves data from the current element in the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|                |                             |
|----------------|-----------------------------|
| <i>current</i> | current element in the list |
| <i>L</i>       | root of the list            |

**Returns**

data from the list element

**5.2.2.25 AK\_search\_empty\_link()**

```
AK_vertex AK_search_empty_link ( )
```

**Author**

Frane Jakelić for empty link for a new graph node

**Parameters**

|                  |                                 |
|------------------|---------------------------------|
| <i>graphRoot</i> | oot node of the graph structure |
|------------------|---------------------------------|

**Returns**

empty link for new graph node

#### 5.2.2.26 AK\_search\_empty\_stack\_link()

```
AK_stack AK_search_empty_stack_link (
    AK_stack stackRoot )
```

Returns a empty link for the stack.

##### Author

Frane Jakelić

##### Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>stackRoot</i> | root node of the selected stack |
|------------------|---------------------------------|

##### Returns

pointer to the empty link

#### 5.2.2.27 AK\_search\_in\_stack()

```
AK_stack AK_search_in_stack (
    int id )
```

Finds an element in the stack.

##### Author

Frane Jakelić

##### Parameters

|           |   |
|-----------|---|
| <i>id</i> | of the node that needs to be found in the stack |
|-----------|---|

##### Returns

pointer to the found stack node

#### 5.2.2.28 AK\_search\_vertex()

```
AK_vertex AK_search_vertex (
    int id )
```

##### Author

Frane Jakelić that searches for a specific graph node by its ID

**Parameters**

|                  |                                      |
|------------------|--------------------------------------|
| <i>id</i>        | of the vertex that needs to be found |
| <i>graphRoot</i> | root node of the graph structure     |

**Returns**

found graph nod or null

**5.2.2.29 Ak\_Size\_L2()**

```
int Ak_Size_L2 (
    struct list_node * L )
```

Function gets a number of the elements in the list.

**Author**

Ljiljana Pintarić.

**Parameters**

|          |                  |
|----------|------------------|
| <i>L</i> | root of the list |
|----------|------------------|

**Returns**

Size of the list

**5.2.2.30 AK\_strcmp()**

```
int AK_strcmp (
    const void * a,
    const void * b )
```

Function compares to Strings.

**Author**

Dino Laktašić

**Parameters**

|           |                             |
|-----------|-----------------------------|
| <i>*a</i> | pointer of value to compare |
| <i>*b</i> | pointer of value to compare |

**Returns**

result of comparison according to strcmp function

**5.2.2.31 AK\_tarjan()**

```
void AK_tarjan (
    int id )
```

Tarjan algorithm that looks for a strongly connected component inside all subgraphs; using DFS.

**Author**

Frane Jakelić

**Parameters**

|           |  |
|-----------|--|
| <i>id</i> | of the element on which the algorithm looks for a id of a strongly connected component |
|-----------|--|

```
#### connected component. Edges:");
```

```
####");
```

**5.2.2.32 AK\_type\_size()**

```
size_t AK_type_size (
    int iDB_type,
    char * szVarchar )
```

Function returns size in bytes of the provided database type.

**Author**

Miroslav Policki

**Parameters**

|                  |  |
|------------------|--|
| <i>iDB_type</i>  | database data type (defined in <a href="#">constants.h</a> )         |
| <i>szVarchar</i> | if iDB_type == TYPE_VARCHAR, pointer to the string, otherwise unused |

**Returns**

size of provided data type in bytes if provided data type is valid, else 0



### 5.2.3 Variable Documentation

#### 5.2.3.1 testMode

testMode

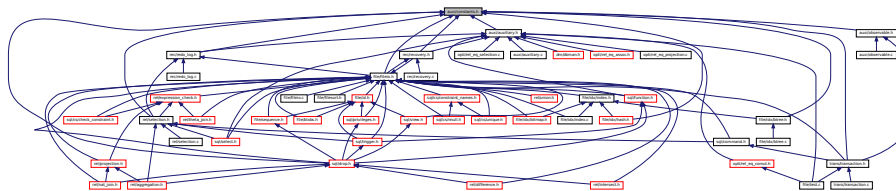
You can turn testMode on or off with TEST\_MODE\_ON and TEST\_MODE\_OFF. To do this, simply enable or disable it in YOUR function (not in any other!) Test mode can be used when you need some special cases in your functions (i.e., when you are testing some functionality, which doesn't apply in normal conditions). But don't forget to turn this mode off, after you are done (within test function for example)!

Author

Domagoj Šitum

## 5.3 auxi/constants.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define MAX_VARCHAR_LENGTH 200`  
Constant declaring maximum length of varchar data value.
- `#define MAX_ATTRIBUTES 10`  
Constant declaring maximum number of attributes per block.
- `#define MAX_ATT_NAME 255`  
Constant declaring maximum length of attribute name string (used in AK\_header->att\_name)
- `#define MAX_CONSTRAINTS 5`  
Constant declaring maximum number of constraints per attribute.
- `#define MAX_CONSTR_NAME 255`  
Constant declaring maximum length of constraint name string (used in AK\_header->constr\_name)
- `#define MAX_CONSTR_CODE 255`  
Constant declaring maximum length of constraint code string.
- `#define MAX_OBSERVABLE_OBSERVERS 4096`  
Constant for declaring maximum number of observers objects for some observable type.
- `#define MAX_ACTIVE_TRANSACTIONS_COUNT 100`  
Constant for declaring maximum number of active transactions in DBMS.
- `#define DATA_BLOCK_SIZE 500`  
Constant declaring length of data block size (used in AK\_block->data)

- `#define DATA_ENTRY_SIZE 10`  
*Constant declaring lenght of data entry in sizeof( int )*
- `#define MAX_QUERY_LIB_MEMORY 255`  
*Constant declaring maximum size of query lib memory.*
- `#define MAX_CACHE_MEMORY 255`  
*Constant declaring maximum size of DB cache memory.*
- `#define MAX_QUERY_DICT_MEMORY 255`  
*Constant declaring maximum size of query dictionary memory.*
- `#define MAX_QUERY_RESULT_MEMORY 255`  
*Constant declaring maximum size of query result cache memory.*
- `#define MAX_TOKENS 255`  
*Constant declaring maximum number of attributes to handle in relation equivalence function.*
- `#define MAX_MAIN_BUCKETS 512`  
*Constant declaring the maximum number of main buckets.*
- `#define MAIN_BUCKET_SIZE 4`  
*Constant declaring the size of main buckets.*
- `#define HASH_BUCKET_SIZE 4`  
*Constant declaring the size of hash buckets.*
- `#define NUMBER_OF_KEYS 4096`  
*Constant declaring number of buckets in hash table.*
- `#define EXIT_SUCCESS 0`  
*Constant declaring successfull exit.*
- `#define EXIT_ERROR -1`  
*Constant declaring unsuccessfull exit.*
- `#define EXIT_WARNING -2`
- `#define BLOCK_TYPE_FREE -1`  
*Constant declaring AK\_free block type (used in AK\_block->type)*
- `#define BLOCK_TYPE_NORMAL 0`  
*Constant declaring normal block type e.g. used by some extent (used in AK\_block->type)*
- `#define BLOCK_TYPE_CHAINED 1`  
*Constant declaring chained block type e.g. used if the block is chained with another (used in AK\_block->type)*
- `#define NOT_CHAINED -1`  
*Constant used in AK\_block->chained\_with if the block isn't chained.*
- `#define FREE_INT -10`  
*Constant declaring dummy data for empty integers.*
- `#define FREE_CHAR '\0'`  
*Constant declaring dummy data for empty chars.*
- `#define SEGMENT_TYPE_SYSTEM_TABLE 0`  
*Constant declaring system table segment type (used in system catalog)*
- `#define SEGMENT_TYPE_TABLE 1`  
*Constant declaring table segment type (used in system catalog)*
- `#define SEGMENT_TYPE_INDEX 2`  
*Constant declaring index segment type (used in system catalog)*
- `#define SEGMENT_TYPE_TRANSACTION 3`  
*Constant declaring transaction segment type (used in system catalog)*
- `#define SEGMENT_TYPE_TEMP 4`  
*Constant declaring temporary segment type (used in system catalog)*
- `#define TYPE_INTERNAL 0`  
*Constant declaring internal data type (used in AK\_header->type and AK\_tuple\_dict->type)*
- `#define TYPE_INT 1`  
*integer data type (used in AK\_header->type and AK\_tuple\_dict->type)*

- `#define TYPE_FLOAT 2`  
*Constant declaring float data type (used in AK\_header->type and AK\_tuple\_dict->type)*
- `#define TYPE_NUMBER 3`  
*Constant declaring number data type (used in AK\_header->type and AK\_tuple\_dict->type)*
- `#define TYPE_VARCHAR 4`  
*Constant declaring varchar data type (used in AK\_header->type and AK\_tuple\_dict->type)*
- `#define TYPE_DATE 5`  
*Constant declaring date data type (used in AK\_header->type and AK\_tuple\_dict->type)*
- `#define TYPE_DATETIME 6`  
*datetime data type (used in AK\_header->type and AK\_tuple\_dict->type)*
- `#define TYPE_TIME 7`  
*Constant declaring time data type (used in AK\_header->type and AK\_tuple\_dict->type)*
- `#define TYPE_BLOB 8`  
*blob data type (used in AK\_header->type and AK\_tuple\_dict->type)*
- `#define TYPE_BOOL 9`  
*Constant declaring boolean data type (used in AK\_header->type and AK\_tuple\_dict->type)*
- `#define TYPE_OPERAND 10`  
*Constant indicating operand in AK\_list.*
- `#define TYPE_OPERATOR 11`  
*indicates operator in AK\_list*
- `#define TYPE_ATTRIBS 12`  
*Constant indicating attribute/s in AK\_list.*
- `#define TYPE_CONDITION 13`  
*Constant indicating condition in AK\_list.*
- `#define BLOCK_CLEAN 0`  
*Constant indicating block cleaning (not changed since read from disk)*
- `#define BLOCK_DIRTY 1`  
*Constant indicating dirty block (changed since read from disk, has to be written)*
- `#define ATTR_DELIMITER ","`  
*Constant declaring attributes delimiter.*
- `#define ATTR_ESCAPE ""`  
*Constant indicating attributes escape section.*
- `#define NULLL "asdfgXYZ"`  
*Constant declaring null value for tables.*
- `#define RO_SELECTION 's'`
- `#define RO_PROJECTION 'p'`
- `#define RO_NAT_JOIN 'n'`
- `#define RO_RENAME 'r'`
- `#define RO_UNION 'u'`
- `#define RO_INTERSECT 'i'`
- `#define RO_EXCEPT 'e'`
- `#define RO_THETA_JOIN 't'`
- `#define NEW_VALUE 0`  
*Constant indicating that data is new value.*
- `#define SEARCH_CONSTRAINT 1`  
*Constant indicating that data is constraint to search for.*
- `#define UPDATE 0`  
*Constant indicating that operation to be performed is update.*
- `#define DELETE 1`  
*Constant indicating that operation to be performed is delete.*
- `#define INSERT 2`

- Constant indicating that operation to be performed is insert.*

  - #define **SELECT** 3

*Constant indicating select operation.*
- #define **FIND** 2

*Constant indicating that operation to be performed is searching.*
- #define **INFO\_BUCKET** 0

*Constant declaring type of bucket as "info bucket" when inserting bucket to block.*
- #define **MAIN\_BUCKET** 1

*Constant declaring type of bucket as "main bucket" when inserting bucket to block.*
- #define **HASH\_BUCKET** 2

*Constant declaring type of bucket as "hash bucket" when inserting bucket to block.*
- #define **SHARED\_LOCK** 0

*Constant declaring type of lock as SHARED LOCK.*
- #define **EXCLUSIVE\_LOCK** 1

*Constant declaring type of lock as EXCLUSIVE LOCK.*
- #define **WAIT\_FOR\_UNLOCK** 0

*Constant declaring that a lock has to wait until other locks release the resource.*
- #define **PASS\_LOCK\_QUEUE** 1

*Constant declaring that a lock can acquire the resource AK\_freely.*
- #define **OK** 1

*Constant declaring that the method is completed successfully.*
- #define **NOT\_OK** 0

*Constant declaring that the method isn't completed successfully.*
- #define **COMMIT** 1

*Constant declaring that the transaction is completed successfully.*
- #define **ABORT** 0

*Constant declaring if the transaction is being aborted.*
- #define **NEW\_ID** 0

*Constant declaring if new obj\_id should be created.*
- #define **MAX\_BLOCKS\_CURRENTLY\_ACCESSED** 32

*Indicates maximum number of threads that can access (read or write) database at the same time.*
- #define **TEST\_MODE\_ON** 1

*This constant is used to turn testMode (auxi/auxillary.h) ON.*
- #define **TEST\_MODE\_OFF** 0

*This constant is used to turn testMode (auxi/auxillary.h) OFF.*
- #define **SEPARATOR** "[{(|&&|)}]"

*Used in [unique.c](#) for separation of names of attributes and their values when UNIQUE constraint is being set or tested on combination of values of attributes.*
- #define **AK\_CONSTRAINTS\_BEWTEEN** "AK\_constraints\_between"

*Defines system table name for storing between constraints.*
- #define **AK\_CONSTRAINTS\_CHECK\_CONSTRAINT** "AK\_constraints\_check\_constraint"

*Defines system table name for storing check constraints.*
- #define **AK\_CONSTRAINTS\_NOT\_NULL** "AK\_constraints\_not\_null"

*Defines system table name for storing check constraints.*
- #define **AK\_CONSTRAINTS\_UNIQUE** "AK\_constraints\_unique"

*Defines system table name for storing check constraints.*
- #define **AK\_REFERENCE** "AK\_reference"

*Defines system table name for storing check constraints.*
- #define **DROP\_TABLE** 0

*Constant which defines number of drop statement.*
- #define **DROP\_INDEX** 1

- Constant which defines number of drop statement.*
  - #define [DROP\\_VIEW](#) 2
- Constant which defines number of drop statement.*
  - #define [DROP\\_SEQUENCE](#) 3
- Constant which defines number of drop statement.*
  - #define [DROP\\_TRIGGER](#) 4
- Constant which defines number of drop statement.*
  - #define [DROP\\_FUNCTION](#) 5
- Constant which defines number of drop statement.*
  - #define [DROP\\_USER](#) 6
- Constant which defines number of drop statement.*
  - #define [DROP\\_GROUP](#) 7
- Constant which defines number of drop statement.*
  - #define [DROP\\_CONSTRAINT](#) 8
- Constant which defines number of drop statement.*
  - #define [NUM\\_SYS\\_TABLES](#) 20
- Constant which defines length of system\_catalog.*

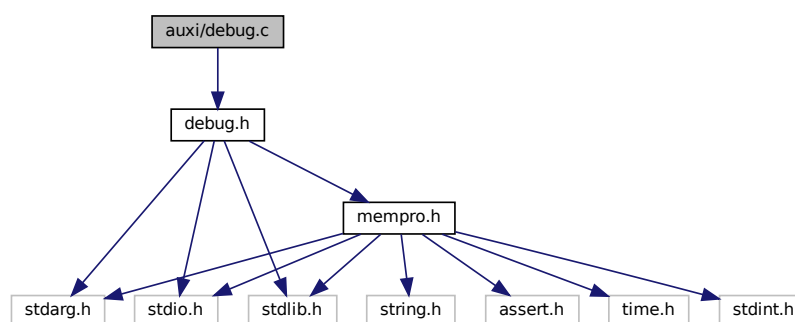
### 5.3.1 Detailed Description

Header file that provides global macros, constants and variables

## 5.4 auxi/debug.c File Reference

```
#include "debug.h"
```

Include dependency graph for debug.c:



### Functions

- int [Ak\\_dbg\\_messg](#) ([DEBUG\\_LEVEL](#) level, [DEBUG\\_TYPE](#) type, const char \*format,...)
 

*Function for printing debug message. Provides debug level, debug type and message with corresponding variables for the output.*

### 5.4.1 Detailed Description

Provides functions for debugging

### 5.4.2 Function Documentation

#### 5.4.2.1 Ak\_dbg\_messg()

```
int Ak_dbg_messg (
    DEBUG_LEVEL level,
    DEBUG_TYPE type,
    const char * format,
    ... )
```

Function for printing debug message. Provides debug level, debug type and message with corresponding variables for the output.

#### Author

Dino Laktašić

#### Parameters

|               |  |
|---------------|--|
| <i>level</i>  | level of debug information for a given DB module           |
| <i>type</i>   | the name of DB module for which to print debug information |
| <i>format</i> | format for the output message                              |
| ...           | variable number of (different) type args used in printf    |

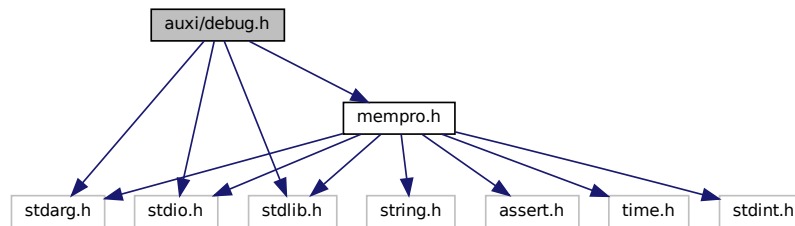
#### Returns

if debug message is printed return 1, else return 0

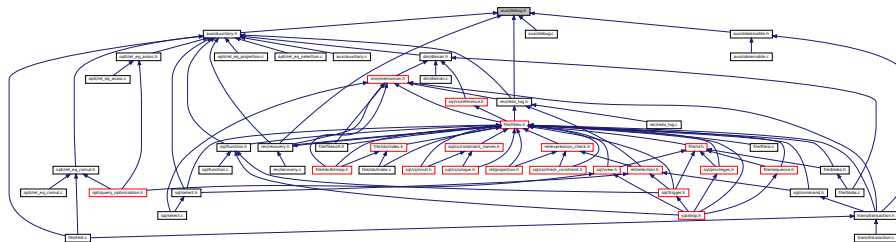
## 5.5 auxi/debug.h File Reference

```
#include "stdarg.h"
#include "stdio.h"
#include "stdlib.h"
#include "mempro.h"
```

Include dependency graph for debug.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define DEBUG_ALL 0`  
Set constant to 1 for complete project debug, else set constant to 0.

## Typedefs

- typedef enum debug\_level **DEBUG\_LEVEL**
- typedef enum debug\_type **DEBUG\_TYPE**

## Enumerations

- enum **debug\_level** { **LOW** = 1, **MIDDLE** = 0, **HIGH** = 0 }
- enum **debug\_type** {  
  **GLOBAL** = 0, **DB\_MAN** = 0, **FILE\_MAN** = 1, **MEMO\_MAN** = 0,  
  **INDICES** = 0, **TABLES** = 0, **REL\_OP** = 0, **REL\_EQ** = 1,  
  **CONSTRAINTS** = 0, **FUNCTIONS** = 0, **SEQUENCES** = 0, **TRIGGERS** = 0,  
  **REDO** = 0 }

## Functions

- int **Ak\_dbg\_messg** (**DEBUG\_LEVEL** level, **DEBUG\_TYPE** type, const char \*format,...)  
Function for printing debug message. Provides debug level, debug type and message with corresponding variables for the output.

### 5.5.1 Detailed Description

Header file that defines global macros, constants and variables for debugging

### 5.5.2 Macro Definition Documentation

#### 5.5.2.1 DEBUG\_ALL

```
#define DEBUG_ALL 0
```

Set constant to 1 for complete project debug, else set constant to 0.

#### Author

Dino Laktašić

### 5.5.3 Function Documentation

#### 5.5.3.1 Ak\_dbg\_messg()

```
int Ak_dbg_messg (  
    DEBUG_LEVEL level,  
    DEBUG_TYPE type,  
    const char * format,  
    ... )
```

Function for printing debug message. Provides debug level, debug type and message with corresponding variables for the output.

#### Author

Dino Laktašić

#### Parameters

|               |  |
|---------------|--|
| <i>level</i>  | level of debug information for a given DB module           |
| <i>type</i>   | the name of DB module for which to print debug information |
| <i>format</i> | format for the output message                              |
| ...           | variable number of (different) type args used in printf    |



## Returns

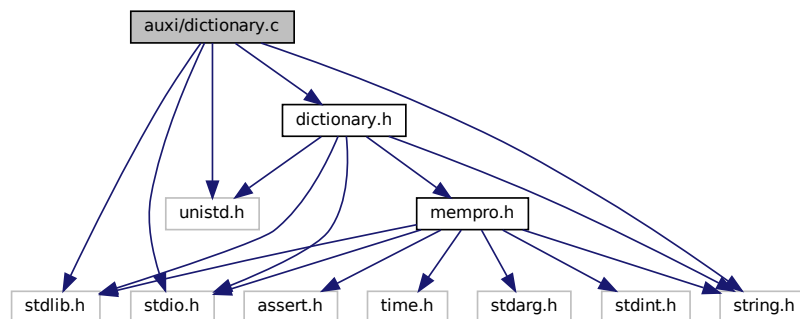
if debug message is printed return 1, else return 0

## 5.6 auxi/dictionary.c File Reference

Implements a dictionary for string variables.

```
#include "dictionary.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Include dependency graph for dictionary.c:



### Macros

- `#define MAXVALSZ 1024`
- `#define DICTMINSZ 128`
- `#define DICT_INVALID_KEY ((char*)-1)`

### Functions

- unsigned `dictionary_hash` (const char \*key)  
*Compute the hash key for a string.*
- `dictionary * dictionary_new` (int size)  
*Create a new dictionary object.*
- void `dictionary_del` (dictionary \*d)  
*Delete a dictionary object.*
- char \* `dictionary_get` (dictionary \*d, const char \*key, char \*def)  
*Get a value from a dictionary.*
- int `dictionary_set` (dictionary \*d, const char \*key, const char \*val)  
*Set a value in a dictionary.*
- void `dictionary_unset` (dictionary \*d, const char \*key)  
*Delete a key in a dictionary.*
- void `dictionary_dump` (dictionary \*d, FILE \*out)  
*Dump a dictionary to an opened file pointer.*

### 5.6.1 Detailed Description

Implements a dictionary for string variables.

#### Author

N. Devillard This module implements a simple dictionary object, i.e. a list of string/string associations. This object is useful to store e.g. informations retrieved from a configuration file (ini files).

### 5.6.2 Macro Definition Documentation

#### 5.6.2.1 DICT\_INVALID\_KEY

```
#define DICT_INVALID_KEY ((char*)-1)
```

Invalid key token

#### 5.6.2.2 DICTMINSZ

```
#define DICTMINSZ 128
```

Minimal allocated number of entries in a dictionary

#### 5.6.2.3 MAXVALSZ

```
#define MAXVALSZ 1024
```

Maximum value size for integers and doubles.

### 5.6.3 Function Documentation

#### 5.6.3.1 dictionary\_del()

```
void dictionary_del (  
    dictionary * d )
```

Delete a dictionary object.

#### Parameters

|          |                                  |
|----------|----------------------------------|
| <i>d</i> | dictionary object to deallocate. |
|----------|----------------------------------|

**Returns**

void

Deallocate a dictionary object and all memory associated to it.

**5.6.3.2 dictionary\_dump()**

```
void dictionary_dump (
    dictionary * d,
    FILE * out )
```

Dump a dictionary to an opened file pointer.

**Parameters**

|          |                      |
|----------|----------------------|
| <i>d</i> | Dictionary to dump   |
| <i>f</i> | Opened file pointer. |

**Returns**

void

Dumps a dictionary onto an opened file pointer. Key pairs are printed out as [Key]=[Value], one per line. It is Ok to provide stdout or stderr as output file pointers.

**5.6.3.3 dictionary\_get()**

```
char* dictionary_get (
    dictionary * d,
    const char * key,
    char * def )
```

Get a value from a dictionary.

**Parameters**

|            |   |
|------------|---|
| <i>d</i>   | dictionary object to search.              |
| <i>key</i> | Key to look for in the dictionary.        |
| <i>def</i> | Default value to return if key not found. |

**Returns**

1 pointer to internally allocated character string.

This function locates a key in a dictionary and returns a pointer to its value, or the passed 'def' pointer if no such key can be found in dictionary. The returned character pointer points to data internal to the dictionary object, you should not try to AK\_free it or modify it.

#### 5.6.3.4 dictionary\_hash()

```
unsigned dictionary_hash (
    const char * key )
```

Compute the hash key for a string.

##### Parameters

|            |                                  |
|------------|----------------------------------|
| <i>key</i> | Character string to use for key. |
|------------|----------------------------------|

##### Returns

1 unsigned int on at least 32 bits.

This hash function has been taken from an Article in Dr Dobbs Journal. This is normally a collision-AK\_free function, distributing keys evenly. The key is stored anyway in the struct so that collision can be avoided by comparing the key itself in last resort.

#### 5.6.3.5 dictionary\_new()

```
dictionary* dictionary_new (
    int size )
```

Create a new dictionary object.

##### Parameters

|             |  |
|-------------|--|
| <i>size</i> | Optional initial size of the dictionary. |
|-------------|--|

##### Returns

1 newly allocated dictionary objet.

This function allocates a new dictionary object of given size and returns it. If you do not know in advance (roughly) the number of entries in the dictionary, give size=0.

#### 5.6.3.6 dictionary\_set()

```
int dictionary_set (
    dictionary * d,
    const char * key,
    const char * val )
```

Set a value in a dictionary.

##### Parameters

|            |                              |
|------------|------------------------------|
| <i>d</i>   | dictionary object to modify. |
| <i>key</i> | Key to modify or add.        |
| <i>val</i> | Value to add.                |

**Returns**

int 0 if Ok, anything else otherwise

If the given key is found in the dictionary, the associated value is replaced by the provided one. If the key cannot be found in the dictionary, it is added to it.

It is Ok to provide a NULL value for val, but NULL values for the dictionary or the key are considered as errors: the function will return immediately in such a case.

Notice that if you dictionary\_set a variable to NULL, a call to dictionary\_get will return a NULL value: the variable will be found, and its value (NULL) is returned. In other words, setting the variable content to NULL is equivalent to deleting the variable from the dictionary. It is not possible (in this implementation) to have a key in the dictionary without value.

This function returns non-zero in case of failure.

**5.6.3.7 dictionary\_unset()**

```
void dictionary_unset (
    dictionary * d,
    const char * key )
```

Delete a key in a dictionary.

**Parameters**

|            |                              |
|------------|------------------------------|
| <i>d</i>   | dictionary object to modify. |
| <i>key</i> | Key to remove.               |

**Returns**

void

This function deletes a key in a dictionary. Nothing is done if the key cannot be found.

**5.7 auxi/dictionary.h File Reference**

Implements a dictionary for string variables.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```



*Get a value from a dictionary.*

- int `dictionary_set` (`dictionary` \*vd, const char \*key, const char \*val)

*Set a value in a dictionary.*

- void `dictionary_unset` (`dictionary` \*d, const char \*key)

*Delete a key in a dictionary.*

- void `dictionary_dump` (`dictionary` \*d, FILE \*out)

*Dump a dictionary to an opened file pointer.*

### 5.7.1 Detailed Description

Implements a dictionary for string variables.

#### Author

N. Devillard This module implements a simple dictionary object, i.e. a list of string/string associations. This object is useful to store e.g. informations retrieved from a configuration file (ini files).

### 5.7.2 Typedef Documentation

#### 5.7.2.1 dictionary

```
typedef struct _dictionary_ dictionary
```

Dictionary object.

This object contains a list of string/string associations. Each association is identified by a unique string key. Looking up values in the dictionary is speeded up by the use of a (hopefully collision-AK\_free) hash function.

### 5.7.3 Function Documentation

#### 5.7.3.1 dictionary\_del()

```
void dictionary_del (
    dictionary * d )
```

Delete a dictionary object.

#### Parameters

|          |                                  |
|----------|----------------------------------|
| <i>d</i> | dictionary object to deallocate. |
|----------|----------------------------------|

**Returns**

void

Deallocate a dictionary object and all memory associated to it.

**5.7.3.2 dictionary\_dump()**

```
void dictionary_dump (
    dictionary * d,
    FILE * out )
```

Dump a dictionary to an opened file pointer.

**Parameters**

|          |                      |
|----------|----------------------|
| <i>d</i> | Dictionary to dump   |
| <i>f</i> | Opened file pointer. |

**Returns**

void

Dumps a dictionary onto an opened file pointer. Key pairs are printed out as [Key]=[Value], one per line. It is Ok to provide stdout or stderr as output file pointers.

**5.7.3.3 dictionary\_get()**

```
char* dictionary_get (
    dictionary * d,
    const char * key,
    char * def )
```

Get a value from a dictionary.

**Parameters**

|            |   |
|------------|---|
| <i>d</i>   | dictionary object to search.              |
| <i>key</i> | Key to look for in the dictionary.        |
| <i>def</i> | Default value to return if key not found. |

**Returns**

1 pointer to internally allocated character string.

This function locates a key in a dictionary and returns a pointer to its value, or the passed 'def' pointer if no such key can be found in dictionary. The returned character pointer points to data internal to the dictionary object, you should not try to AK\_free it or modify it.



#### 5.7.3.4 dictionary\_hash()

```
unsigned dictionary_hash (
    const char * key )
```

Compute the hash key for a string.

##### Parameters

|            |                                  |
|------------|----------------------------------|
| <i>key</i> | Character string to use for key. |
|------------|----------------------------------|

##### Returns

1 unsigned int on at least 32 bits.

This hash function has been taken from an Article in Dr Dobbs Journal. This is normally a collision-AK\_free function, distributing keys evenly. The key is stored anyway in the struct so that collision can be avoided by comparing the key itself in last resort.

#### 5.7.3.5 dictionary\_new()

```
dictionary* dictionary_new (
    int size )
```

Create a new dictionary object.

##### Parameters

|             |  |
|-------------|--|
| <i>size</i> | Optional initial size of the dictionary. |
|-------------|--|

##### Returns

1 newly allocated dictionary objet.

This function allocates a new dictionary object of given size and returns it. If you do not know in advance (roughly) the number of entries in the dictionary, give size=0.

#### 5.7.3.6 dictionary\_set()

```
int dictionary_set (
    dictionary * d,
    const char * key,
    const char * val )
```

Set a value in a dictionary.

##### Parameters

|            |                              |
|------------|------------------------------|
| <i>d</i>   | dictionary object to modify. |
| <i>key</i> | Key to modify or add.        |
| <i>val</i> | Value to add.                |

**Returns**

int 0 if Ok, anything else otherwise

If the given key is found in the dictionary, the associated value is replaced by the provided one. If the key cannot be found in the dictionary, it is added to it.

It is Ok to provide a NULL value for val, but NULL values for the dictionary or the key are considered as errors: the function will return immediately in such a case.

Notice that if you dictionary\_set a variable to NULL, a call to dictionary\_get will return a NULL value: the variable will be found, and its value (NULL) is returned. In other words, setting the variable content to NULL is equivalent to deleting the variable from the dictionary. It is not possible (in this implementation) to have a key in the dictionary without value.

This function returns non-zero in case of failure.

**5.7.3.7 dictionary\_unset()**

```
void dictionary_unset (
    dictionary * d,
    const char * key )
```

Delete a key in a dictionary.

**Parameters**

|            |                              |
|------------|------------------------------|
| <i>d</i>   | dictionary object to modify. |
| <i>key</i> | Key to remove.               |

**Returns**

void

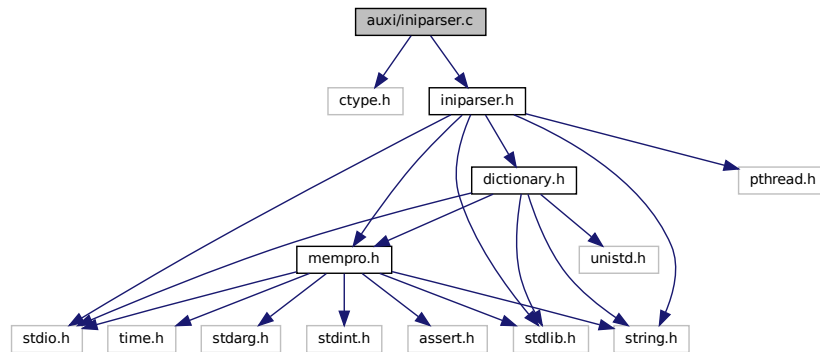
This function deletes a key in a dictionary. Nothing is done if the key cannot be found.

**5.8 auxi/iniparser.c File Reference**

Parser for ini files.

```
#include <ctype.h>
#include "iniparser.h"
```

Include dependency graph for iniparser.c:



## Macros

- `#define ASCIIINESZ (1024)`
- `#define INI_INVALID_KEY ((char*)-1)`

## Typedefs

- `typedef enum _line_status line_status`

## Enumerations

- `enum _line_status {  
    LINE_UNPROCESSED, LINE_ERROR, LINE_EMPTY, LINE_COMMENT,  
    LINE_SECTION, LINE_VALUE }`

## Functions

- `int iniparser_getnsec (dictionary *d)`  
*Get number of sections in a dictionary.*
- `char * iniparser_getsecname (dictionary *d, int n)`  
*Get name for section n in a dictionary.*
- `void iniparser_dump (dictionary *d, FILE *f)`  
*Dump a dictionary to an opened file pointer.*
- `void iniparser_dump_ini (dictionary *d, FILE *f)`  
*Save a dictionary to a loadable ini file.*
- `void iniparser_dumpsection_ini (dictionary *d, char *s, FILE *f)`  
*Save a dictionary section to a loadable ini file.*
- `int iniparser_getsecnkeys (dictionary *d, char *s)`  
*Get the number of keys in a section of a dictionary.*
- `char ** iniparser_getseckeys (dictionary *d, char *s)`  
*Get the number of keys in a section of a dictionary.*
- `char * iniparser_getstring (dictionary *d, const char *key, char *def)`

- Get the string associated to a key.*
- int `iniparser_getint` (`dictionary` \*d, const char \*key, int notfound)
- Get the string associated to a key, convert to an int.*
- double `iniparser_getdouble` (`dictionary` \*d, const char \*key, double notfound)
- Get the string associated to a key, convert to a double.*
- int `iniparser_getboolean` (`dictionary` \*d, const char \*key, int notfound)
- Get the string associated to a key, convert to a boolean.*
- int `iniparser_find_entry` (`dictionary` \*ini, const char \*entry)
- Finds out if a given entry exists in a dictionary.*
- int `iniparser_set` (`dictionary` \*ini, const char \*entry, const char \*val)
- Set an entry in a dictionary.*
- void `iniparser_unset` (`dictionary` \*ini, const char \*entry)
- Delete an entry in a dictionary.*
- `dictionary` \* `iniparser_load` (const char \*ininame)
- Parse an ini file and return an allocated dictionary object.*
- void `iniparser_AK_freedict` (`dictionary` \*d)
- Free all memory associated to an ini dictionary.*
- void `AK_inflate_config` ()

## Variables

- pthread\_mutex\_t `iniParserMutex` = PTHREAD\_MUTEX\_INITIALIZER
- `dictionary` \* `AK_config`

## 5.8.1 Detailed Description

Parser for ini files.

Author

N. Devillard

## 5.8.2 Typedef Documentation

### 5.8.2.1 line\_status

```
typedef enum _line_status_ line_status
```

This enum stores the status for each parsed line (internal use only).

## 5.8.3 Enumeration Type Documentation

### 5.8.3.1 `_line_status_`

enum `_line_status_`

This enum stores the status for each parsed line (internal use only).

## 5.8.4 Function Documentation

### 5.8.4.1 `iniparser_AK_freedict()`

```
void iniparser_AK_freedict (
    dictionary * d )
```

Free all memory associated to an ini dictionary.

#### Parameters

|          |                       |
|----------|-----------------------|
| <i>d</i> | Dictionary to AK_free |
|----------|-----------------------|

#### Returns

void

Free all memory associated to an ini dictionary. It is mandatory to call this function before the dictionary object gets out of the current context.

### 5.8.4.2 `iniparser_dump()`

```
void iniparser_dump (
    dictionary * d,
    FILE * f )
```

Dump a dictionary to an opened file pointer.

#### Parameters

|          |                                 |
|----------|---------------------------------|
| <i>d</i> | Dictionary to dump.             |
| <i>f</i> | Opened file pointer to dump to. |

#### Returns

void

This function prints out the contents of a dictionary, one element by line, onto the provided file pointer. It is OK to specify `stderr` or `stdout` as output files. This function is meant for debugging purposes mostly.

#### 5.8.4.3 iniparser\_dump\_ini()

```
void iniparser_dump_ini (
    dictionary * d,
    FILE * f )
```

Save a dictionary to a loadable ini file.

##### Parameters

|          |                                |
|----------|--------------------------------|
| <i>d</i> | Dictionary to dump             |
| <i>f</i> | Opened file pointer to dump to |

##### Returns

void

This function dumps a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

#### 5.8.4.4 iniparser\_dumpsection\_ini()

```
void iniparser_dumpsection_ini (
    dictionary * d,
    char * s,
    FILE * f )
```

Save a dictionary section to a loadable ini file.

##### Parameters

|          |                                    |
|----------|------------------------------------|
| <i>d</i> | Dictionary to dump                 |
| <i>s</i> | Section name of dictionary to dump |
| <i>f</i> | Opened file pointer to dump to     |

##### Returns

void

This function dumps a given section of a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

#### 5.8.4.5 iniparser\_find\_entry()

```
int iniparser_find_entry (
    dictionary * ini,
    const char * entry )
```

Finds out if a given entry exists in a dictionary.

## Parameters

|              |                               |
|--------------|-------------------------------|
| <i>ini</i>   | Dictionary to search          |
| <i>entry</i> | Name of the entry to look for |

## Returns

integer 1 if entry exists, 0 otherwise

Finds out if a given entry exists in the dictionary. Since sections are stored as keys with NULL associated values, this is the only way of querying for the presence of sections in a dictionary.

## 5.8.4.6 iniparser\_getboolean()

```
int iniparser_getboolean (
    dictionary * d,
    const char * key,
    int notfound )
```

Get the string associated to a key, convert to a boolean.

## Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>d</i>        | Dictionary to search             |
| <i>key</i>      | Key string to look for           |
| <i>notfound</i> | Value to return in case of error |

## Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

A true boolean is found if one of the following is matched:

- A string starting with 'y'
- A string starting with 'Y'
- A string starting with 't'
- A string starting with 'T'
- A string starting with '1'

A false boolean is found if one of the following is matched:

- A string starting with 'n'
- A string starting with 'N'
- A string starting with 'f'
- A string starting with 'F'
- A string starting with '0'

The notfound value returned if no boolean is identified, does not necessarily have to be 0 or 1.

#### 5.8.4.7 iniparser\_getdouble()

```
double iniparser_getdouble (
    dictionary * d,
    const char * key,
    double notfound )
```

Get the string associated to a key, convert to a double.

##### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>d</i>        | Dictionary to search             |
| <i>key</i>      | Key string to look for           |
| <i>notfound</i> | Value to return in case of error |

##### Returns

double

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

#### 5.8.4.8 iniparser\_getint()

```
int iniparser_getint (
    dictionary * d,
    const char * key,
    int notfound )
```

Get the string associated to a key, convert to an int.

##### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>d</i>        | Dictionary to search             |
| <i>key</i>      | Key string to look for           |
| <i>notfound</i> | Value to return in case of error |

##### Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

"42" -> 42 "042" -> 34 (octal -> decimal) "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()



#### 5.8.4.9 iniparser\_getnsec()

```
int iniparser_getnsec (
    dictionary * d )
```

Get number of sections in a dictionary.

##### Parameters

|          |                       |
|----------|-----------------------|
| <i>d</i> | Dictionary to examine |
|----------|-----------------------|

##### Returns

int Number of sections found in dictionary

This function returns the number of sections found in a dictionary. The test to recognize sections is done on the string stored in the dictionary: a section name is given as "section" whereas a key is stored as "section:key", thus the test looks for entries that do not contain a colon.

This clearly fails in the case a section name contains a colon, but this should simply be avoided.

This function returns -1 in case of error.

#### 5.8.4.10 iniparser\_getseckeys()

```
char** iniparser_getseckeys (
    dictionary * d,
    char * s )
```

Get the number of keys in a section of a dictionary.

##### Parameters

|          |                                       |
|----------|---------------------------------------|
| <i>d</i> | Dictionary to examine                 |
| <i>s</i> | Section name of dictionary to examine |

##### Returns

pointer to statically allocated character strings

This function queries a dictionary and finds all keys in a given section. Each pointer in the returned char pointer-to-pointer is pointing to a string allocated in the dictionary; do not `AK_free` or modify them.

This function returns NULL in case of error.

#### 5.8.4.11 iniparser\_getsecname()

```
char* iniparser_getsecname (
    dictionary * d,
    int n )
```

Get name for section n in a dictionary.

**Parameters**

|          |                                    |
|----------|------------------------------------|
| <i>d</i> | Dictionary to examine              |
| <i>n</i> | Section number (from 0 to nsec-1). |

**Returns**

Pointer to char string

This function locates the n-th section in a dictionary and returns its name as a pointer to a string statically allocated inside the dictionary. Do not `AK_free` or modify the returned string!

This function returns `NULL` in case of error.

**5.8.4.12 `iniparser_getsecnkeys()`**

```
int iniparser_getsecnkeys (  
    dictionary * d,  
    char * s )
```

Get the number of keys in a section of a dictionary.

**Parameters**

|          |                                       |
|----------|---------------------------------------|
| <i>d</i> | Dictionary to examine                 |
| <i>s</i> | Section name of dictionary to examine |

**Returns**

Number of keys in section

**5.8.4.13 `iniparser_getstring()`**

```
char* iniparser_getstring (  
    dictionary * d,  
    const char * key,  
    char * def )
```

Get the string associated to a key.

**Parameters**

|            |   |
|------------|---|
| <i>d</i>   | Dictionary to search                      |
| <i>key</i> | Key string to look for                    |
| <i>def</i> | Default value to return if key not found. |

**Returns**

pointer to statically allocated character string

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the pointer passed as 'def' is returned. The returned char pointer is pointing to a string allocated in the dictionary, do not `AK_free` or modify it.

**5.8.4.14 iniparser\_load()**

```
dictionary* iniparser_load (  
    const char * ininame )
```

Parse an ini file and return an allocated dictionary object.

**Parameters**

|                |                               |
|----------------|-------------------------------|
| <i>ininame</i> | Name of the ini file to read. |
|----------------|-------------------------------|

**Returns**

Pointer to newly allocated dictionary

This is the parser for ini files. This function is called, providing the name of the file to be read. It returns a dictionary object that should not be accessed directly, but through accessor functions instead.

The returned dictionary must be `AK_freed` using [iniparser\\_AK\\_freedict\(\)](#).

**5.8.4.15 iniparser\_set()**

```
int iniparser_set (  
    dictionary * ini,  
    const char * entry,  
    const char * val )
```

Set an entry in a dictionary.

**Parameters**

|              |                                      |
|--------------|--------------------------------------|
| <i>ini</i>   | Dictionary to modify.                |
| <i>entry</i> | Entry to modify (entry name)         |
| <i>val</i>   | New value to associate to the entry. |

**Returns**

int 0 if Ok, -1 otherwise.

If the given entry can be found in the dictionary, it is modified to contain the provided value. If it cannot be found, -1 is returned. It is Ok to set val to NULL.

#### 5.8.4.16 iniparser\_unset()

```
void iniparser_unset (
    dictionary * ini,
    const char * entry )
```

Delete an entry in a dictionary.

##### Parameters

|              |                              |
|--------------|------------------------------|
| <i>ini</i>   | Dictionary to modify         |
| <i>entry</i> | Entry to delete (entry name) |

##### Returns

void

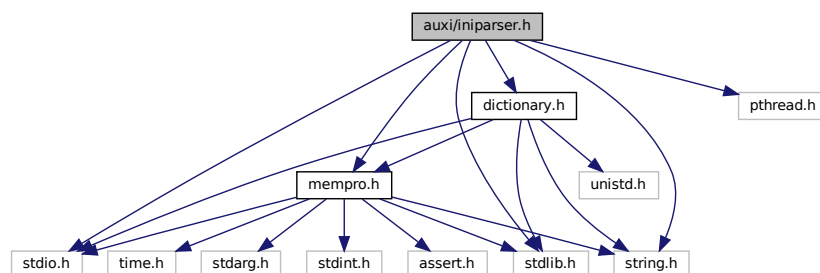
If the given entry can be found, it is deleted from the dictionary.

## 5.9 auxi/iniparser.h File Reference

Parser for ini files.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include "dictionary.h"
#include "mempro.h"
```

Include dependency graph for iniparser.h:





## Variables

- `dictionary` \* `AK_config`

### 5.9.1 Detailed Description

Parser for ini files.

#### Author

N. Devillard

### 5.9.2 Function Documentation

#### 5.9.2.1 `iniparser_AK_freedict()`

```
void iniparser_AK_freedict (
    dictionary * d )
```

Free all memory associated to an ini dictionary.

#### Parameters

|          |                       |
|----------|-----------------------|
| <i>d</i> | Dictionary to AK_free |
|----------|-----------------------|

#### Returns

void

Free all memory associated to an ini dictionary. It is mandatory to call this function before the dictionary object gets out of the current context.

#### 5.9.2.2 `iniparser_dump()`

```
void iniparser_dump (
    dictionary * d,
    FILE * f )
```

Dump a dictionary to an opened file pointer.

#### Parameters

|          |                                 |
|----------|---------------------------------|
| <i>d</i> | Dictionary to dump.             |
| <i>f</i> | Opened file pointer to dump to. |

**Returns**

void

This function prints out the contents of a dictionary, one element by line, onto the provided file pointer. It is OK to specify `stderr` or `stdout` as output files. This function is meant for debugging purposes mostly.

**5.9.2.3 iniparser\_dump\_ini()**

```
void iniparser_dump_ini (
    dictionary * d,
    FILE * f )
```

Save a dictionary to a loadable ini file.

**Parameters**

|          |                                |
|----------|--------------------------------|
| <i>d</i> | Dictionary to dump             |
| <i>f</i> | Opened file pointer to dump to |

**Returns**

void

This function dumps a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

**5.9.2.4 iniparser\_dumpsection\_ini()**

```
void iniparser_dumpsection_ini (
    dictionary * d,
    char * s,
    FILE * f )
```

Save a dictionary section to a loadable ini file.

**Parameters**

|          |                                    |
|----------|------------------------------------|
| <i>d</i> | Dictionary to dump                 |
| <i>s</i> | Section name of dictionary to dump |
| <i>f</i> | Opened file pointer to dump to     |

**Returns**

void

This function dumps a given section of a given dictionary into a loadable ini file. It is Ok to specify `stderr` or `stdout` as output files.

### 5.9.2.5 iniparser\_find\_entry()

```
int iniparser_find_entry (
    dictionary * ini,
    const char * entry )
```

Finds out if a given entry exists in a dictionary.

#### Parameters

|              |                               |
|--------------|-------------------------------|
| <i>ini</i>   | Dictionary to search          |
| <i>entry</i> | Name of the entry to look for |

#### Returns

integer 1 if entry exists, 0 otherwise

Finds out if a given entry exists in the dictionary. Since sections are stored as keys with NULL associated values, this is the only way of querying for the presence of sections in a dictionary.

### 5.9.2.6 iniparser\_getboolean()

```
int iniparser_getboolean (
    dictionary * d,
    const char * key,
    int notfound )
```

Get the string associated to a key, convert to a boolean.

#### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>d</i>        | Dictionary to search             |
| <i>key</i>      | Key string to look for           |
| <i>notfound</i> | Value to return in case of error |

#### Returns

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

A true boolean is found if one of the following is matched:

- A string starting with 'y'
- A string starting with 'Y'
- A string starting with 't'
- A string starting with 'T'



- A string starting with '1'

A false boolean is found if one of the following is matched:

- A string starting with 'n'
- A string starting with 'N'
- A string starting with 'f'
- A string starting with 'F'
- A string starting with '0'

The notfound value returned if no boolean is identified, does not necessarily have to be 0 or 1.

#### 5.9.2.7 iniparser\_getdouble()

```
double iniparser_getdouble (
    dictionary * d,
    const char * key,
    double notfound )
```

Get the string associated to a key, convert to a double.

##### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>d</i>        | Dictionary to search             |
| <i>key</i>      | Key string to look for           |
| <i>notfound</i> | Value to return in case of error |

##### Returns

double

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

#### 5.9.2.8 iniparser\_getint()

```
int iniparser_getint (
    dictionary * d,
    const char * key,
    int notfound )
```

Get the string associated to a key, convert to an int.

##### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>d</i>        | Dictionary to search             |
| <i>key</i>      | Key string to look for           |
| <i>notfound</i> | Value to return in case of error |

**Returns**

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

- "42" -> 42
- "042" -> 34 (octal -> decimal)
- "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

**Parameters**

|                 |                                  |
|-----------------|----------------------------------|
| <i>d</i>        | Dictionary to search             |
| <i>key</i>      | Key string to look for           |
| <i>notfound</i> | Value to return in case of error |

**Returns**

integer

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the notfound value is returned.

Supported values for integers include the usual C notation so decimal, octal (starting with 0) and hexadecimal (starting with 0x) are supported. Examples:

"42" -> 42 "042" -> 34 (octal -> decimal) "0x42" -> 66 (hexa -> decimal)

Warning: the conversion may overflow in various ways. Conversion is totally outsourced to strtol(), see the associated man page for overflow handling.

Credits: Thanks to A. Becker for suggesting strtol()

**5.9.2.9 iniparser\_getnsec()**

```
int iniparser_getnsec (
    dictionary * d )
```

Get number of sections in a dictionary.

#### Parameters

|          |                       |
|----------|-----------------------|
| <i>d</i> | Dictionary to examine |
|----------|-----------------------|

#### Returns

int Number of sections found in dictionary

This function returns the number of sections found in a dictionary. The test to recognize sections is done on the string stored in the dictionary: a section name is given as "section" whereas a key is stored as "section:key", thus the test looks for entries that do not contain a colon.

This clearly fails in the case a section name contains a colon, but this should simply be avoided.

This function returns -1 in case of error.

#### 5.9.2.10 iniparser\_getseckey()

```
char** iniparser_getseckey (
    dictionary * d,
    char * s )
```

Get the number of keys in a section of a dictionary.

#### Parameters

|          |                                       |
|----------|---------------------------------------|
| <i>d</i> | Dictionary to examine                 |
| <i>s</i> | Section name of dictionary to examine |

#### Returns

pointer to statically allocated character strings

This function queries a dictionary and finds all keys in a given section. Each pointer in the returned char pointer-to-pointer is pointing to a string allocated in the dictionary; do not AK\_free or modify them.

This function returns NULL in case of error.

#### 5.9.2.11 iniparser\_getsecname()

```
char* iniparser_getsecname (
    dictionary * d,
    int n )
```

Get name for section n in a dictionary.

#### Parameters

|          |                                    |
|----------|------------------------------------|
| <i>d</i> | Dictionary to examine              |
| <i>n</i> | Section number (from 0 to nsec-1). |

**Returns**

Pointer to char string

This function locates the n-th section in a dictionary and returns its name as a pointer to a string statically allocated inside the dictionary. Do not `AK_free` or modify the returned string!

This function returns `NULL` in case of error.

**5.9.2.12 `iniparser_getsecnkeys()`**

```
int iniparser_getsecnkeys (
    dictionary * d,
    char * s )
```

Get the number of keys in a section of a dictionary.

**Parameters**

|          |                                       |
|----------|---------------------------------------|
| <i>d</i> | Dictionary to examine                 |
| <i>s</i> | Section name of dictionary to examine |

**Returns**

Number of keys in section

**5.9.2.13 `iniparser_getstring()`**

```
char* iniparser_getstring (
    dictionary * d,
    const char * key,
    char * def )
```

Get the string associated to a key.

**Parameters**

|            |   |
|------------|---|
| <i>d</i>   | Dictionary to search                      |
| <i>key</i> | Key string to look for                    |
| <i>def</i> | Default value to return if key not found. |

**Returns**

pointer to statically allocated character string

This function queries a dictionary for a key. A key as read from an ini file is given as "section:key". If the key cannot be found, the pointer passed as 'def' is returned. The returned char pointer is pointing to a string allocated in the dictionary, do not `AK_free` or modify it.

#### 5.9.2.14 iniparser\_load()

```
dictionary* iniparser_load (
    const char * ininame )
```

Parse an ini file and return an allocated dictionary object.

##### Parameters

|                |                               |
|----------------|-------------------------------|
| <i>ininame</i> | Name of the ini file to read. |
|----------------|-------------------------------|

##### Returns

Pointer to newly allocated dictionary

This is the parser for ini files. This function is called, providing the name of the file to be read. It returns a dictionary object that should not be accessed directly, but through accessor functions instead.

The returned dictionary must be AK\_freed using [iniparser\\_AK\\_freedict\(\)](#).

#### 5.9.2.15 iniparser\_set()

```
int iniparser_set (
    dictionary * ini,
    const char * entry,
    const char * val )
```

Set an entry in a dictionary.

##### Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>ini</i>   | Dictionary to modify.                |
| <i>entry</i> | Entry to modify (entry name)         |
| <i>val</i>   | New value to associate to the entry. |

##### Returns

int 0 if Ok, -1 otherwise.

If the given entry can be found in the dictionary, it is modified to contain the provided value. If it cannot be found, -1 is returned. It is Ok to set val to NULL.

#### 5.9.2.16 iniparser\_unset()

```
void iniparser_unset (
    dictionary * ini,
    const char * entry )
```

Delete an entry in a dictionary.

**Parameters**

|              |                              |
|--------------|------------------------------|
| <i>ini</i>   | Dictionary to modify         |
| <i>entry</i> | Entry to delete (entry name) |

**Returns**

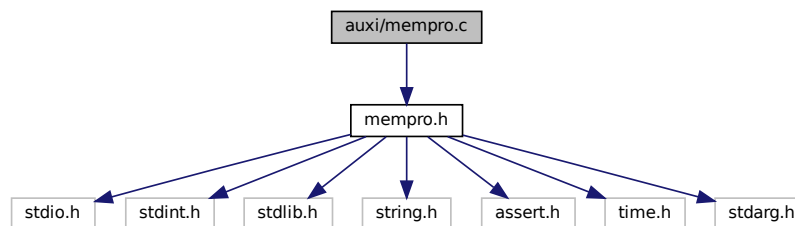
void

If the given entry can be found, it is deleted from the dictionary.

**5.10 auxi/mempro.c File Reference**

```
#include "mempro.h"
```

Include dependency graph for mempro.c:

**Functions**

- void [AK\\_debmod\\_d](#) ([AK\\_debmod\\_state](#) \*ds, const char \*message)  
*Function prints debug message [private function].*
- void [AK\\_debmod\\_dv](#) ([AK\\_debmod\\_state](#) \*ds, const char \*format,...)  
*Function prints debug message [private function].*
- void [AK\\_debmod\\_enter\\_critical\\_sec](#) ([AK\\_debmod\\_state](#) \*ds)  
*Reserves ds for use [private function].*
- void [AK\\_debmod\\_leave\\_critical\\_sec](#) ([AK\\_debmod\\_state](#) \*ds)  
*Makes ds available [private function].*
- [AK\\_debmod\\_state](#) \* [AK\\_debmod\\_init](#) (void)  
*Initializes debug mode structure [public function].*
- void [AK\\_debmod\\_die](#) ([AK\\_debmod\\_state](#) \*ds)  
*Destroy debug mode state (call before main() exit) [public function].*
- void \* [AK\\_debmod\\_calloc](#) ([AK\\_debmod\\_state](#) \*ds, uint32\_t size)  
*Allocates memory [private function].*
- void [AK\\_debmod\\_free](#) ([AK\\_debmod\\_state](#) \*ds, void \*memory)  
*Frees memory allocated with debmod\_alloc [private function].*
- void \* [AK\\_calloc](#) (size\_t num, size\_t size)  
*Allocates memory (see calloc) [public function].*

- void \* [AK\\_malloc](#) (size\_t size)  
*Allocate memory (see malloc) [public function].*
- void [AK\\_free](#) (void \*ptr)  
*Free memory at ptr (see free) [public function].*
- void \* [AK\\_realloc](#) (void \*ptr, size\_t size)  
*Reallocates memory (see realloc) [public function].*
- void [AK\\_write\\_protect](#) (void \*memory)  
*Function write-protects memory [public function].*
- void [AK\\_write\\_unprotect](#) (void \*memory)  
*Function write-unprotects memory [public function].*
- void [AK\\_check\\_for\\_writes](#) (void)  
*Marks pages dirty if there were writes between calls to this function.*
- int32\_t [AK\\_debmod\\_func\\_id](#) ([AK\\_debmod\\_state](#) \*ds, const char \*func\_name)  
*Returns function id for given func\_name.*
- const char \* [AK\\_debmod\\_func\\_get\\_name](#) ([AK\\_debmod\\_state](#) \*ds, int32\_t function\_id)  
*Lookup function name [private function].*
- int32\_t [AK\\_debmod\\_func\\_add](#) ([AK\\_debmod\\_state](#) \*ds, const char \*func\_name)  
*Adds function name to list [private function].*
- void [AK\\_debmod\\_fstack\\_push](#) ([AK\\_debmod\\_state](#) \*ds, int32\_t func\_id)  
*Push function id on stack [private function].*
- int32\_t [AK\\_debmod\\_fstack\\_pop](#) ([AK\\_debmod\\_state](#) \*ds)  
*Pops function id from stack [private function].*
- void [AK\\_debmod\\_function\\_current](#) ([AK\\_debmod\\_state](#) \*ds, int32\_t new\_function\_id)  
*Sets current function [private function].*
- void [AK\\_debmod\\_function\\_prologue](#) (const char \*func\_name, const char \*source\_file, int source\_line)  
*Not for direct use (only with macro AK\_PRO). Marks function prologue.*
- void [AK\\_debmod\\_log\\_memory\\_alloc](#) (int32\_t func\_id)  
*print debmod information on function [private function]*
- void [AK\\_debmod\\_function\\_epilogue](#) (const char \*func\_name, const char \*source\_file, int source\_line)  
*Not for direct use (only with macro AK\_EPI). Marks function epilogue.*
- void [AK\\_debmod\\_print\\_function\\_use](#) (const char \*func\_name, uint8\_t in\_recur)  
*Print function dependency [private function].*
- void [AK\\_print\\_function\\_use](#) (const char \*func\_name)  
*Print function dependency [public function].*
- void [AK\\_print\\_function\\_uses](#) ()  
*Print function dependency for all functions [public function].*
- void [AK\\_print\\_active\\_functions](#) ()  
*Print all detected functions.*
- size\_t [AK\\_fwrite](#) (const void \*buf, size\_t size, size\_t count, FILE \*fp)  
*Write to a file from a buffer (see fwrite) [public function].*
- size\_t [AK\\_fread](#) (void \*buf, size\_t size, size\_t count, FILE \*fp)  
*Read from a file (see fread) [public function].*
- void [AK\\_mempro\\_test](#) ()  
*Test function.*

### 5.10.1 Detailed Description

Implementation of the memory wrappers and debug mode of Kalashnikov DB.

## 5.10.2 Function Documentation

### 5.10.2.1 AK\_malloc()

```
void* AK_malloc (
    size_t num,
    size_t size )
```

Allocates memory (see calloc) [public function].

#### Author

Marin Rukavina, Mislav Bozicevic

#### Parameters

|             |                     |
|-------------|---------------------|
| <i>num</i>  | number of elements  |
| <i>size</i> | of element in bytes |

#### Returns

allocated memory or NULL

### 5.10.2.2 AK\_check\_for\_writes()

```
void AK_check_for_writes (
    void )
```

Marks pages dirty if there were writes between calls to this function.

#### Author

Marin Rukavina, Mislav Bozicevic

#### Returns

void

### 5.10.2.3 AK\_debmod\_malloc()

```
void* AK_debmod_malloc (
    AK_debmod_state * ds,
    uint32_t size )
```

Allocates memory [private function].

#### Author

Marin Rukavina, Mislav Bozicevic



**Parameters**

|             |                      |
|-------------|----------------------|
| <i>ds</i>   | debug mode state     |
| <i>size</i> | in bytes to allocate |

**Returns**

pointer to allocated memory or NULL

**5.10.2.4 AK\_debmod\_d()**

```
void AK_debmod_d (
    AK_debmod_state * ds,
    const char * message )
```

Function prints debug message [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                |                  |
|----------------|------------------|
| <i>ds</i>      | debug mode state |
| <i>message</i> | string to print  |

**Returns**

void

**5.10.2.5 AK\_debmod\_die()**

```
void AK_debmod_die (
    AK_debmod_state * ds )
```

Destroy debug mode state (call before main() exit) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|           |                  |
|-----------|------------------|
| <i>ds</i> | debug mode state |
|-----------|------------------|

**Returns**

void

**5.10.2.6 AK\_debmod\_dv()**

```
void AK_debmod_dv (
    AK_debmod_state * ds,
    const char * format,
    ... )
```

Function prints debug message [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|               |                           |
|---------------|---------------------------|
| <i>ds</i>     | debug mode state          |
| <i>format</i> | format string like printf |

**Returns**

void

**5.10.2.7 AK\_debmod\_enter\_critical\_sec()**

```
void AK_debmod_enter_critical_sec (
    AK_debmod_state * ds )
```

Reserves ds for use [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|           |                  |
|-----------|------------------|
| <i>ds</i> | debug mode state |
|-----------|------------------|

**Returns**

void

### 5.10.2.8 AK\_debmod\_free()

```
void AK_debmod_free (
    AK_debmod_state * ds,
    void * memory )
```

Frees memory allocated with debmod\_alloc [private function].

#### Author

Marin Rukavina, Mislav Bozicevic

#### Parameters

|               |                  |
|---------------|------------------|
| <i>ds</i>     | debug mode state |
| <i>memory</i> |                  |

#### Returns

void

### 5.10.2.9 AK\_debmod\_fstack\_pop()

```
int32_t AK_debmod_fstack_pop (
    AK_debmod_state * ds )
```

Pops function id from stack [private function].

#### Author

Marin Rukavina, Mislav Bozicevic

#### Parameters

|           |                  |
|-----------|------------------|
| <i>ds</i> | debug mode state |
|-----------|------------------|

#### Returns

function id popped

### 5.10.2.10 AK\_debmod\_fstack\_push()

```
void AK_debmod_fstack_push (
    AK_debmod_state * ds,
    int32_t func_id )
```

Push function id on stack [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                             |                  |
|-----------------------------|------------------|
| <i>ds</i>                   | debug mode state |
| <i>func</i> ↔<br><i>_id</i> | function id      |

**Returns**

void

**5.10.2.11 AK\_debmod\_func\_add()**

```
int32_t AK_debmod_func_add (
    AK_debmod_state * ds,
    const char * func_name )
```

Adds function name to list [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                  |                  |
|------------------|------------------|
| <i>ds</i>        | debug mode state |
| <i>func_name</i> |                  |

**Returns**

id for added function name

**5.10.2.12 AK\_debmod\_func\_get\_name()**

```
const char* AK_debmod_func_get_name (
    AK_debmod_state * ds,
    int32_t function_id )
```

Lookup function name [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                    |                  |
|--------------------|------------------|
| <i>ds</i>          | debug mode state |
| <i>function_id</i> |                  |

**Returns**

function name for given function\_id

**5.10.2.13 AK\_debmod\_func\_id()**

```
int32_t AK_debmod_func_id (
    AK_debmod_state * ds,
    const char * func_name )
```

Returns function id for given func\_name.

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                  |                                  |
|------------------|----------------------------------|
| <i>ds</i>        | debug mode state                 |
| <i>func_name</i> | function name [private function] |

**Returns**

function id

**5.10.2.14 AK\_debmod\_function\_current()**

```
void AK_debmod_function_current (
    AK_debmod_state * ds,
    int32_t new_function_id )
```

Sets current function [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                                     |                  |
|-------------------------------------|------------------|
| <i>ds</i>                           | debug mode state |
| <i>new_function</i> ↔<br><i>_id</i> |                  |

**Returns**

void

**5.10.2.15 AK\_debmod\_function\_epilogue()**

```
void AK_debmod_function_epilogue (
    const char * func_name,
    const char * source_file,
    int source_line )
```

Not for direct use (only with macro AK\_EPI). Marks function epilogue.

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                    |   |
|--------------------|---|
| <i>func_name</i>   | function name as in source              |
| <i>source_file</i> | file name where function is defined     |
| <i>source_line</i> | line from which this function is called |

**Returns**

void

**5.10.2.16 AK\_debmod\_function\_prologue()**

```
void AK_debmod_function_prologue (
    const char * func_name,
    const char * source_file,
    int source_line )
```

Not for direct use (only with macro AK\_PRO). Marks function prologue.

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                    |   |
|--------------------|---|
| <i>func_name</i>   | function name as in source              |
| <i>source_file</i> | file name where function is defined     |
| <i>source_line</i> | line from which this function is called |

**Returns**

void

**5.10.2.17 AK\_debmod\_init()**

```
AK_debmod_state* AK_debmod_init (
    void )
```

Initializes debug mode structure [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Returns**

initialized debug mode state

**5.10.2.18 AK\_debmod\_leave\_critical\_sec()**

```
void AK_debmod_leave_critical_sec (
    AK_debmod_state * ds )
```

Makes ds available [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|           |                  |
|-----------|------------------|
| <i>ds</i> | debug mode state |
|-----------|------------------|

**Returns**

void

#### 5.10.2.19 AK\_debmod\_log\_memory\_alloc()

```
void AK_debmod_log_memory_alloc (
    int32_t func_id )
```

print debmod information on function [private function]

##### Author

Marin Rukavina, Mislav Bozicevic

##### Parameters

|                |                     |
|----------------|---------------------|
| <i>func_id</i> | calling function id |
|----------------|---------------------|

##### Returns

void

#### 5.10.2.20 AK\_debmod\_print\_function\_use()

```
void AK_debmod_print_function_use (
    const char * func_name,
    uint8_t in_recur )
```

Print function dependency [private function].

##### Author

Marin Rukavina, Mislav Bozicevic

##### Parameters

|                  |                     |
|------------------|---------------------|
| <i>func_name</i> | function name       |
| <i>in_recur</i>  | called in recursion |

##### Returns

void

#### 5.10.2.21 AK\_fread()

```
size_t AK_fread (
    void * buf,
```



```
size_t size,  
size_t count,  
FILE * fp )
```

Read from a file (see fread) [public function].

#### Author

Marin Rukavina, Mislav Bozicevic

#### Returns

number of items read

#### 5.10.2.22 AK\_free()

```
void AK_free (  
    void * ptr )
```

Free memory at ptr (see free) [public function].

#### Author

Marin Rukavina, Mislav Bozicevic

#### Parameters

|            |                   |
|------------|-------------------|
| <i>ptr</i> | pointer to memory |
|------------|-------------------|

#### Returns

void

#### 5.10.2.23 AK\_fwrite()

```
size_t AK_fwrite (  
    const void * buf,  
    size_t size,  
    size_t count,  
    FILE * fp )
```

Write to a file from a buffer (see fwrite) [public function].

#### Author

Marin Rukavina, Mislav Bozicevic

**Returns**

number of items written

**5.10.2.24 AK\_malloc()**

```
void* AK_malloc (
    size_t size )
```

Allocate memory (see malloc) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|             |                                |
|-------------|--------------------------------|
| <i>size</i> | of memory to allocate in bytes |
|-------------|--------------------------------|

**Returns**

allocated memory or NULL

**5.10.2.25 AK\_mempro\_test()**

```
void AK_mempro_test ( )
```

Test function.

**Author**

Ivan Kristo

**5.10.2.26 AK\_print\_active\_functions()**

```
void AK_print_active_functions ( )
```

Print all detected functions.

**Author**

Marin Rukavina, Mislav Bozicevic

**Returns**

void

#### 5.10.2.27 AK\_print\_function\_use()

```
void AK_print_function_use (
    const char * func_name )
```

Print function dependency [public function].

##### Author

Marin Rukavina, Mislav Bozicevic

##### Parameters

|                  |               |
|------------------|---------------|
| <i>func_name</i> | function name |
|------------------|---------------|

##### Returns

void

#### 5.10.2.28 AK\_print\_function\_uses()

```
void AK_print_function_uses ( )
```

Print function dependency for all functions [public function].

##### Author

Marin Rukavina, Mislav Bozicevic

##### Returns

void

#### 5.10.2.29 AK\_realloc()

```
void* AK_realloc (
    void * ptr,
    size_t size )
```

Reallocates memory (see realloc) [public function].

##### Author

Marin Rukavina, Mislav Bozicevic

**Parameters**

|             |            |
|-------------|------------|
| <i>ptr</i>  | old memory |
| <i>size</i> | new size   |

**Returns**

reallocated memory or NULL

**5.10.2.30 AK\_write\_protect()**

```
void AK_write_protect (
    void * memory )
```

Function write-protects memory [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|               |  |
|---------------|--|
| <i>memory</i> |  |
|---------------|--|

**Returns**

void

**5.10.2.31 AK\_write\_unprotect()**

```
void AK_write_unprotect (
    void * memory )
```

Function write-unprotects memory [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|               |  |
|---------------|--|
| <i>memory</i> |  |
|---------------|--|

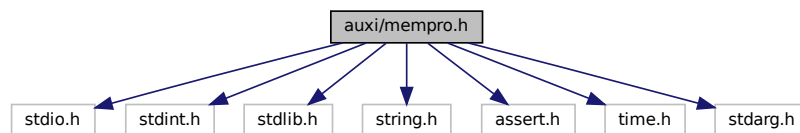
## Returns

void

## 5.11 auxi/mempro.h File Reference

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <time.h>
#include <stdarg.h>
```

Include dependency graph for mempro.h:



### Classes

- struct [AK\\_debmod\\_state](#)  
Global structure that holds all relevant information for the debug mode and related functionality.

### Macros

- `#define NEW(type, type_size) (calloc(type_size, sizeof(type)))`
- `#define AK_INLINE __inline__`
- `#define AK_DEBMOD_ON 0`  
Zero to switch memory protection and debug mode off.
- `#define AK_DEBMOD_PRINT 0`  
Defines if debug mode messages are going to be printed.
- `#define AK_DEBMOD_PAGES_NUM 8192`  
Defines the total of available memory pages for allocation.
- `#define AK_DEBMOD_MAX_WRITE_DETECTIONS (AK_DEBMOD_PAGES_NUM * 10)`  
Defines the maximum number of memory write detections.
- `#define AK_DEBMOD_STACKSIZE AK_DEBMOD_PAGES_NUM`  
Defines the monitored functions stack.
- `#define AK_DEBMOD_MAX_FUNCTIONS 500`  
Defines the maximum number of function names in the application.
- `#define AK_DEBMOD_MAX_FUNC_NAME 80`  
Defines maximum function name length possible.
- `#define AK_PRO AK_debmod_function_prologue(__func__, __FILE__, __LINE__);`  
Mandatory function prologue for all functions (AK\_debmod and related functions are excluded). Put this macro after variable declarations, before any function instruction.
- `#define AK_EPI AK_debmod_function_epilogue(__func__, __FILE__, __LINE__);`  
Mandatory function epilogue for all functions (AK\_debmod and related functions are excluded). Put this macro after last function instruction, before every return statement.

## Functions

- void [AK\\_debmod\\_d](#) ([AK\\_debmod\\_state](#) \*, const char \*)  
*Function prints debug message [private function].*
- void [AK\\_debmod\\_dv](#) ([AK\\_debmod\\_state](#) \*, const char \*,...)  
*Function prints debug message [private function].*
- void [AK\\_debmod\\_enter\\_critical\\_sec](#) ([AK\\_debmod\\_state](#) \*)  
*Reserves ds for use [private function].*
- void [AK\\_debmod\\_leave\\_critical\\_sec](#) ([AK\\_debmod\\_state](#) \*)  
*Makes ds available [private function].*
- [AK\\_debmod\\_state](#) \* [AK\\_debmod\\_init](#) (void)  
*Initializes debug mode structure [public function].*
- void [AK\\_debmod\\_die](#) ([AK\\_debmod\\_state](#) \*)  
*Destroy debug mode state (call before main() exit) [public function].*
- void \* [AK\\_debmod\\_calloc](#) ([AK\\_debmod\\_state](#) \*, uint32\_t)  
*Allocates memory [private function].*
- void [AK\\_debmod\\_free](#) ([AK\\_debmod\\_state](#) \*, void \*)  
*Frees memory allocated with debmod\_alloc [private function].*
- void \* [AK\\_calloc](#) (size\_t, size\_t)  
*Allocates memory (see calloc) [public function].*
- void \* [AK\\_malloc](#) (size\_t)  
*Allocate memory (see malloc) [public function].*
- void [AK\\_free](#) (void \*)  
*Free memory at ptr (see free) [public function].*
- void \* [AK\\_realloc](#) (void \*, size\_t)  
*Reallocates memory (see realloc) [public function].*
- void [AK\\_write\\_protect](#) (void \*)  
*Function write-protects memory [public function].*
- void [AK\\_write\\_unprotect](#) (void \*)  
*Function write-unprotects memory [public function].*
- void [AK\\_check\\_for\\_writes](#) (void)  
*Marks pages dirty if there were writes between calls to this function.*
- int32\_t [AK\\_debmod\\_func\\_id](#) ([AK\\_debmod\\_state](#) \*, const char \*)  
*Returns function id for given func\_name.*
- const char \* [AK\\_debmod\\_func\\_get\\_name](#) ([AK\\_debmod\\_state](#) \*, int32\_t)  
*Lookup function name [private function].*
- int32\_t [AK\\_debmod\\_func\\_add](#) ([AK\\_debmod\\_state](#) \*, const char \*)  
*Adds function name to list [private function].*
- void [AK\\_debmod\\_fstack\\_push](#) ([AK\\_debmod\\_state](#) \*, int32\_t)  
*Push function id on stack [private function].*
- int32\_t [AK\\_debmod\\_fstack\\_pop](#) ([AK\\_debmod\\_state](#) \*)  
*Pops function id from stack [private function].*
- void [AK\\_debmod\\_function\\_current](#) ([AK\\_debmod\\_state](#) \*, int32\_t)  
*Sets current function [private function].*
- void [AK\\_debmod\\_function\\_prologue](#) (const char \*, const char \*, int)  
*Not for direct use (only with macro AK\_PRO). Marks function prologue.*
- void [AK\\_debmod\\_function\\_epilogue](#) (const char \*, const char \*, int)  
*Not for direct use (only with macro AK\_EPI). Marks function epilogue.*
- void [AK\\_debmod\\_log\\_memory\\_alloc](#) (int32\_t)  
*print debmod information on function [private function]*
- void [AK\\_debmod\\_print\\_function\\_use](#) (const char \*, uint8\_t)

- Print function dependency [private function].*
- void [AK\\_print\\_function\\_use](#) (const char \*)
- Print function dependency [public function].*
- void [AK\\_print\\_function\\_uses](#) ()
- Print function dependency for all functions [public function].*
- void [AK\\_print\\_active\\_functions](#) ()
- Print all detected functions.*
- void [AK\\_mempro\\_test](#) ()
- Test function.*

## Variables

- [AK\\_debmod\\_state](#) \* [AK\\_DEBMOD\\_STATE](#)

### 5.11.1 Detailed Description

Data structures, includes, macros and declarations for the memory wrappers and debug mode of Kalashnikov DB.

### 5.11.2 Function Documentation

#### 5.11.2.1 [AK\\_malloc\(\)](#)

```
void* AK_malloc (
    size_t num,
    size_t size )
```

Allocates memory (see [calloc](#)) [public function].

#### Author

Marin Rukavina, Mislav Bozicevic

#### Parameters

|             |                     |
|-------------|---------------------|
| <i>num</i>  | number of elements  |
| <i>size</i> | of element in bytes |

#### Returns

allocated memory or NULL

#### 5.11.2.2 AK\_check\_for\_writes()

```
void AK_check_for_writes (
    void )
```

Marks pages dirty if there were writes between calls to this function.

##### Author

Marin Rukavina, Mislav Bozicevic

##### Returns

void

#### 5.11.2.3 AK\_debmod\_calloc()

```
void* AK_debmod_calloc (
    AK_debmod_state * ds,
    uint32_t size )
```

Allocates memory [private function].

##### Author

Marin Rukavina, Mislav Bozicevic

##### Parameters

|             |                      |
|-------------|----------------------|
| <i>ds</i>   | debug mode state     |
| <i>size</i> | in bytes to allocate |

##### Returns

pointer to allocated memory or NULL

#### 5.11.2.4 AK\_debmod\_d()

```
void AK_debmod_d (
    AK_debmod_state * ds,
    const char * message )
```

Function prints debug message [private function].

##### Author

Marin Rukavina, Mislav Bozicevic



## Parameters

|                |                  |
|----------------|------------------|
| <i>ds</i>      | debug mode state |
| <i>message</i> | string to print  |

## Returns

void

## 5.11.2.5 AK\_debmod\_die()

```
void AK_debmod_die (
    AK_debmod_state * ds )
```

Destroy debug mode state (call before main() exit) [public function].

## Author

Marin Rukavina, Mislav Bozicevic

## Parameters

|           |                  |
|-----------|------------------|
| <i>ds</i> | debug mode state |
|-----------|------------------|

## Returns

void

## 5.11.2.6 AK\_debmod\_dv()

```
void AK_debmod_dv (
    AK_debmod_state * ds,
    const char * format,
    ... )
```

Function prints debug message [private function].

## Author

Marin Rukavina, Mislav Bozicevic

## Parameters

|               |                           |
|---------------|---------------------------|
| <i>ds</i>     | debug mode state          |
| <i>format</i> | format string like printf |

**Returns**

void

**5.11.2.7 AK\_debmod\_enter\_critical\_sec()**

```
void AK_debmod_enter_critical_sec (
    AK_debmod_state * ds )
```

Reserves ds for use [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|           |                  |
|-----------|------------------|
| <i>ds</i> | debug mode state |
|-----------|------------------|

**Returns**

void

**5.11.2.8 AK\_debmod\_free()**

```
void AK_debmod_free (
    AK_debmod_state * ds,
    void * memory )
```

Frees memory allocated with debmod\_alloc [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|               |                  |
|---------------|------------------|
| <i>ds</i>     | debug mode state |
| <i>memory</i> |                  |

**Returns**

void

### 5.11.2.9 AK\_debmod\_fstack\_pop()

```
int32_t AK_debmod_fstack_pop (
    AK_debmod_state * ds )
```

Pops function id from stack [private function].

#### Author

Marin Rukavina, Mislav Bozicevic

#### Parameters

|           |                  |
|-----------|------------------|
| <i>ds</i> | debug mode state |
|-----------|------------------|

#### Returns

function id popped

### 5.11.2.10 AK\_debmod\_fstack\_push()

```
void AK_debmod_fstack_push (
    AK_debmod_state * ds,
    int32_t func_id )
```

Push function id on stack [private function].

#### Author

Marin Rukavina, Mislav Bozicevic

#### Parameters

|  |                  |
|--|------------------|
| <i>ds</i>                              | debug mode state |
| <i>func</i> <sub>↔</sub><br><i>_id</i> | function id      |

#### Returns

void

### 5.11.2.11 AK\_debmod\_func\_add()

```
int32_t AK_debmod_func_add (
    AK_debmod_state * ds,
    const char * func_name )
```

Adds function name to list [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                  |                  |
|------------------|------------------|
| <i>ds</i>        | debug mode state |
| <i>func_name</i> |                  |

**Returns**

id for added function name

**5.11.2.12 AK\_debmod\_func\_get\_name()**

```
const char* AK_debmod_func_get_name (
    AK_debmod_state * ds,
    int32_t function_id )
```

Lookup function name [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                          |                  |
|--------------------------|------------------|
| <i>ds</i>                | debug mode state |
| <i>function↔<br/>_id</i> |                  |

**Returns**

function name for given function\_id

**5.11.2.13 AK\_debmod\_func\_id()**

```
int32_t AK_debmod_func_id (
    AK_debmod_state * ds,
    const char * func_name )
```

Returns function id for given func\_name.

**Author**

Marin Rukavina, Mislav Bozicevic

## Parameters

|                  |                                  |
|------------------|----------------------------------|
| <i>ds</i>        | debug mode state                 |
| <i>func_name</i> | function name [private function] |

## Returns

function id

## 5.11.2.14 AK\_debmod\_function\_current()

```
void AK_debmod_function_current (
    AK_debmod_state * ds,
    int32_t new_function_id )
```

Sets current function [private function].

## Author

Marin Rukavina, Mislav Bozicevic

## Parameters

|                        |                  |
|------------------------|------------------|
| <i>ds</i>              | debug mode state |
| <i>new_function_id</i> |                  |

## Returns

void

## 5.11.2.15 AK\_debmod\_function\_epilogue()

```
void AK_debmod_function_epilogue (
    const char * func_name,
    const char * source_file,
    int source_line )
```

Not for direct use (only with macro AK\_EPI). Marks function epilogue.

## Author

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                    |   |
|--------------------|---|
| <i>func_name</i>   | function name as in source              |
| <i>source_file</i> | file name where function is defined     |
| <i>source_line</i> | line from which this function is called |

**Returns**

void

**5.11.2.16 AK\_debmod\_function\_prologue()**

```
void AK_debmod_function_prologue (
    const char * func_name,
    const char * source_file,
    int source_line )
```

Not for direct use (only with macro AK\_PRO). Marks function prologue.

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                    |   |
|--------------------|---|
| <i>func_name</i>   | function name as in source              |
| <i>source_file</i> | file name where function is defined     |
| <i>source_line</i> | line from which this function is called |

**Returns**

void

**5.11.2.17 AK\_debmod\_init()**

```
AK_debmod_state* AK_debmod_init (
    void )
```

Initializes debug mode structure [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Returns**

initialized debug mode state

#### 5.11.2.18 AK\_debmod\_leave\_critical\_sec()

```
void AK_debmod_leave_critical_sec (
    AK_debmod_state * ds )
```

Makes ds available [private function].

##### Author

Marin Rukavina, Mislav Bozicevic

##### Parameters

|           |                  |
|-----------|------------------|
| <i>ds</i> | debug mode state |
|-----------|------------------|

##### Returns

void

#### 5.11.2.19 AK\_debmod\_log\_memory\_alloc()

```
void AK_debmod_log_memory_alloc (
    int32_t func_id )
```

print debmod information on function [private function]

##### Author

Marin Rukavina, Mislav Bozicevic

##### Parameters

|  |                     |
|--|---------------------|
| <i>func</i> <sub>↔</sub><br><i>_id</i> | calling function id |
|--|---------------------|

##### Returns

void

#### 5.11.2.20 AK\_debmod\_print\_function\_use()

```
void AK_debmod_print_function_use (
    const char * func_name,
    uint8_t in_recur )
```

Print function dependency [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                  |                     |
|------------------|---------------------|
| <i>func_name</i> | function name       |
| <i>in_recur</i>  | called in recursion |

**Returns**

void

**5.11.2.21 AK\_free()**

```
void AK_free (
    void * ptr )
```

Free memory at ptr (see free) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|            |                   |
|------------|-------------------|
| <i>ptr</i> | pointer to memory |
|------------|-------------------|

**Returns**

void

**5.11.2.22 AK\_malloc()**

```
void* AK_malloc (
    size_t size )
```

Allocate memory (see malloc) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic



**Parameters**

|             |                                |
|-------------|--------------------------------|
| <i>size</i> | of memory to allocate in bytes |
|-------------|--------------------------------|

**Returns**

allocated memory or NULL

**5.11.2.23 AK\_mempro\_test()**

```
void AK_mempro_test ( )
```

Test function.

**Author**

Ivan Kristo

**5.11.2.24 AK\_print\_active\_functions()**

```
void AK_print_active_functions ( )
```

Print all detected functions.

**Author**

Marin Rukavina, Mislav Bozicevic

**Returns**

void

**5.11.2.25 AK\_print\_function\_use()**

```
void AK_print_function_use (
    const char * func_name )
```

Print function dependency [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|                  |               |
|------------------|---------------|
| <i>func_name</i> | function name |
|------------------|---------------|

**Returns**

void

**5.11.2.26 AK\_print\_function\_uses()**

```
void AK_print_function_uses ( )
```

Print function dependency for all functions [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Returns**

void

**5.11.2.27 AK\_realloc()**

```
void* AK_realloc (
    void * ptr,
    size_t size )
```

Reallocates memory (see realloc) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

|             |            |
|-------------|------------|
| <i>ptr</i>  | old memory |
| <i>size</i> | new size   |

**Returns**

reallocated memory or NULL

#### 5.11.2.28 AK\_write\_protect()

```
void AK_write_protect (
    void * memory )
```

Function write-protects memory [public function].

##### Author

Marin Rukavina, Mislav Bozicevic

##### Parameters

|               |  |
|---------------|--|
| <i>memory</i> |  |
|---------------|--|

##### Returns

void

#### 5.11.2.29 AK\_write\_unprotect()

```
void AK_write_unprotect (
    void * memory )
```

Function write-unprotects memory [public function].

##### Author

Marin Rukavina, Mislav Bozicevic

##### Parameters

|               |  |
|---------------|--|
| <i>memory</i> |  |
|---------------|--|

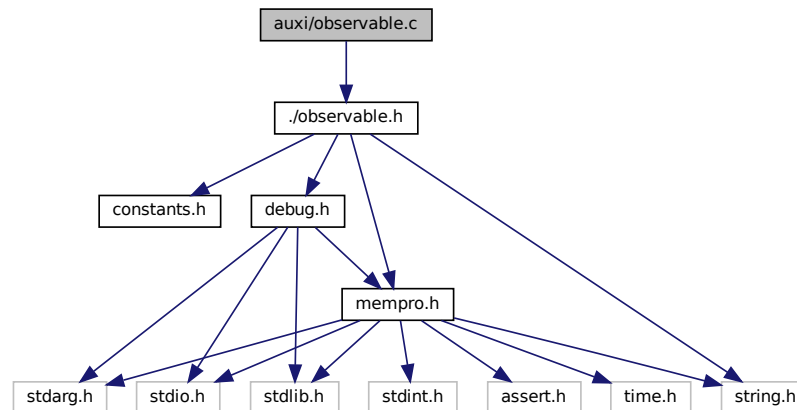
##### Returns

void

## 5.12 auxi/observable.c File Reference

```
#include "../observable.h"
```

Include dependency graph for observable.c:



## Classes

- struct [\\_notifyDetails](#)
- struct [TypeObservable](#)
- struct [TypeObserver](#)

## Typedefs

- typedef struct [\\_notifyDetails](#) **NotifyDetails**
- typedef struct [TypeObservable](#) **AK\_TypeObservable**
- typedef struct [TypeObserver](#) **AK\_TypeObserver**
- typedef struct [TypeObserver](#) **AK\_TypeObserver\_Second**

## Enumerations

- enum **NotifyType** { **ERROR**, **INFO**, **WARMING** }

## Functions

- [AK\\_observable](#) \* [AK\\_init\\_observable](#) (void \*AK\_observable\_type, AK\_ObservableType\_Enum AK\_observable\_type\_Def, void \*AK\_custom\_action)  
*Function for initializing observable object.*
- [AK\\_observer](#) \* [AK\\_init\\_observer](#) (void \*observer\_type, void(\*observer\_type\_event\_handler)(void \*, void \*, AK\_ObservableType\_Enum))  
*Function for initializing observer object.*
- char \* **AK\_get\_message** ([AK\\_TypeObservable](#) \*self)
- int **AK\_custom\_register\_observer** ([AK\\_TypeObservable](#) \*self, [AK\\_observer](#) \*observer)
- int **AK\_custom\_unregister\_observer** ([AK\\_TypeObservable](#) \*self, [AK\\_observer](#) \*observer)
- void **AK\_set\_notify\_info\_details** ([AK\\_TypeObservable](#) \*self, NotifyType type, char \*message)
- int **AK\_custom\_action** (void \*data)
- [AK\\_TypeObservable](#) \* **init\_observable\_type** ()

- void **handle\_ak\_custom\_type** ([AK\\_TypeObserver](#) \*observer, [AK\\_TypeObservable](#) \*observable)
- void **custom\_observer\_event\_handler** (void \*observer, void \*observable, [AK\\_ObservableType\\_Enum](#) AK\_ObservableType\_Def)
- [AK\\_TypeObserver](#) \* **init\_observer\_type** (void \*observable)
- [AK\\_TypeObserver](#) \* **init\_observer\_type\_second** ()
- void [AK\\_observable\\_test](#) ()  
*Function that runs tests for observable pattern.*
- void **AK\_observable\_pattern** ()

### 5.12.1 Detailed Description

File that provides implementations of functions for observable pattern

### 5.12.2 Function Documentation

#### 5.12.2.1 AK\_init\_observable()

```
AK\_observable* AK_init_observable (
    void * AK_observable_type,
    AK\_ObservableType\_Enum AK_ObservableType_Def,
    void * AK_custom_action )
```

Function for initializing observable object.

#### Author

Ivan Pusic

#### Returns

Pointer to new observable object

#### 5.12.2.2 AK\_init\_observer()

```
AK\_observer* AK_init_observer (
    void * observer_type,
    void(*) (void *, void *, AK\_ObservableType\_Enum) observer_type_event_handler )
```

Function for initializing observer object.

#### Author

Ivan Pusic

#### Returns

Pointer to new observer object

### 5.12.2.3 AK\_observable\_test()

```
void AK_observable_test ( )
```

Function that runs tests for observable pattern.

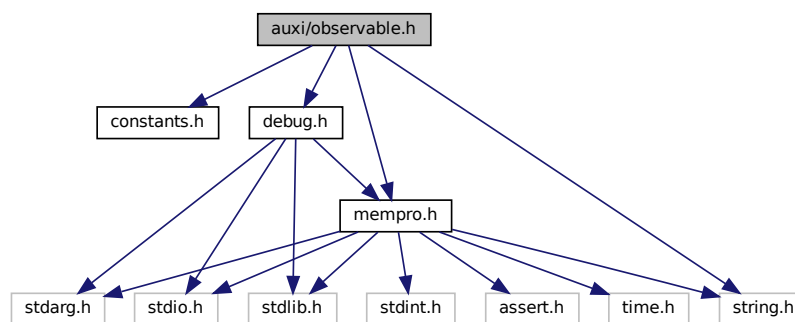
Author

Ivan Pusic

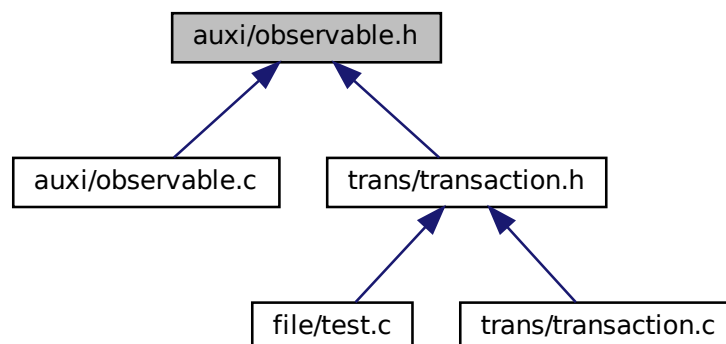
## 5.13 auxi/observable.h File Reference

```
#include "constants.h"
#include "debug.h"
#include "mempro.h"
#include <string.h>
```

Include dependency graph for observable.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [Observer](#)  
*Structure defines functions for observer object.*
- struct [Observable](#)  
*Structure defines functions for observable object.*

## Typedefs

- typedef struct [Observer](#) **AK\_observer**
- typedef struct [Observable](#) **AK\_observable**

## Enumerations

- enum **AK\_ObservableType\_Enum** { **AK\_TRANSACTION**, **AK\_TRIGGER**, **AK\_CUSTOM\_FIRST**, **AK\_CUSTOM\_SECOND** }

## Functions

- [AK\\_observer](#) \* [AK\\_init\\_observer](#) (void \*observable\_type, void(\*observable\_type\_event\_handler)(void \*, void \*, AK\_ObservableType\_Enum))  
*Function for initializing observer object.*
- [AK\\_observable](#) \* [AK\\_init\\_observable](#) (void \*AK\_observable\_type, AK\_ObservableType\_Enum AK\_observable\_type\_Def, void \*AK\_custom\_action)  
*Function for initializing observable object.*
- void [AK\\_observable\\_test](#) ()  
*Function that runs tests for observable pattern.*
- void **AK\_observable\_pattern** ()

### 5.13.1 Detailed Description

Header file that provides data structures and declarations of functions for observable pattern

### 5.13.2 Function Documentation

#### 5.13.2.1 [AK\\_init\\_observable\(\)](#)

```
AK\_observable* AK_init_observable (
    void * AK_observable_type,
    AK_ObservableType_Enum AK_ObservableType_Def,
    void * AK_custom_action )
```

Function for initializing observable object.

#### Author

Ivan Pusic

#### Returns

Pointer to new observable object

### 5.13.2.2 AK\_init\_observer()

```
AK_observer* AK_init_observer (
    void * observer_type,
    void(*) (void *, void *, AK_ObservableType_Enum) observer_type_event_handler )
```

Function for initializing observer object.

#### Author

Ivan Pusic

#### Returns

Pointer to new observer object

### 5.13.2.3 AK\_observable\_test()

```
void AK_observable_test ( )
```

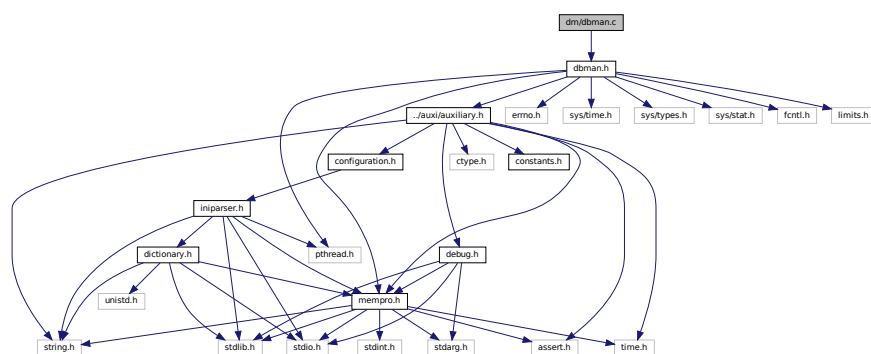
Function that runs tests for observable pattern.

#### Author

Ivan Pusic

## 5.14 dm/dbman.c File Reference

```
#include "dbman.h"
Include dependency graph for dbman.c:
```





## Functions

- int [AK\\_init\\_db\\_file](#) (int size)  
*Function that initializes a new database file named DB\_FILE. It opens database file. New block is allocated. In this block type of header is set to FREE\_INT, attribute names are set to FREE\_CHAR, integrities are set to FREE\_INT, constraint names are set to FREE\_CHAR, constraint names and codes are set to FREE\_CHAR. Type, address and size of tuples are set to FREE\_INT. Data in block is set to FREE\_CHAR. Type of block is BLOCK\_TYPE\_FREE, it is not chained and id of last tuple is 0.*
- int [AK\\_get\\_allocation\\_set](#) (int \*allocationSet, int fromWhere, int gaplength, int numRequestedBlocks, [AK\\_allocation\\_set\\_mode](#) mode, int target)  
*Function prepare demanded sets from allocation table.*
- void [AK\\_allocationtable\\_dump](#) (int verbosity)  
*Dumps the allocation table from the global allocation bit-vector onto standard output.*
- void [AK\\_blocktable\\_dump](#) (int verbosity)  
*Dumps the bit-table from the global allocation bit-vector onto standard output.*
- int [AK\\_blocktable\\_flush](#) ()  
*Function flushes bitmask table to disk.*
- void [AK\\_allocate\\_block\\_activity\\_modes](#) ()  
*Allocation of array which will contain information about which blocks are being accessed. Creates an array. Each element of this array will correspond to one initialized block. For more info, see explanation in [dbman.h](#).*
- int [AK\\_blocktable\\_get](#) ()  
*Function gets allocation table from disk.*
- int [fsize](#) (FILE \*fp)  
*Helper function to determine file size.*
- int [AK\\_init\\_allocation\\_table](#) ()  
*Function that initializes allocation table, write it to disk and cache in memory.*
- [AK\\_block](#) \* [AK\\_init\\_block](#) ()  
*Function that initializes new block.*
- int [AK\\_print\\_block](#) ([AK\\_block](#) \*block, int num, char \*gg, FILE \*fpp)  
*Function that dumps block.*
- int [AK\\_allocate\\_blocks](#) (FILE \*db, [AK\\_block](#) \*block, int FromWhere, int HowMany)  
*Function that allocates new blocks by placing them to appropriate place and then update last initialized index.*
- [AK\\_block](#) \* [AK\\_read\\_block](#) (int address)  
*Function that reads a block at a given address (block number less than db\_file\_size). New block is allocated. Database file is opened. Position is set to provided address block. At the end function reads file from that position. Completely thread-safe.*
- int [AK\\_write\\_block](#) ([AK\\_block](#) \*block)  
*Function writes a block to DB file. Database file is opened. Position is set to provided address block. Block is written to provided address. Completely thread-safe.*
- int [AK\\_copy\\_header](#) ([AK\\_header](#) \*header, int \*blockSet, int blockSetSize)  
*Function copy header to blocks. Completely thread-safe.*
- int \* [AK\\_get\\_extent](#) (int start\_address, int desired\_size, [AK\\_allocation\\_set\\_mode](#) \*mode, int border, int target, [AK\\_header](#) \*header, int gl)  
*Function allocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.*
- int \* [AK\\_increase\\_extent](#) (int start\_address, int add\_size, [AK\\_allocation\\_set\\_mode](#) \*mode, int border, int target, [AK\\_header](#) \*header, int gl)  
*Function allocates new blocks for increasing extent size.*
- int [AK\\_new\\_extent](#) (int start\_address, int old\_size, int extent\_type, [AK\\_header](#) \*header)  
*Function allocates new extent of blocks. If argument "old\_size" is 0 than size of extent is INITIAL\_EXTENT\_SIZE. Otherwise, resize factor is set according to type of extent. If writing of block is successful, number of blocks is incremented.*
- int [AK\\_new\\_segment](#) (char \*name, int type, [AK\\_header](#) \*header)

Function that allocates new segment of extents. In this phase of implementation, only extents containing INITIAL\_EXTENT\_SIZE blocks can be allocated. If extent is successfully allocated, number of allocated extents is incremented and function goes to next block after allocated extent. Otherwise, function moves to INITIAL\_EXTENT\_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK\_free.

- **AK\_header \* AK\_create\_header** (char \*attribute\_name, int type, int integrity, char \*constr\_name, char \*constr\_code)

Function for creating header and initialize integrity, constraint name and constraint code with parameter values of function.

- **void AK\_insert\_entry** (AK\_block \*block\_address, int type, void \*entry\_data, int i)

Function for inserting entry in tuple\_dict and data of a block. Address, type and size of catalog\_tuple\_dict are set. Free space of block is also set.

- **int AK\_init\_system\_tables\_catalog** (int relation, int attribute, int index, int view, int sequence, int function, int function\_arguments, int trigger, int trigger\_conditions, int db, int db\_obj, int user, int group, int user\_group, int user\_right, int group\_right, int constraint, int constraintNull, int constraintCheck, int constraintUnique, int reference)

Function initialises the sytem table catalog and writes the result in first (0) block in db\_file. Catalog block, catalog header name, catalog header address are allocated. Address, type, chained\_with and AK\_free\_space attributes are initialized. Names of various database elements are written in block.

- **void AK\_memset\_int** (void \*block, int value, size\_t num)

Function that sets the first num ints of a block of memory to the specified value.

- **int AK\_register\_system\_tables** (int relation, int attribute, int index, int view, int sequence, int function, int function\_arguments, int trigger, int trigger\_conditions, int db, int db\_obj, int user, int group, int user\_group, int user\_right, int group\_right, int constraint, int constraintNull, int constraintCheck, int constraintUnique, int reference)

Function that registers system tables. Block at the given address is read. Various data from function arguments are written in block about different database elements.

- **int AK\_init\_system\_catalog** ()

Function initializes the system catalog. Headers for system tables are defined. Segments for those system tables are allocated. Above function [AK\\_register\\_system\\_tables\(\)](#) to register system tables.

- **int AK\_delete\_block** (int address)

Function deletes a block by a given block address (resets the header and data). Types, integrities, constraint names, constraint codes are set to "AK\_free" values. In tuple dictionary type, address and size are set to FREE\_INT values. Data of block is set to FREE\_CHAR.

- **int AK\_delete\_extent** (int begin, int end)

Function deletes an extent between begin and end blocks.

- **int AK\_delete\_segment** (char \*name, int type)

- **int AK\_init\_disk\_manager** ()

- **void AK\_allocationbit\_test** ()

- **void AK\_allocationtable\_test** ()

- **void AK\_thread\_safe\_block\_access\_test** ()

This function tests thread safe reading and writing to blocks. There is N writing and N reading threads, which are going through iterations. Each reading thread should read the data (character) that was set by last writing thread.

- **void \* AK\_read\_block\_for\_testing** (void \*address)

This function is only for testing. It has to be there, because pthread\_create only accepts void\* function\_name (void \*) function format. So AK\_read\_block is no-go for pthread\_create.

- **void \* AK\_write\_block\_for\_testing** (void \*block)

This function is only for testing. It has to be there, because pthread\_create only accepts void\* function\_name (void \*) function format. So AK\_write\_block is no-go for pthread\_create.

## Variables

- pthread\_mutex\_t **fileLockMutex** = PTHREAD\_MUTEX\_INITIALIZER
- char **test\_lastCharacterWritten** = '\0'

*This variable is used only when TEST\_MODE is ON! It is used only for testing functionality of [AK\\_thread\\_safe\\_block\\_access\\_test\(\)](#) function. It will contain first character of last written block. When reading thread reads the block (written by some other thread), it will compare the first character from this block to character contained in this variables. If they don't match, then the error occurred! It is assumed that the same block is being written to and read from (just like [AK\\_thread\\_safe\\_block\\_access\\_test](#) function works!)*

- `int test_threadSafeBlockAccessSucceeded = 1`

*Used in combination with `test_lastCharacterWritten`. Will give the answer to question: "Has [AK\\_thread\\_safe\\_block\\_access\\_test](#) succeeded?" 0 means NO, 1 means YES.*

### 5.14.1 Detailed Description

Defines functions for the disk manager

### 5.14.2 Function Documentation

#### 5.14.2.1 AK\_allocate\_block\_activity\_modes()

```
void AK_allocate_block_activity_modes ( )
```

Allocation of array which will contain information about which blocks are being accessed. Creates an array. Each element of this array will correspond to one initialized block. For more info, see explanation in [dbman.h](#).

Author

Domagoj Šitum

#### 5.14.2.2 AK\_allocate\_blocks()

```
int AK_allocate_blocks (
    FILE * db,
    AK_block * block,
    int FromWhere,
    int HowMany )
```

Function that allocates new blocks by placing them to appropriate place and then update last initialized index.

Author

Markus Schatten , rearranged by dv

Returns

EXIT\_SUCCESS if the file has been written to disk, EXIT\_ERROR otherwise

#### 5.14.2.3 AK\_allocationtable\_dump()

```
void AK_allocationtable_dump (
    int verbosity )
```

Dumps the allocation table from the global allocation bit-vector onto standard output.

Author

dv

**Parameters**

|                  |   |
|------------------|---|
| <i>verbosity</i> | level of verbosity (1 - minimal, 0 - no output) |
|------------------|---|

**5.14.2.4 AK\_blocktable\_dump()**

```
void AK_blocktable_dump (
    int verbosity )
```

Dumps the bit-table from the global allocation bit-vector onto standard output.

**Author**

dv

**Parameters**

|                  |   |
|------------------|---|
| <i>verbosity</i> | level of verbosity (1 - verbose, 0 - minimal) |
|------------------|---|

**5.14.2.5 AK\_blocktable\_flush()**

```
int AK_blocktable_flush ( )
```

Function flushes bitmask table to disk.

**Author**

dv

**Returns**

EXIT\_SUCCESS if the file has been written to disk, EXIT\_ERROR otherwise

**5.14.2.6 AK\_blocktable\_get()**

```
int AK_blocktable_get ( )
```

Function gets allocation table from disk.

**Author**

dv

**Returns**

EXIT\_SUCCESS if the file has been taken from disk, EXIT\_ERROR otherwise

#### 5.14.2.7 AK\_copy\_header()

```
int AK_copy_header (
    AK_header * header,
    int * blockSet,
    int blockSetSize )
```

Function copy header to blocks. Completely thread-safe.

##### Author

Nikola Bakoš, updated by Dino Laktašić (fixed header BUG), refurbished by dv

##### Parameters

|                     |  |
|---------------------|--|
| <i>header</i>       | Pointer to header which will be copied into each block in blockSet |
| <i>blockSet</i>     | Pointer to array of block addresses into which to copy header      |
| <i>blockSetSize</i> | Number of blocks in blockSet                                       |

##### Returns

number of performed header copy

#### 5.14.2.8 AK\_create\_header()

```
AK_header* AK_create_header (
    char * attribute_name,
    int type,
    int integrity,
    char * constr_name,
    char * contr_code )
```

Function for creating header and initialize integrity, constraint name and constraint code with parameter values of function.

##### Author

Matija Novak

##### Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>name</i>        | name of the attribute           |
| <i>type</i>        | type of the attribute           |
| <i>integrity</i>   | standard integrity constraint   |
| <i>constr_name</i> | extra integrity constraint name |
| <i>contr_code</i>  | extra integrity constraint code |

**Returns**

[AK\\_header](#)

**5.14.2.9 AK\_delete\_block()**

```
int AK_delete_block (
    int address )
```

Function deletes a block by a given block address (resets the header and data). Types, integrities, constraint names, constraint codes are set to "AK\_free" values. In tuple dictionary type, address and size are set to FREE\_INT values. Data of block is set to FREE\_CHAR.

**Author**

Markus Schatten

**Parameters**

|                |                                    |
|----------------|------------------------------------|
| <i>address</i> | address of the block to be deleted |
|----------------|------------------------------------|

**Returns**

returns EXIT\_SUCCESS if deletion successful, else EXIT\_ERROR

**5.14.2.10 AK\_delete\_extent()**

```
int AK_delete_extent (
    int begin,
    int end )
```

Function deletes an extent between begin and end blocks.

**Author**

Dejan Samboljæ

**Parameters**

|              |                                 |
|--------------|---------------------------------|
| <i>begin</i> | address of extent's first block |
| <i>end</i>   | address of extent's last block  |

**Returns**

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

### 5.14.2.11 AK\_delete\_segment()

```
int AK_delete_segment (
    char * name,
    int type )
```

#### Author

Mislav Ćakariæ

#### Parameters

|             |                     |
|-------------|---------------------|
| <i>name</i> | name of the segment |
| <i>type</i> | type of the segment |

#### Returns

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

### 5.14.2.12 AK\_get\_allocation\_set()

```
int AK_get_allocation_set (
    int * allocationSet,
    int fromWhere,
    int gaplength,
    int numRequestedBlocks,
    AK_allocation_set_mode mode,
    int target )
```

Function prepare demanded sets from allocation table.

#### Author

dv

#### Parameters

|                           |  |
|---------------------------|--|
| <i>allocationSet</i>      | Pointer to array which will be filled and represent the allocation set   |
| <i>fromWhere</i>          | Has meaning only if mode is SEQUENCE. It describes from which address searching starts.                              |
| <i>gaplength</i>          | Tells how many used blocks can be tolerated in allocation set  |
| <i>numRequestedBlocks</i> | Tells how many AK_free blocks have been requested  |
| <i>mode</i>               | Defines how to obtain set of indexes to AK_free addresses  |
| <i>target</i>             | Has meaning just if mode is AROUND: set will be as close as possible to the requested target address from both sides |

**Returns**

the first element of the allocation set

**5.14.2.13 AK\_get\_extent()**

```
int* AK_get_extent (
    int start_address,
    int desired_size,
    AK_allocation_set_mode * mode,
    int border,
    int target,
    AK_header * header,
    int gl )
```

Function allocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.

**Author**

dv

**Parameters**

|                               |  |
|-------------------------------|--|
| <i>start_address</i>          | address (block number) to start searching for sufficient space   |
| <i>desired_size</i>           | number of desired blocks   |
| <i>AK_allocation_set_mode</i> | a way of trying to find AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND |
| <i>border</i>                 | number of allocated blocks gap   |
| <i>target</i>                 | block address around which other blocks have to be searched  |
| <i>header</i>                 | pointer to header that should be written to the new extent (all blocks)  |
| <i>int</i>                    | gl gap size  |

**Returns**

pointer to set of allocated block addresses

vars for loop [for]

if some blocks are not successfully allocated, which means that the extend allocation has FAILED

**5.14.2.14 AK\_increase\_extent()**

```
int* AK_increase_extent (
    int start_address,
    int add_size,
    AK_allocation_set_mode * mode,
    int border,
    int target,
    AK_header * header,
    int gl )
```

Function allocates new blocks for increasing extent size.



**Author**

dv

**Parameters**

|                               |  |
|-------------------------------|--|
| <i>start_address</i>          | first address of extent that is subject of increasing  |
| <i>add_size</i>               | number how many new blocks is to be added to existing extent   |
| <i>AK_allocation_set_mode</i> | a way of trying to find AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND |
| <i>border</i>                 | number of allocated blocks gap   |
| <i>target</i>                 | block address around which other blocks have to be searched  |
| <i>header</i>                 | pointer to header that should be written to the new extent (all blocks)  |
| <i>int</i>                    | gl gap size  |

**Returns**

pointer to set of allocated block addresses

**5.14.2.15 AK\_init\_allocation\_table()**

```
int AK_init_allocation_table ( )
```

Function that initializes allocation table, write it to disk and cache in memory.

**Author**

dv

**Returns**

EXIT\_SUCCESS if the file has been written to disk, EXIT\_ERROR otherwise

**5.14.2.16 AK\_init\_block()**

```
AK_block* AK_init_block ( )
```

Function that initializes new block.

**Author**

Markus Schatten , rearranged by dv

**Returns**

pointer to block allocated in memory

#### 5.14.2.17 AK\_init\_db\_file()

```
int AK_init_db_file (
    int size )
```

Function that initializes a new database file named DB\_FILE. It opens database file. New block is allocated. In this block type of header is set to FREE\_INT, attribute names are set to FREE\_CHAR, integrities are set to FREE\_INT, constraint names are set to FREE\_CHAR, constraint names and codes are set to FREE\_CHAR. Type, address and size of tuples are set to FREE\_INT. Data in block is set to FREE\_CHAR. Type of block is BLOCK\_TYPE\_FREE, it is not chained and id of last tuple is 0.

##### Author

Markus Schatten

##### Parameters

|      |                               |
|------|-------------------------------|
| size | size of new file in in blocks |
|------|-------------------------------|

##### Returns

EXIT\_SUCCESS if the file has been written to disk, EXIT\_ERROR otherwise

#### 5.14.2.18 AK\_init\_disk\_manager()

```
int AK_init_disk_manager ( )
```

##### Author

Markus Schatten

##### Returns

Function that calls functions [AK\\_init\\_db\\_file\(\)](#) and [AK\\_init\\_system\\_catalog\(\)](#) to initialize disk manager. It also calls [AK\\_allocate\\_array\\_currently\\_accessed\\_blocks\(\)](#) to allocate memory needed for thread-safe reading and writing to disk.

#### 5.14.2.19 AK\_init\_system\_catalog()

```
int AK_init_system_catalog ( )
```

Function initializes the system catalog. Headers for system tables are defined. Segments for those system tables are allocated. Above function [AK\\_register\\_system\\_tables\(\)](#) to register system tables.

##### Author

Miroslav Policki

##### Returns

EXIT\_SUCCESS if the system catalog has been successfully initialized, EXIT\_ERROR otherwise

## 5.14.2.20 AK\_init\_system\_tables\_catalog()

```

int AK_init_system_tables_catalog (
    int relation,
    int attribute,
    int index,
    int view,
    int sequence,
    int function,
    int function_arguments,
    int trigger,
    int trigger_conditions,
    int db,
    int db_obj,
    int user,
    int group,
    int user_group,
    int user_right,
    int group_right,
    int constraint,
    int constraintNull,
    int constraintCheck,
    int constraintUnique,
    int reference )

```

Function initialises the sytem table catalog and writes the result in first (0) block in db\_file. Catalog block, catalog header name, catalog header address are allocated. Address, type, chained\_with and AK\_free\_space attributes are initialized. Names of various database elements are written in block.

## Author

Matija Novak

## Parameters

|                           |  |
|---------------------------|--|
| <i>relation</i>           | address of system table of relation in db_file                     |
| <i>attribute</i>          | address of system table of attribute in db_file                    |
| <i>index</i>              | address of system table of index in db_file                        |
| <i>view</i>               | address of system table of view in db_file                         |
| <i>sequence</i>           | address of system table of sequence in db_file                     |
| <i>function</i>           | address of system table of function in db_file                     |
| <i>function_arguments</i> | address of system table of function_arguments in db_file           |
| <i>trigger</i>            | address of system table of trigger in db_file                      |
| <i>trigger_conditions</i> | address of system table of trigger_conditions in db_file           |
| <i>db</i>                 | address of system table of db in db_file                           |
| <i>db_obj</i>             | address of system table of db_obj in db_file                       |
| <i>user</i>               | address of system table of user in db_file                         |
| <i>group</i>              | address of system table of group in db_file                        |
| <i>user_group</i>         | address of system table of users associated with groups in db_file |
| <i>user_right</i>         | address of system table of user right in db_file                   |
| <i>group_right</i>        | address of system table of group right in db_file                  |
| <i>constraint</i>         | address of system table of constraint in db_file                   |
| <i>constraintNull</i>     | address of system table of constraintNull in db_file               |
| <i>constraintCheck</i>    | system table address for check constraint                          |
| <i>reference</i>          | address of system table of reference in db_file                    |

**Returns**

EXIT\_SUCCESS if initialization was succesful if not returns EXIT\_ERROR

first header attribute of catalog\_block

second attribute of catalog\_block

initialize other elements of block (adress, type, chained\_with, AK\_free\_space)

using as an address for the first AK\_free space in block->data

merge catalog\_heder with heders created before

**5.14.2.21 AK\_insert\_entry()**

```
void AK_insert_entry (
    AK_block * block_address,
    int type,
    void * entry_data,
    int i )
```

Function for inserting entry in tuple\_dict and data of a block. Address, type and size of catalog\_tuple\_dict are set. Free space of block is also set.

**Author**

Matija Novak

**Parameters**

|                     |   |
|---------------------|---|
| <i>block_adress</i> | adress of a block in which we want insert data                                |
| <i>type</i>         | type of entry_data  |
| <i>entry_data</i>   | (char) data which is inserted, can be int but must first be converted to char |
| <i>i</i>            | (int) adress in tuple_dict array (example block_address->tuple_dict[i])       |

**Returns**

No return value because it gets the address of an block like a function parameter and works directly with the original block

copy data into bloc->data on start position bloc->AK\_free\_space

address of entry data in block->data

calculate next AK\_free space for the next entry data

```
sizeof(entry_data)+1); /// (sizeof(int));
```

no need for "+strlen(entry\_data)" while "+1" is like "new line"

type of entry data

size of entry data

copy tuple\_dict to block->tuple\_dict[i] must use & becouse tuple\_dict[i] is value and catalog\_tuple\_dict adress

#### 5.14.2.22 AK\_memset\_int()

```
void AK_memset_int (
    void * block,
    int value,
    size_t num )
```

Function that sets the first num ints of a block of memory to the specified value.

##### Author

Miroslav Policki

##### Parameters

|              |   |
|--------------|---|
| <i>block</i> | pointer to the block of memory to fill          |
| <i>value</i> | int value to be set                             |
| <i>num</i>   | number of ints in the block of memory to be set |

##### Returns

No return value

#### 5.14.2.23 AK\_new\_extent()

```
int AK_new_extent (
    int start_address,
    int old_size,
    int extent_type,
    AK_header * header )
```

Function allocates new extent of blocks. If argument "old\_size" is 0 than size of extent is INITIAL\_EXTENT\_SIZE. Otherwise, resize factor is set according to type of extent. If writing of block is successful, number of blocks is incremented.

##### Author

Nikola Bakoš, updated by Dino Laktašić (fixed header BUG), refurbished by dv

##### Parameters

|                      |  |
|----------------------|--|
| <i>start_address</i> | address (block number) to start searching for sufficient space   |
| <i>old_size</i>      | size of previous extent in same segment (in blocks)  |
| <i>extent_type</i>   | type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP) |
| <i>header</i>        | pointer to header that should be written to the new extent (all blocks)  |

**Returns**

address (block number) of new extent if successful, EXIT\_ERROR otherwise

**5.14.2.24 AK\_new\_segment()**

```
int AK_new_segment (
    char * name,
    int type,
    AK_header * header )
```

Function that allocates new segment of extents. In this phase of implementation, only extents containing INITIAL\_EXTENT\_SIZE blocks can be allocated. If extent is successfully allocated, number of allocated extents is incremented and function goes to next block after allocated extent. Otherwise, function moves to INITIAL\_EXTENT\_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK\_free.

**Author**

Tomislav Fotak, refurbished by dv

**Parameters**

|               |  |
|---------------|--|
| <i>name</i>   | (character pointer) name of segment  |
| <i>type</i>   | segment type (possible values: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP) |
| <i>header</i> | (header pointer) pointer to header that should be written to the new extent (all blocks)   |

**Returns**

EXIT\_SUCCESS for success or EXIT\_ERROR if some error occurs

start address for segment because we can not allocate segment in block 0

**5.14.2.25 AK\_print\_block()**

```
int AK_print_block (
    AK_block * block,
    int num,
    char * gg,
    FILE * fpp )
```

Function that dumps block.

**Author**

dv

**Returns**

nothing

#### 5.14.2.26 AK\_read\_block()

```
AK_block* AK_read_block (
    int address )
```

Function that reads a block at a given address (block number less than `db_file_size`). New block is allocated. Database file is opened. Position is set to provided address block. At the end function reads file from that position. Completely thread-safe.

##### Author

Markus Schatten, updated dv and Domagoj Šitum (thread-safe enabled)

##### Parameters

|                |                        |
|----------------|------------------------|
| <i>address</i> | block number (address) |
|----------------|------------------------|

##### Returns

pointer to block allocated in memory

#### 5.14.2.27 AK\_read\_block\_for\_testing()

```
void* AK_read_block_for_testing (
    void * address )
```

This function is only for testing. It has to be there, because `pthread_create` only accepts `void* function_name (void *)` function format. So `AK_read_block` is no-go for `pthread_create`.

##### Author

Domagoj Šitum

#### 5.14.2.28 AK\_register\_system\_tables()

```
int AK_register_system_tables (
    int relation,
    int attribute,
    int index,
    int view,
    int sequence,
    int function,
    int function_arguments,
    int trigger,
    int trigger_conditions,
    int db,
```

```

    int db_obj,
    int user,
    int group,
    int user_group,
    int user_right,
    int group_right,
    int constraint,
    int constraintNull,
    int constraintCheck,
    int constraintUnique,
    int reference )

```

Function that registers system tables. Block at the given address is read. Various data from function arguments are written in block about different database elements.

#### Author

Unknown

#### Parameters

|                           |  |
|---------------------------|--|
| <i>relation</i>           | relation in database                   |
| <i>attribute</i>          | attribute in database                  |
| <i>index</i>              | index in database                      |
| <i>view</i>               | view in database                       |
| <i>sequence</i>           | sequence in database                   |
| <i>function</i>           | function in database                   |
| <i>function_arguments</i> | functional_arguments in database       |
| <i>trigger</i>            | trigger in database                    |
| <i>trigger_conditions</i> | trigger conditions in database         |
| <i>db</i>                 | database                               |
| <i>db_obj</i>             | database object                        |
| <i>user</i>               | user in database                       |
| <i>group</i>              | group in database                      |
| <i>user_group</i>         | user associated with group in database |
| <i>user_right</i>         | user right in database                 |
| <i>group_right</i>        | group right in database                |
| <i>constraint</i>         | constraint in database                 |
| <i>constraintNull</i>     | Null constraint in database            |
| <i>constraintCheck</i>    | Check constraint in database           |
| <i>reference</i>          | reference database                     |

#### Returns

EXIT\_SUCCESS

#### 5.14.2.29 AK\_thread\_safe\_block\_access\_test()

```
void AK_thread_safe_block_access_test ( )
```

This function tests thread safe reading and writing to blocks. There is N writing and N reading threads, which are going through iterations. Each reading thread should read the data (character) that was set by last writing thread.



**Author**

Domagoj Šitum

**5.14.2.30 AK\_write\_block()**

```
int AK_write_block (
    AK_block * block )
```

Function writes a block to DB file. Database file is opened. Position is set to provided address block. Block is written to provided address. Completely thread-safe.

**Author**

Markus Schatten, updated by Domagoj Šitum (thread-safe enabled)

**Parameters**

|              |   |
|--------------|---|
| <i>block</i> | pointer to block allocated in memory to write |
|--------------|---|

**Returns**

EXIT\_SUCCESS if successful, EXIT\_ERROR otherwise

**5.14.2.31 AK\_write\_block\_for\_testing()**

```
void* AK_write_block_for_testing (
    void * block )
```

This function is only for testing. It has to be there, because pthread\_create only accepts void\* function\_name (void \*) function format. So AK\_write\_block is no-go for pthread\_create.

**Author**

Domagoj Šitum

**5.14.2.32 fsize()**

```
int fsize (
    FILE * fp )
```

Helper function to determine file size.

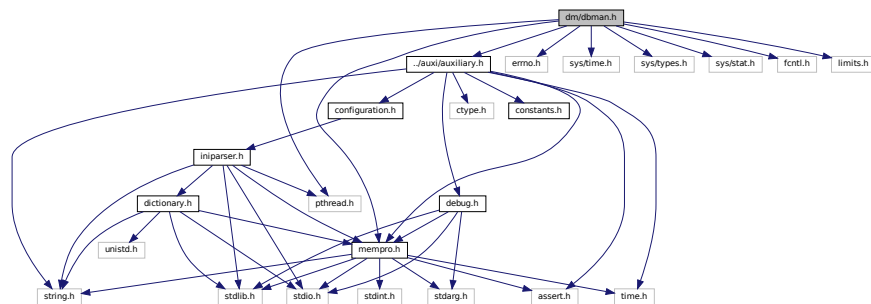
**Returns**

file size

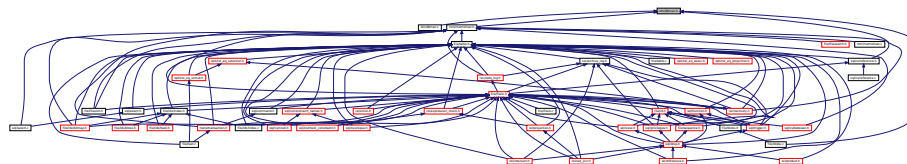
## 5.15 dm/dbman.h File Reference

```
#include "../auxi/auxiliary.h"
#include <errno.h>
#include <pthread.h>
#include "sys/time.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "../auxi/mempro.h"
#include <limits.h>
```

Include dependency graph for dbman.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [AK\\_header](#)  
Structure that represents header structure of blocks (describes an attribute inside an object). It contains type, attribute name, integrity, constraint name and constraint code.
- struct [AK\\_tuple\\_dict](#)  
Structure that defines a mapping in a header of an object to the actual entries (data). It contains type, address and size.
- struct [AK\\_block](#)  
Structure that defines a block of data inside a DB file. It contains address, type, chained\_with, AK\_free space, last\_tuple\_dict\_id, header and tuple\_dict and data.
- struct [table\\_addresses](#)  
Structure that defines start and end address of extent.
- struct [AK\\_blocktable](#)
- struct [AK\\_block\\_activity](#)

Structure which holds information about each block, whether it is locked for reading or writing. It is important to note such information, to enable quick and thread-safe reading from or writing to disk. Structure contains of: `locked_for_reading` - thread which locks particular block for reading will set this value `locked_for_writing` - thread which locks particular block for writing will set this value `block_lock` - each reading and writing operation will be done atomically and uninterruptable, using this mutex `block_lock_reading_done` - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block `block_lock_writing_done` - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it `thread_holding_lock` - the only thread which can unlock locked "block\_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it.

## Macros

- `#define BITMASK(b) (1 << ((b) % CHAR_BIT))`
- `#define BITSLOT(b) ((int)((b) / CHAR_BIT))`
- `#define BITSET(a, b) ((a)[BITSLOT(b)] |= BITMASK(b))`
- `#define BITCLEAR(a, b) ((a)[BITSLOT(b)] &= ~BITMASK(b))`
- `#define BITTEST(a, b) ((a)[BITSLOT(b)] & BITMASK(b))`
- `#define BITNSLOTS(nb) ((int)(nb + CHAR_BIT - 1) / CHAR_BIT)`
- `#define SEGMENTLENGTH() (BITNSLOTS(DB_FILE_BLOCKS_NUM) + 2*sizeof(int))`
- `#define DB_FILE_SIZE_EX 200`
- `#define DB_FILE_BLOCKS_NUM_EX (int)(1024 * 1024 * DB_FILE_SIZE_EX / sizeof(AK_block))`
- `#define AK_ALLOCATION_TABLE_SIZE sizeof(AK_blocktable)`  
*Holds size of allocation table.*
- `#define CHAR_IN_LINE 80`  
*How many characters could line contain.*
- `#define MAX_BLOCK_INIT_NUM MAX_CACHE_MEMORY`  
*How many blocks would be initially allocated.*

## Enumerations

- enum `AK_allocation_set_mode` {  
`allocationSEQUENCE` = 10001, `allocationUPPER`, `allocationLOWER`, `allocationAROUND`,  
`allocationNOMODE` }  
*Different modes to obtain allocation indexes: SEQUENCE - first found set of sequence indexes UPPER - set tries to place itself to upper part of allocation table LOWER - set tries to place itself to lower part of allocation table AROUND - set tries to place itself around targeted index.*

## Functions

- int `AK_print_block` (`AK_block` \*block, int num, char \*gg, FILE \*fpp)  
*Function that dumps block.*
- void `AK_allocationbit_test` ()
- void `AK_allocationtable_test` ()
- int \* `AK_increase_extent` (int start\_address, int add\_size, `AK_allocation_set_mode` \*mode, int border, int target, `AK_header` \*header, int gl)  
*Function allocates new blocks for increasing extent size.*
- int \* `AK_get_extent` (int start\_address, int desired\_size, `AK_allocation_set_mode` \*mode, int border, int target, `AK_header` \*header, int gl)  
*Function allocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.*
- int `AK_get_allocation_set` (int \*bitsetbs, int fromWhere, int gaplength, int num, `AK_allocation_set_mode` mode, int target)  
*Function prepare demanded sets from allocation table.*

- int [AK\\_copy\\_header](#) ([AK\\_header](#) \*header, int \*blocknum, int num)  
*Function copy header to blocks. Completely thread-safe.*
- int [AK\\_allocate\\_blocks](#) (FILE \*db, [AK\\_block](#) \*block, int FromWhere, int HowMany)  
*Function that allocates new blocks by placing them to appropriate place and then update last initialized index.*
- [AK\\_block](#) \* [AK\\_init\\_block](#) ()  
*Function that initializes new block.*
- void [AK\\_allocationtable\\_dump](#) (int zz)  
*Dumps the allocation table from the global allocation bit-vector onto standard output.*
- void [AK\\_blocktable\\_dump](#) (int zz)  
*Dumps the bit-table from the global allocation bit-vector onto standard output.*
- int [AK\\_blocktable\\_flush](#) ()  
*Function flushes bitmask table to disk.*
- void [AK\\_thread\\_safe\\_block\\_access\\_test](#) ()  
*This function tests thread safe reading and writing to blocks. There is N writing and N reading threads, which are going through iterations. Each reading thread should read the data (character) that was set by last writing thread.*
- void \* [AK\\_read\\_block\\_for\\_testing](#) (void \*address)  
*This function is only for testing. It has to be there, because pthread\_create only accepts void\* function\_name (void \*) function format. So AK\_read\_block is no-go for pthread\_create.*
- void \* [AK\\_write\\_block\\_for\\_testing](#) (void \*block)  
*This function is only for testing. It has to be there, because pthread\_create only accepts void\* function\_name (void \*) function format. So AK\_write\_block is no-go for pthread\_create.*
- int [AK\\_blocktable\\_get](#) ()  
*Function gets allocation table from disk.*
- int [fsize](#) (FILE \*fp)  
*Helper function to determine file size.*
- int [AK\\_init\\_allocation\\_table](#) ()  
*Function that initializes allocation table, write it to disk and cache in memory.*
- int [AK\\_init\\_db\\_file](#) (int size)  
*Function that initializes a new database file named DB\_FILE. It opens database file. New block is allocated. In this block type of header is set to FREE\_INT, attribute names are set to FREE\_CHAR, integrities are set to FREE\_INT, constraint names are set to FREE\_CHAR, constraint names and codes are set to FREE\_CHAR. Type, address and size of tuples are set to FREE\_INT. Data in block is set to FREE\_CHAR. Type of block is BLOCK\_TYPE\_FREE, it is not chained and id of last tuple is 0.*
- [AK\\_block](#) \* [AK\\_read\\_block](#) (int address)  
*Function that reads a block at a given address (block number less than db\_file\_size). New block is allocated. Database file is opened. Position is set to provided address block. At the end function reads file from that position. Completely thread-safe.*
- int [AK\\_write\\_block](#) ([AK\\_block](#) \*block)  
*Function writes a block to DB file. Database file is opened. Position is set to provided address block. Block is written to provided address. Completely thread-safe.*
- int [AK\\_new\\_extent](#) (int start\_address, int old\_size, int extent\_type, [AK\\_header](#) \*header)  
*Function allocates new extent of blocks. If argument "old\_size" is 0 than size of extent is INITIAL\_EXTENT\_SIZE. Otherwise, resize factor is set according to type of extent. If writing of block is successful, number of blocks is incremented.*
- int [AK\\_new\\_segment](#) (char \*name, int type, [AK\\_header](#) \*header)  
*Function that allocates new segment of extents. In this phase of implementation, only extents containing INITIAL\_EXTENT\_SIZE blocks can be allocated. If extent is successfully allocated, number of allocated extents is incremented and function goes to next block after allocated extent. Otherwise, function moves to INITIAL\_EXTENT\_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK\_free.*
- [AK\\_header](#) \* [AK\\_create\\_header](#) (char \*name, int type, int integrity, char \*constr\_name, char \*constr\_code)  
*Function for creating header and initialize integrity, constraint name and constraint code with parameter values of function.*
- void [AK\\_insert\\_entry](#) ([AK\\_block](#) \*block\_address, int type, void \*entry\_data, int i)

*Function for inserting entry in tuple\_dict and data of a block. Address, type and size of catalog\_tuple\_dict are set. Free space of block is also set.*

- int [AK\\_init\\_system\\_tables\\_catalog](#) (int relation, int attribute, int index, int view, int sequence, int function, int function\_arguments, int trigger, int trigger\_conditions, int [db](#), int db\_obj, int user, int group, int user\_group, int user\_right, int group\_right, int constraint, int constraintNull, int constraintCheck, int constraintUnique, int reference)

*Function initialises the sytem table catalog and writes the result in first (0) block in db\_file. Catalog block, catalog header name, catalog header address are allocated. Address, type, chained\_with and AK\_free\_space attributes are initialized. Names of various database elements are written in block.*

- void [AK\\_memset\\_int](#) (void \*block, int value, size\_t num)

*Function that sets the first num ints of a block of memory to the specified value.*

- int [AK\\_register\\_system\\_tables](#) (int relation, int attribute, int index, int view, int sequence, int function, int function\_arguments, int trigger, int trigger\_conditions, int [db](#), int db\_obj, int user, int group, int user\_group, int user\_right, int group\_right, int constraint, int constraintNull, int constraintCheck, int constraintUnique, int reference)

*Function that registers system tables. Block at the given address is read. Various data from function arguments are written in block about different database elements.*

- int [AK\\_init\\_system\\_catalog](#) ()

*Function initializes the system catalog. Headers for system tables are defined. Segments for those system tables are allocated. Above function [AK\\_register\\_system\\_tables\(\)](#) to register system tables.*

- int [AK\\_delete\\_block](#) (int address)

*Function deletes a block by a given block address (resets the header and data). Types, integrities, constraint names, constraint codes are set to "AK\_free" values. In tuple dictionary type, address and size are set to FREE\_INT values. Data of block is set to FREE\_CHAR.*

- int [AK\\_delete\\_extent](#) (int begin, int end)

*Function deletes an extent between begin and end blocks.*

- int [AK\\_delete\\_segment](#) (char \*name, int type)
- int [AK\\_init\\_disk\\_manager](#) ()

## Variables

- FILE \* [db](#)

*Variable that defines the DB file file handle.*

- unsigned int [db\\_file\\_size](#)

*Variable that defines the size of the DB file (in blocks)*

- [AK\\_blocktable](#) \* [AK\\_allocationbit](#)

*Global variable that holds allocation bit-vector.*

- [AK\\_block\\_activity](#) \* [AK\\_block\\_activity\\_info](#)
- [AK\\_synchronization\\_info](#) \* [dbmanFileLock](#)

## 5.15.1 Detailed Description

Header file that defines includes and datastructures for the disk manager of Kalashnikov DB

## 5.15.2 Macro Definition Documentation

#### 5.15.2.1 AK\_ALLOCATION\_TABLE\_SIZE

```
#define AK_ALLOCATION_TABLE_SIZE sizeof(AK_blocktable)
```

Holds size of allocation table.

Author

dv

#### 5.15.2.2 CHAR\_IN\_LINE

```
#define CHAR_IN_LINE 80
```

How many characters could line contain.

Author

dv

#### 5.15.2.3 MAX\_BLOCK\_INIT\_NUM

```
#define MAX_BLOCK_INIT_NUM MAX_CACHE_MEMORY
```

How many blocks would be initially allocated.

Author

dv

### 5.15.3 Enumeration Type Documentation

#### 5.15.3.1 AK\_allocation\_set\_mode

```
enum AK_allocation_set_mode
```

Different modes to obtain allocation indexes: SEQUENCE - first found set of sequence indexes UPPER - set tries to place itself to upper part of allocation table LOWER - set tries to place itself to lower part of allocation table AROUND - set tries to place itself around targeted index.

Author

dv

## 5.15.4 Function Documentation

### 5.15.4.1 AK\_allocate\_blocks()

```
int AK_allocate_blocks (
    FILE * db,
    AK_block * block,
    int FromWhere,
    int HowMany )
```

Function that allocates new blocks by placing them to appropriate place and then update last initialized index.

#### Author

Markus Schatten , rearranged by dv

#### Returns

EXIT\_SUCCESS if the file has been written to disk, EXIT\_ERROR otherwise

### 5.15.4.2 AK\_allocationtable\_dump()

```
void AK_allocationtable_dump (
    int verbosity )
```

Dumps the allocation table from the global allocation bit-vector onto standard output.

#### Author

dv

#### Parameters

|                  |   |
|------------------|---|
| <i>verbosity</i> | level of verbosity (1 - minimal, 0 - no output) |
|------------------|---|

### 5.15.4.3 AK\_blocktable\_dump()

```
void AK_blocktable_dump (
    int verbosity )
```

Dumps the bit-table from the global allocation bit-vector onto standard output.

**Author**

dv

**Parameters**

|                  |   |
|------------------|---|
| <i>verbosity</i> | level of verbosity (1 - verbose, 0 - minimal) |
|------------------|---|

**5.15.4.4 AK\_blocktable\_flush()**

```
int AK_blocktable_flush ( )
```

Function flushes bitmask table to disk.

**Author**

dv

**Returns**

EXIT\_SUCCESS if the file has been written to disk, EXIT\_ERROR otherwise

**5.15.4.5 AK\_blocktable\_get()**

```
int AK_blocktable_get ( )
```

Function gets allocation table from disk.

**Author**

dv

**Returns**

EXIT\_SUCCESS if the file has been taken from disk, EXIT\_ERROR otherwise

**5.15.4.6 AK\_copy\_header()**

```
int AK_copy_header (
    AK_header * header,
    int * blockSet,
    int blockSetSize )
```

Function copy header to blocks. Completely thread-safe.

**Author**

Nikola Bakoš, updated by Dino Laktašić (fixed header BUG), refurbished by dv



## Parameters

|                     |  |
|---------------------|--|
| <i>header</i>       | Pointer to header which will be copied into each block in blockSet |
| <i>blockSet</i>     | Pointer to array of block addresses into which to copy header      |
| <i>blockSetSize</i> | Number of blocks in blockSet                                       |

## Returns

number of performed header copy

## 5.15.4.7 AK\_create\_header()

```
AK_header* AK_create_header (
    char * attribute_name,
    int type,
    int integrity,
    char * constr_name,
    char * contr_code )
```

Function for creating header and initialize integrity, constraint name and constraint code with parameter values of function.

## Author

Matija Novak

## Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>name</i>        | name of the attribute           |
| <i>type</i>        | type of the attribute           |
| <i>integrity</i>   | standard integrity constraint   |
| <i>constr_name</i> | extra integrity constraint name |
| <i>contr_code</i>  | extra integrity constraint code |

## Returns

[AK\\_header](#)

## 5.15.4.8 AK\_delete\_block()

```
int AK_delete_block (
    int address )
```

Function deletes a block by a given block address (resets the header and data). Types, integrities, constraint names, constraint codes are set to "AK\_free" values. In tuple dictionary type, address and size are set to FREE\_INT values. Data of block is set to FREE\_CHAR.

**Author**

Markus Schatten

**Parameters**

|                |                                    |
|----------------|------------------------------------|
| <i>address</i> | address of the block to be deleted |
|----------------|------------------------------------|

**Returns**

returns EXIT\_SUCCESS if deletion successful, else EXIT\_ERROR

**5.15.4.9 AK\_delete\_extent()**

```
int AK_delete_extent (
    int begin,
    int end )
```

Function deletes an extent between begin and end blocks.

**Author**

Dejan Samboliæ

**Parameters**

|              |                                 |
|--------------|---------------------------------|
| <i>begin</i> | address of extent's first block |
| <i>end</i>   | address of extent's last block  |

**Returns**

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

**5.15.4.10 AK\_delete\_segment()**

```
int AK_delete_segment (
    char * name,
    int type )
```

**Author**

Mislav Ėakarizæ

## Parameters

|             |                     |
|-------------|---------------------|
| <i>name</i> | name of the segment |
| <i>type</i> | type of the segment |

## Returns

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

## 5.15.4.11 AK\_get\_allocation\_set()

```
int AK_get_allocation_set (
    int * allocationSet,
    int fromWhere,
    int gaplength,
    int numRequestedBlocks,
    AK_allocation_set_mode mode,
    int target )
```

Function prepare demanded sets from allocation table.

## Author

dv

## Parameters

|                           |  |
|---------------------------|--|
| <i>allocationSet</i>      | Pointer to array which will be filled and represent the allocation set   |
| <i>fromWhere</i>          | Has meaning only if mode is SEQUENCE. It describes from which address searching starts.                              |
| <i>gaplength</i>          | Tells how many used blocks can be tolerated in allocation set  |
| <i>numRequestedBlocks</i> | Tells how many AK_free blocks have been requested  |
| <i>mode</i>               | Defines how to obtain set of indexes to AK_free addresses  |
| <i>target</i>             | Has meaning just if mode is AROUND: set will be as close as possible to the requested target address from both sides |

## Returns

the first element of the allocation set

## 5.15.4.12 AK\_get\_extent()

```
int* AK_get_extent (
    int start_address,
```

```

    int desired_size,
    AK_allocation_set_mode * mode,
    int border,
    int target,
    AK_header * header,
    int gl )

```

Function allocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.

#### Author

dv

#### Parameters

|                               |  |
|-------------------------------|--|
| <i>start_address</i>          | address (block number) to start searching for sufficient space   |
| <i>desired_size</i>           | number of desired blocks   |
| <i>AK_allocation_set_mode</i> | a way of trying to find AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND |
| <i>border</i>                 | number of allocated blocks gap   |
| <i>target</i>                 | block address around which other blocks have to be searched  |
| <i>header</i>                 | pointer to header that should be written to the new extent (all blocks)  |
| <i>int</i>                    | gl gap size  |

#### Returns

pointer to set of allocated block addresses

vars for loop [for]

if some blocks are not successfully allocated, which means that the extend allocation has FAILED

#### 5.15.4.13 AK\_increase\_extent()

```

int* AK_increase_extent (
    int start_address,
    int add_size,
    AK_allocation_set_mode * mode,
    int border,
    int target,
    AK_header * header,
    int gl )

```

Function allocates new blocks for increasing extent size.

#### Author

dv

#### Parameters

|                      |   |
|----------------------|---|
| <i>start_address</i> | first address of extent that is subject of increasing |
|----------------------|---|

## Parameters

|                               |  |
|-------------------------------|--|
| <i>add_size</i>               | number how many new blocks is to be added to existing extent   |
| <i>AK_allocation_set_mode</i> | a way of trying to find AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND |
| <i>border</i>                 | number of allocated blocks gap   |
| <i>target</i>                 | block address around which other blocks have to be searched  |
| <i>header</i>                 | pointer to header that should be written to the new extent (all blocks)  |
| <i>int</i>                    | gl gap size  |

## Returns

pointer to set of allocated block addresses

## 5.15.4.14 AK\_init\_allocation\_table()

```
int AK_init_allocation_table ( )
```

Function that initializes allocation table, write it to disk and cache in memory.

## Author

dv

## Returns

EXIT\_SUCCESS if the file has been written to disk, EXIT\_ERROR otherwise

## 5.15.4.15 AK\_init\_block()

```
AK_block* AK_init_block ( )
```

Function that initializes new block.

## Author

Markus Schatten , rearranged by dv

## Returns

pointer to block allocated in memory

#### 5.15.4.16 AK\_init\_db\_file()

```
int AK_init_db_file (
    int size )
```

Function that initializes a new database file named DB\_FILE. It opens database file. New block is allocated. In this block type of header is set to FREE\_INT, attribute names are set to FREE\_CHAR, integrities are set to FREE\_INT, constraint names are set to FREE\_CHAR, constraint names and codes are set to FREE\_CHAR. Type, address and size of tuples are set to FREE\_INT. Data in block is set to FREE\_CHAR. Type of block is BLOCK\_TYPE\_FREE, it is not chained and id of last tuple is 0.

##### Author

Markus Schatten

##### Parameters

|      |                               |
|------|-------------------------------|
| size | size of new file in in blocks |
|------|-------------------------------|

##### Returns

EXIT\_SUCCESS if the file has been written to disk, EXIT\_ERROR otherwise

#### 5.15.4.17 AK\_init\_disk\_manager()

```
int AK_init_disk_manager ( )
```

##### Author

Markus Schatten

##### Returns

Function that calls functions [AK\\_init\\_db\\_file\(\)](#) and [AK\\_init\\_system\\_catalog\(\)](#) to initialize disk manager. It also calls [AK\\_allocate\\_array\\_currently\\_accessed\\_blocks\(\)](#) to allocate memory needed for thread-safe reading and writing to disk.

#### 5.15.4.18 AK\_init\_system\_catalog()

```
int AK_init_system_catalog ( )
```

Function initializes the system catalog. Headers for system tables are defined. Segments for those system tables are allocated. Above function [AK\\_register\\_system\\_tables\(\)](#) to register system tables.

##### Author

Miroslav Policki

##### Returns

EXIT\_SUCCESS if the system catalog has been successfully initialized, EXIT\_ERROR otherwise

## 5.15.4.19 AK\_init\_system\_tables\_catalog()

```

int AK_init_system_tables_catalog (
    int relation,
    int attribute,
    int index,
    int view,
    int sequence,
    int function,
    int function_arguments,
    int trigger,
    int trigger_conditions,
    int db,
    int db_obj,
    int user,
    int group,
    int user_group,
    int user_right,
    int group_right,
    int constraint,
    int constraintNull,
    int constraintCheck,
    int constraintUnique,
    int reference )

```

Function initialises the sytem table catalog and writes the result in first (0) block in db\_file. Catalog block, catalog header name, catalog header address are allocated. Address, type, chained\_with and AK\_free\_space attributes are initialized. Names of various database elements are written in block.

## Author

Matija Novak

## Parameters

|                           |  |
|---------------------------|--|
| <i>relation</i>           | address of system table of relation in db_file                     |
| <i>attribute</i>          | address of system table of attribute in db_file                    |
| <i>index</i>              | address of system table of index in db_file                        |
| <i>view</i>               | address of system table of view in db_file                         |
| <i>sequence</i>           | address of system table of sequence in db_file                     |
| <i>function</i>           | address of system table of function in db_file                     |
| <i>function_arguments</i> | address of system table of function_arguments in db_file           |
| <i>trigger</i>            | address of system table of trigger in db_file                      |
| <i>trigger_conditions</i> | address of system table of trigger_conditions in db_file           |
| <i>db</i>                 | address of system table of db in db_file                           |
| <i>db_obj</i>             | address of system table of db_obj in db_file                       |
| <i>user</i>               | address of system table of user in db_file                         |
| <i>group</i>              | address of system table of group in db_file                        |
| <i>user_group</i>         | address of system table of users associated with groups in db_file |
| <i>user_right</i>         | address of system table of user right in db_file                   |
| <i>group_right</i>        | address of system table of group right in db_file                  |
| <i>constraint</i>         | address of system table of constraint in db_file                   |
| <i>constraintNull</i>     | address of system table of constraintNull in db_file               |
| <i>constraintCheck</i>    | system table address for check constraint                          |
| <i>reference</i>          | address of system table of reference in db_file                    |

**Returns**

EXIT\_SUCCESS if initialization was succesful if not returns EXIT\_ERROR

first header attribute of catalog\_block

second attribute of catalog\_block

initialize other elements of block (adress, type, chained\_with, AK\_free\_space)

using as an address for the first AK\_free space in block->data

merge catalog\_heder with heders created before

**5.15.4.20 AK\_insert\_entry()**

```
void AK_insert_entry (
    AK_block * block_address,
    int type,
    void * entry_data,
    int i )
```

Function for inserting entry in tuple\_dict and data of a block. Address, type and size of catalog\_tuple\_dict are set. Free space of block is also set.

**Author**

Matija Novak

**Parameters**

|                     |   |
|---------------------|---|
| <i>block_adress</i> | adress of a block in which we want insert data                                |
| <i>type</i>         | type of entry_data  |
| <i>entry_data</i>   | (char) data which is inserted, can be int but must first be converted to char |
| <i>i</i>            | (int) adress in tuple_dict array (example block_address->tuple_dict[i])       |

**Returns**

No return value because it gets the address of an block like a function parameter and works directly with the original block

copy data into bloc->data on start position bloc->AK\_free\_space

address of entry data in block->data

calculate next AK\_free space for the next entry data

```
sizeof(entry_data)+1); /// (sizeof(int));
```

no need for "+strlen(entry\_data)" while "+1" is like "new line"

type of entry data

size of entry data

copy tuple\_dict to block->tuple\_dict[i] must use & becouse tuple\_dict[i] is value and catalog\_tuple\_dict adress



#### 5.15.4.21 AK\_memset\_int()

```
void AK_memset_int (
    void * block,
    int value,
    size_t num )
```

Function that sets the first num ints of a block of memory to the specified value.

##### Author

Miroslav Policki

##### Parameters

|              |   |
|--------------|---|
| <i>block</i> | pointer to the block of memory to fill          |
| <i>value</i> | int value to be set                             |
| <i>num</i>   | number of ints in the block of memory to be set |

##### Returns

No return value

#### 5.15.4.22 AK\_new\_extent()

```
int AK_new_extent (
    int start_address,
    int old_size,
    int extent_type,
    AK_header * header )
```

Function allocates new extent of blocks. If argument "old\_size" is 0 than size of extent is INITIAL\_EXTENT\_SIZE. Otherwise, resize factor is set according to type of extent. If writing of block is successful, number of blocks is incremented.

##### Author

Nikola Bakoš, updated by Dino Laktašić (fixed header BUG), refurbished by dv

##### Parameters

|                      |  |
|----------------------|--|
| <i>start_address</i> | address (block number) to start searching for sufficient space   |
| <i>old_size</i>      | size of previous extent in same segment (in blocks)  |
| <i>extent_type</i>   | type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP) |
| <i>header</i>        | pointer to header that should be written to the new extent (all blocks)  |

**Returns**

address (block number) of new extent if successful, EXIT\_ERROR otherwise

**5.15.4.23 AK\_new\_segment()**

```
int AK_new_segment (
    char * name,
    int type,
    AK_header * header )
```

Function that allocates new segment of extents. In this phase of implementation, only extents containing INITIAL\_EXTENT\_SIZE blocks can be allocated. If extent is successfully allocated, number of allocated extents is incremented and function goes to next block after allocated extent. Otherwise, function moves to INITIAL\_EXTENT\_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK\_free.

**Author**

Tomislav Fotak, refurbished by dv

**Parameters**

|               |  |
|---------------|--|
| <i>name</i>   | (character pointer) name of segment  |
| <i>type</i>   | segment type (possible values: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP) |
| <i>header</i> | (header pointer) pointer to header that should be written to the new extent (all blocks)   |

**Returns**

EXIT\_SUCCESS for success or EXIT\_ERROR if some error occurs

start address for segment because we can not allocate segment in block 0

**5.15.4.24 AK\_print\_block()**

```
int AK_print_block (
    AK_block * block,
    int num,
    char * gg,
    FILE * fpp )
```

Function that dumps block.

**Author**

dv

**Returns**

nothing

#### 5.15.4.25 AK\_read\_block()

```
AK_block* AK_read_block (
    int address )
```

Function that reads a block at a given address (block number less than `db_file_size`). New block is allocated. Database file is opened. Position is set to provided address block. At the end function reads file from that position. Completely thread-safe.

##### Author

Markus Schatten, updated dv and Domagoj Šitum (thread-safe enabled)

##### Parameters

|                |                        |
|----------------|------------------------|
| <i>address</i> | block number (address) |
|----------------|------------------------|

##### Returns

pointer to block allocated in memory

#### 5.15.4.26 AK\_read\_block\_for\_testing()

```
void* AK_read_block_for_testing (
    void * address )
```

This function is only for testing. It has to be there, because `pthread_create` only accepts `void* function_name (void *)` function format. So `AK_read_block` is no-go for `pthread_create`.

##### Author

Domagoj Šitum

#### 5.15.4.27 AK\_register\_system\_tables()

```
int AK_register_system_tables (
    int relation,
    int attribute,
    int index,
    int view,
    int sequence,
    int function,
    int function_arguments,
    int trigger,
    int trigger_conditions,
    int db,
```

```

    int db_obj,
    int user,
    int group,
    int user_group,
    int user_right,
    int group_right,
    int constraint,
    int constraintNull,
    int constraintCheck,
    int constraintUnique,
    int reference )

```

Function that registers system tables. Block at the given address is read. Various data from function arguments are written in block about different database elements.

#### Author

Unknown

#### Parameters

|                           |  |
|---------------------------|--|
| <i>relation</i>           | relation in database                   |
| <i>attribute</i>          | attribute in database                  |
| <i>index</i>              | index in database                      |
| <i>view</i>               | view in database                       |
| <i>sequence</i>           | sequence in database                   |
| <i>function</i>           | function in database                   |
| <i>function_arguments</i> | functional_arguments in database       |
| <i>trigger</i>            | trigger in database                    |
| <i>trigger_conditions</i> | trigger conditions in database         |
| <i>db</i>                 | database                               |
| <i>db_obj</i>             | database object                        |
| <i>user</i>               | user in database                       |
| <i>group</i>              | group in database                      |
| <i>user_group</i>         | user associated with group in database |
| <i>user_right</i>         | user right in database                 |
| <i>group_right</i>        | group right in database                |
| <i>constraint</i>         | constraint in database                 |
| <i>constraintNull</i>     | Null constraint in database            |
| <i>constraintCheck</i>    | Check constraint in database           |
| <i>reference</i>          | reference database                     |

#### Returns

EXIT\_SUCCESS

#### 5.15.4.28 AK\_thread\_safe\_block\_access\_test()

```
void AK_thread_safe_block_access_test ( )
```

This function tests thread safe reading and writing to blocks. There is N writing and N reading threads, which are going through iterations. Each reading thread should read the data (character) that was set by last writing thread.

**Author**

Domagoj Šitum

**5.15.4.29 AK\_write\_block()**

```
int AK_write_block (
    AK_block * block )
```

Function writes a block to DB file. Database file is opened. Position is set to provided address block. Block is written to provided address. Completely thread-safe.

**Author**

Markus Schatten, updated by Domagoj Šitum (thread-safe enabled)

**Parameters**

|              |   |
|--------------|---|
| <i>block</i> | pointer to block allocated in memory to write |
|--------------|---|

**Returns**

EXIT\_SUCCESS if successful, EXIT\_ERROR otherwise

**5.15.4.30 AK\_write\_block\_for\_testing()**

```
void* AK_write_block_for_testing (
    void * block )
```

This function is only for testing. It has to be there, because pthread\_create only accepts void\* function\_name (void \*) function format. So AK\_write\_block is no-go for pthread\_create.

**Author**

Domagoj Šitum

**5.15.4.31 fsize()**

```
int fsize (
    FILE * fp )
```

Helper function to determine file size.

**Returns**

file size

### 5.15.5 Variable Documentation

#### 5.15.5.1 AK\_allocationbit

`AK_allocationbit`

Global variable that holds allocation bit-vector.

**Author**

dv

#### 5.15.5.2 db

`db`

Variable that defines the DB file file handle.

**Author**

Markus Schatten

#### 5.15.5.3 db\_file\_size

`db_file_size`

Variable that defines the size of the DB file (in blocks)

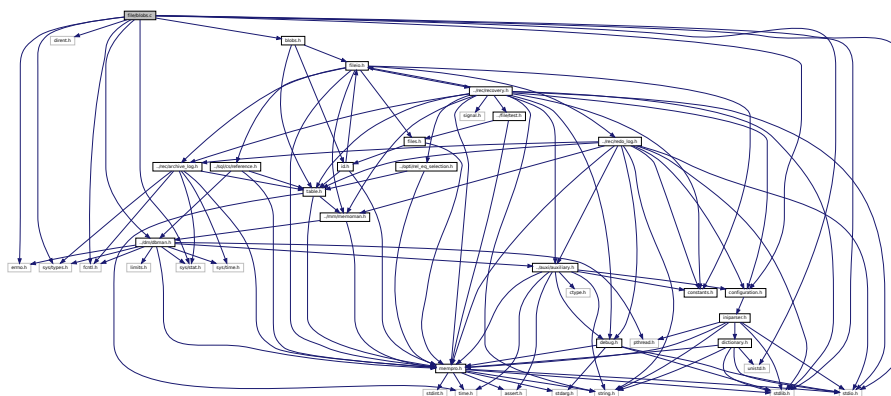
**Author**

Markus Schatten

## 5.16 file/blobs.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include "../dm/dbman.h"
#include "../auxi/configuration.h"
#include "blobs.h"
```

Include dependency graph for blobs.c:



### Functions

- [AK\\_File\\_Metadata](#) [AK\\_File\\_Metadata\\_malloc](#) ()
- char \* [AK\\_GUID](#) ()  
*Function for generating GUID.*
- int [AK\\_folder\\_exists](#) (char \*foldername)  
*Function for checking if folder blobs already exists.*
- int [AK\\_mkdir](#) (const char \*path)  
*Function for creating new folder.*
- int [AK\\_copy](#) (const char \*from, const char \*to)
- char \* [AK\\_concat](#) (char \*s1, char \*s2)  
*Function for AK\_concatinating 2 strings.*
- char \* [AK\\_clear\\_all\\_newline](#) (char \*s)
- int [AK\\_check\\_folder\\_blobs](#) ()  
*Function for checking if folder blobs exists.*
- void [AK\\_split\\_path\\_file](#) (char \*\*p, char \*\*f, char \*pf)  
*Function for splitting path from filename.*
- int [AK\\_write\\_metadata](#) (char \*oid, [AK\\_File\\_Metadata](#) meta)
- [AK\\_File\\_Metadata](#) [AK\\_read\\_metadata](#) (char \*oid)
- char \* [AK\\_lo\\_import](#) (char \*filepath)  
*Function for importing large objects to database.*
- int [AK\\_lo\\_export](#) (char \*oid, char \*filepath)  
*Function for retrieving large objects.*

- int [AK\\_lo\\_unlink](#) (char \*oid)  
*Function for deleting large objects.*
- void [AK\\_lo\\_test](#) ()  
*Tests.*

### 5.16.1 Detailed Description

Provides functions for manipulations of binary large objects

### 5.16.2 Function Documentation

#### 5.16.2.1 [AK\\_check\\_folder\\_blobs\(\)](#)

```
int AK_check_folder_blobs ( )
```

Function for checking if folder blobs exists.

##### Author

Samuel Picek

##### Returns

OID (object ID)

#### 5.16.2.2 [AK\\_concat\(\)](#)

```
char* AK_concat (
    char * s1,
    char * s2 )
```

Function for AK\_concatinating 2 strings.

##### Author

Samuel Picek

##### Returns

returns new string



### 5.16.2.3 AK\_folder\_exists()

```
int AK_folder_exists (
    char * foldername )
```

Function for checking if folder blobs already exists.

#### Author

Samuel Picek

#### Returns

returns 0 for true and 1 for false

### 5.16.2.4 AK\_GUID()

```
char* AK_GUID ( )
```

Function for generating GUID.

#### Author

Samuel Picek

#### Returns

returns globally universal identifier based on kernel implementation

### 5.16.2.5 AK\_lo\_export()

```
int AK_lo_export (
    char * oid,
    char * filepath )
```

Function for retrieving large objects.

#### Author

Samuel Picek

#### Returns

returns 0 for true and 1 for false

#### 5.16.2.6 AK\_lo\_import()

```
char* AK_lo_import (
    char * filepath )
```

Function for importing large objects to database.

##### Author

Samuel Picek

##### Returns

OID (object ID)

#### 5.16.2.7 AK\_lo\_test()

```
void AK_lo_test ( )
```

Tests.

##### Author

Samuel Picek

#### 5.16.2.8 AK\_lo\_unlink()

```
int AK_lo_unlink (
    char * oid )
```

Function for deleting large objects.

##### Author

Samuel Picek

##### Returns

OID (object ID)

#### 5.16.2.9 AK\_mkdir()

```
int AK_mkdir (
    const char * path )
```

Function for creating new folder.

**Author**

Samuel Picek

## Returns

returns 0 for true and 1 for false

#### 5.16.2.10 AK\_split\_path\_file()

```
void AK_split_path_file (
    char ** p,
    char ** f,
    char * pf )
```

Function for splitting path from filename.

**Author**

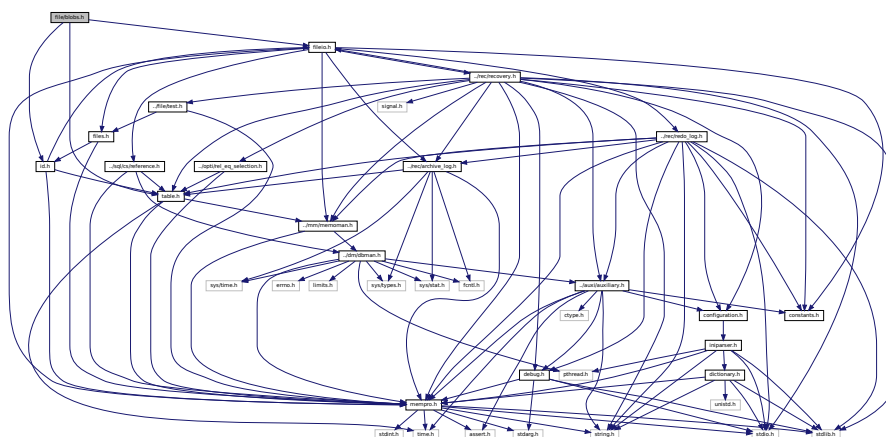
Samuel Picek

## Returns

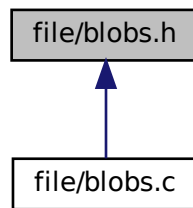
void

## 5.17 file/blobs.h File Reference

```
#include "table.h"
#include "fileio.h"
#include "id.h"
Include dependency graph for blobs.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [\\_file\\_metadata](#)

## Typedefs

- typedef struct [\\_file\\_metadata](#) **AK\_Metadata**
- typedef struct [\\_file\\_metadata](#) \* **AK\_File\_Metadata**

## Functions

- [AK\\_File\\_Metadata](#) **AK\_File\_Metadata\_malloc** ()
- int [AK\\_mkdir](#) (const char \*path)  
*Function for creating new folder.*
- int **AK\_copy** (const char \*from, const char \*to)
- char \* [AK\\_concat](#) (char \*s1, char \*s2)  
*Function for AK\_concatinating 2 strings.*
- char \* **AK\_clear\_all\_newline** (char \*str)
- void [AK\\_split\\_path\\_file](#) (char \*\*p, char \*\*f, char \*pf)  
*Function for spliting path from filename.*
- char \* [AK\\_GUID](#) ()  
*Function for generating GUID.*
- int [AK\\_folder\\_exists](#) (char \*foldername)  
*Function for checking if folder blobs already exists.*
- int [AK\\_check\\_folder\\_blobs](#) ()  
*Function for checking if folder blobs exists.*
- int **AK\_write\_metadata** (char \*oid, [AK\\_File\\_Metadata](#) meta)
- [AK\\_File\\_Metadata](#) **AK\_read\_metadata** (char \*oid)
- char \* [AK\\_lo\\_import](#) (char \*filepath)  
*Function for importing large objects to database.*
- int [AK\\_lo\\_export](#) (char \*oid, char \*filepath)  
*Function for retrieving large objects.*
- int [AK\\_lo\\_unlink](#) (char \*oid)  
*Function for deleting large objects.*
- void [AK\\_lo\\_test](#) ()  
*Tests.*

### 5.17.1 Detailed Description

Provides data structures for manipulating blobs

### 5.17.2 Function Documentation

#### 5.17.2.1 AK\_check\_folder\_blobs()

```
int AK_check_folder_blobs ( )
```

Function for checking if folder blobs exists.

##### Author

Samuel Picek

##### Returns

OID (object ID)

#### 5.17.2.2 AK\_concat()

```
char* AK_concat (
    char * s1,
    char * s2 )
```

Function for AK\_concatinating 2 strings.

##### Author

Samuel Picek

##### Returns

returns new string

### 5.17.2.3 AK\_folder\_exists()

```
int AK_folder_exists (
    char * foldername )
```

Function for checking if folder blobs already exists.

#### Author

Samuel Picek

#### Returns

returns 0 for true and 1 for false

### 5.17.2.4 AK\_GUID()

```
char* AK_GUID ( )
```

Function for generating GUID.

#### Author

Samuel Picek

#### Returns

returns globally universal identifier based on kernel implementation

### 5.17.2.5 AK\_lo\_export()

```
int AK_lo_export (
    char * oid,
    char * filepath )
```

Function for retrieving large objects.

#### Author

Samuel Picek

#### Returns

returns 0 for true and 1 for false

#### 5.17.2.6 AK\_lo\_import()

```
char* AK_lo_import (
    char * filepath )
```

Function for importing large objects to database.

##### Author

Samuel Picek

##### Returns

OID (object ID)

#### 5.17.2.7 AK\_lo\_test()

```
void AK_lo_test ( )
```

Tests.

##### Author

Samuel Picek

#### 5.17.2.8 AK\_lo\_unlink()

```
int AK_lo_unlink (
    char * oid )
```

Function for deleting large objects.

##### Author

Samuel Picek

##### Returns

OID (object ID)





## Functions

- void [Ak\\_Insert\\_New\\_Element\\_For\\_Update](#) (int newtype, void \*data, char \*table, char \*attribute\_name, struct [list\\_node](#) \*ElementBefore, int newconstraint)  
*Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.*
- void [Ak\\_Insert\\_New\\_Element](#) (int newtype, void \*data, char \*table, char \*attribute\_name, struct [list\\_node](#) \*ElementBefore)  
*Function inserts new element after some element, to insert on first place give list as before element. It calls function [Ak\\_Insert\\_New\\_Element\\_For\\_Update](#).*
- int [Ak\\_insert\\_row\\_to\\_block](#) (struct [list\\_node](#) \*row\_root, [AK\\_block](#) \*temp\_block)  
*Function inserts one row into some block. Firstly it checks wether block contain attributes from the list. Then data, type, size and last\_tuple\_id are put in temp\_block.*
- int [Ak\\_insert\\_row](#) (struct [list\\_node](#) \*row\_root)  
*Function inserts a one row into table. Firstly it is checked whether inserted row would violite reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_DIRTY.*
- void [Ak\\_update\\_row\\_from\\_block](#) ([AK\\_block](#) \*temp\_block, struct [list\\_node](#) \*row\_root)  
*Function updates row from table in given block.*
- void [Ak\\_delete\\_row\\_from\\_block](#) ([AK\\_block](#) \*temp\_block, struct [list\\_node](#) \*row\_root)  
*Function deletes row from table in given block. Given list of elements is firstly back-upped.*
- int [Ak\\_delete\\_update\\_segment](#) (struct [list\\_node](#) \*row\_root, int del)  
*Function updates or deletes the whole segment of an table. Addresses for given table atr fetched. For each block in extent row is updated or deleted according to operator del.*
- int [Ak\\_delete\\_row](#) (struct [list\\_node](#) \*row\_root)  
*Function deletes rows.*
- void [Ak\\_delete\\_row\\_by\\_id](#) (int id, char \*tableName)  
*Function deletes row by id.*
- int [Ak\\_update\\_row](#) (struct [list\\_node](#) \*row\_root)  
*Function updates rows of some table.*
- void [Ak\\_fileio\\_test](#) ()

### 5.18.1 Detailed Description

Provides functions for file input/output

### 5.18.2 Function Documentation

#### 5.18.2.1 [Ak\\_delete\\_row\(\)](#)

```
int Ak_delete_row (
    struct list\_node * row_root )
```

Function deletes rows.

#### Author

Matija Novak, Dejan Frankovic (added referential integrity)

**Parameters**

|                 |   |
|-----------------|---|
| <i>row_root</i> | elements of one row EXIT_SUCCESS if success |
|-----------------|---|

**5.18.2.2 Ak\_delete\_row\_by\_id()**

```
void Ak_delete_row_by_id (
    int id,
    char * tableName )
```

Function deletes row by id.

**Author**

Dražen Bandić

**Parameters**

|                  |                                 |
|------------------|---------------------------------|
| <i>id</i>        | id of row                       |
| <i>tableName</i> | name of table to delete the row |

**5.18.2.3 Ak\_delete\_row\_from\_block()**

```
void Ak_delete_row_from_block (
    AK_block * temp_block,
    struct list_node * row_root )
```

Function deletes row from table in given block. Given list of elements is firstly back-upped.

**Author**

Matija Novak, updated by Dino Laktašić, changed by Davorin Vukelic, updated by Mario Peroković

**Parameters**

|                   |  |
|-------------------|--|
| <i>temp_block</i> | block to work with                                       |
| <i>row_list</i>   | list of elements which contain data for delete or update |

**Returns**

No return value

#### 5.18.2.4 Ak\_delete\_update\_segment()

```
int Ak_delete_update_segment (
    struct list_node * row_root,
    int del )
```

Function updates or deletes the whole segment of an table. Addresses for given table atr fetched. For each block in extent row is updated or deleted according to operator del.

##### Author

Matija Novak, updated by Matija Šestak (function now uses caching)

##### Parameters

|                 |                     |
|-----------------|---------------------|
| <i>row_root</i> | elements of one row |
| <i>del</i>      | - DELETE or UPDATE  |

##### Returns

EXIT\_SUCCESS if success

#### 5.18.2.5 Ak\_Insert\_New\_Element()

```
void Ak_Insert_New_Element (
    int newtype,
    void * data,
    char * table,
    char * attribute_name,
    struct list_node * ElementBefore )
```

Function inserts new element after some element, to insert on first place give list as before element. It calls function Ak\_Insert\_New\_Element\_For\_Update.

##### Author

Matija Novak, changed by Dino Laktašić

##### Parameters

|                       |   |
|-----------------------|---|
| <i>newtype</i>        | type of the data                              |
| <i>data</i>           | the data                                      |
| <i>table</i>          | table name                                    |
| <i>attribute_name</i> | attribute name                                |
| <i>element</i>        | element after we which insert the new element |
| <i>constraint</i>     | is NEW_VALUE                                  |

**Returns**

No return value

**5.18.2.6 Ak\_Insert\_New\_Element\_For\_Update()**

```
void Ak_Insert_New_Element_For_Update (
    int newtype,
    void * data,
    char * table,
    char * attribute_name,
    struct list_node * ElementBefore,
    int newconstraint )
```

Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.

**Author**

Matija Novak

**Parameters**

|                       |   |
|-----------------------|---|
| <i>newtype</i>        | type of the data  |
| <i>data</i>           | the data  |
| <i>table</i>          | table name  |
| <i>attribute_name</i> | attribute name  |
| <i>element</i>        | element after we which insert the new element   |
| <i>constraint</i>     | NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for |

**Returns**

No return value

**5.18.2.7 Ak\_insert\_row()**

```
int Ak_insert_row (
    struct list_node * row_root )
```

Function inserts a one row into table. Firstly it is checked whether inserted row would violite reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_↔ DIRTY.

**Author**

Matija Novak, updated by Matija Šestak (function now uses caching), updated by Dejan Frankovic (added reference check), updated by Dino Laktašić (removed variable AK\_free, variable table initialized using memset)

**Parameters**

|                 |  |
|-----------------|--|
| <i>row_root</i> | list of elements which contain data of one row |
|-----------------|--|

**Returns**

EXIT\_SUCCESS if success else EXIT\_ERROR

**5.18.2.8 Ak\_insert\_row\_to\_block()**

```
int Ak_insert_row_to_block (
    struct list_node * row_root,
    AK_block * temp_block )
```

Function inserts one row into some block. Firstly it checks wether block contain attributes from the list. Then data, type, size and last\_tuple\_id are put in temp\_block.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

|                   |                               |
|-------------------|-------------------------------|
| <i>row_root</i>   | list of elements to insert    |
| <i>temp_block</i> | block in which we insert data |

**Returns**

EXIT SUCCES if success

**5.18.2.9 Ak\_update\_row()**

```
int Ak_update_row (
    struct list_node * row_root )
```

Function updates rows of some table.

**Author**

Matija Novak, Dejan Frankovic (added referential integrity)

**Parameters**

|                 |                     |
|-----------------|---------------------|
| <i>row_root</i> | elements of one row |
|-----------------|---------------------|

**Returns**

EXIT\_SUCCESS if success

**5.18.2.10 Ak\_update\_row\_from\_block()**

```
void Ak_update_row_from_block (
    AK_block * temp_block,
    struct list_node * row_root )
```

Function updates row from table in given block.

**Author**

Matija Novak, updated by Dino Laktašić, updated by Mario Peroković - separated from deletion

**Parameters**

|                   |  |
|-------------------|--|
| <i>temp_block</i> | block to work with                                       |
| <i>row_list</i>   | list of elements which contain data for delete or update |

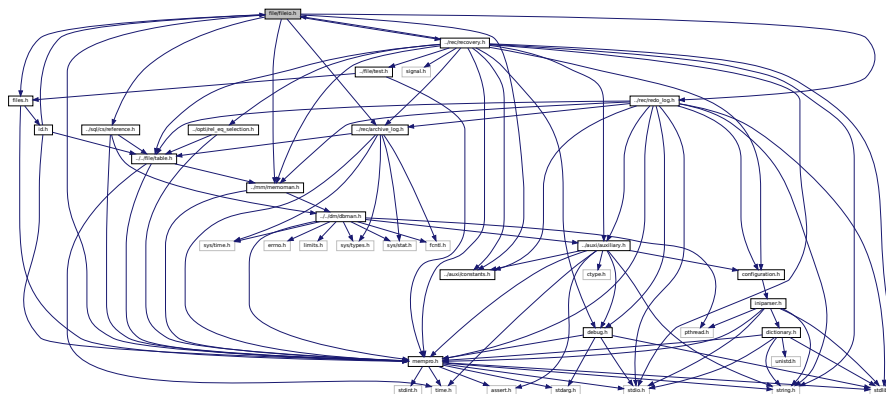
**Returns**

No return value

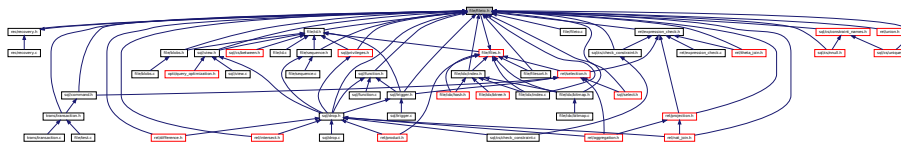
**5.19 file/fileio.h File Reference**

```
#include "../auxi/constants.h"
#include "../sql/cs/reference.h"
#include "../mm/memoman.h"
#include "../rec/recovery.h"
#include "../rec/archive_log.h"
#include "../rec/redo_log.h"
#include "files.h"
#include "../auxi/mempro.h"
```

Include dependency graph for fileio.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [Ak\\_Insert\\_New\\_Element\\_For\\_Update](#) (int newtype, void \*data, char \*table, char \*attribute\_name, struct [list\\_node](#) \*ElementBefore, int newconstraint)  
*Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elemets are set according to function arguments. Pointers are changed so that before element points to new element.*
- void [Ak\\_Insert\\_New\\_Element](#) (int newtype, void \*data, char \*table, char \*attribute\_name, struct [list\\_node](#) \*ElementBefore)  
*Function inserts new element after some element, to insert on first place give list as before element. It calls function [Ak\\_Insert\\_New\\_Element\\_For\\_Update](#).*
- int [Ak\\_insert\\_row\\_to\\_block](#) (struct [list\\_node](#) \*row\_root, [AK\\_block](#) \*temp\_block)  
*Function inserts one row into some block. Firstly it checks wether block contain attributes from the list. Then data, type, size and last\_tuple\_id are put in temp\_block.*
- int [Ak\\_insert\\_row](#) (struct [list\\_node](#) \*row\_root)  
*Function inserts a one row into table. Firstly it is checked whether inserted row would violite reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_DIRTY.*
- void [Ak\\_update\\_row\\_from\\_block](#) ([AK\\_block](#) \*temp\_block, struct [list\\_node](#) \*row\_root)  
*Function updates row from table in given block.*
- void [Ak\\_delete\\_row\\_from\\_block](#) ([AK\\_block](#) \*temp\_block, struct [list\\_node](#) \*row\_root)  
*Function deletes row from table in given block. Given list of elements is firstly back-upped.*
- int [Ak\\_delete\\_update\\_segment](#) (struct [list\\_node](#) \*row\_root, int del)  
*Function updates or deletes the whole segment of an table. Addresses for given table atr fetched. For each block in extent row is updated or deleted according to operator del.*
- int [Ak\\_delete\\_row](#) (struct [list\\_node](#) \*row\_root)  
*Function deletes rows.*
- int [Ak\\_update\\_row](#) (struct [list\\_node](#) \*row\_root)  
*Function updates rows of some table.*
- void [Ak\\_fileio\\_test](#) ()
- void [Ak\\_delete\\_row\\_by\\_id](#) (int id, char \*tableName)  
*Function deletes row by id.*

### 5.19.1 Detailed Description

Header file provides data structures for file input/output

### 5.19.2 Function Documentation

### 5.19.2.1 Ak\_delete\_row()

```
int Ak_delete_row (
    struct list_node * row_root )
```

Function deletes rows.

#### Author

Matija Novak, Dejan Frankovic (added referential integrity)

#### Parameters

|                 |   |
|-----------------|---|
| <i>row_root</i> | elements of one row EXIT_SUCCESS if success |
|-----------------|---|

### 5.19.2.2 Ak\_delete\_row\_by\_id()

```
void Ak_delete_row_by_id (
    int id,
    char * tableName )
```

Function deletes row by id.

#### Author

Dražen Bandić

#### Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>id</i>        | id of row                       |
| <i>tableName</i> | name of table to delete the row |

### 5.19.2.3 Ak\_delete\_row\_from\_block()

```
void Ak_delete_row_from_block (
    AK_block * temp_block,
    struct list_node * row_root )
```

Function deletes row from table in given block. Given list of elements is firstly back-upped.

#### Author

Matija Novak, updated by Dino Laktašić, changed by Davorin Vukelic, updated by Mario Peroković



## Parameters

|                   |  |
|-------------------|--|
| <i>temp_block</i> | block to work with                                       |
| <i>row_list</i>   | list of elements which contain data for delete or update |

## Returns

No return value

## 5.19.2.4 Ak\_delete\_update\_segment()

```
int Ak_delete_update_segment (
    struct list_node * row_root,
    int del )
```

Function updates or deletes the whole segment of an table. Addresses for given table atr fetched. For each block in extent row is updated or deleted according to operator del.

## Author

Matija Novak, updated by Matija Šestak (function now uses caching)

## Parameters

|                 |                     |
|-----------------|---------------------|
| <i>row_root</i> | elements of one row |
| <i>del</i>      | - DELETE or UPDATE  |

## Returns

EXIT\_SUCCESS if success

## 5.19.2.5 Ak\_Insert\_New\_Element()

```
void Ak_Insert_New_Element (
    int newtype,
    void * data,
    char * table,
    char * attribute_name,
    struct list_node * ElementBefore )
```

Function inserts new element after some element, to insert on first place give list as before element. It calls function Ak\_Insert\_New\_Element\_For\_Update.

## Author

Matija Novak, changed by Dino Laktašić

**Parameters**

|                       |   |
|-----------------------|---|
| <i>newtype</i>        | type of the data                              |
| <i>data</i>           | the data                                      |
| <i>table</i>          | table name                                    |
| <i>attribute_name</i> | attribute name                                |
| <i>element</i>        | element after we which insert the new element |
| <i>constraint</i>     | is NEW_VALUE                                  |

**Returns**

No return value

**5.19.2.6 Ak\_Insert\_New\_Element\_For\_Update()**

```
void Ak_Insert_New_Element_For_Update (
    int newtype,
    void * data,
    char * table,
    char * attribute_name,
    struct list_node * ElementBefore,
    int newconstraint )
```

Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elements are set according to function arguments. Pointers are changed so that before element points to new element.

**Author**

Matija Novak

**Parameters**

|                       |   |
|-----------------------|---|
| <i>newtype</i>        | type of the data  |
| <i>data</i>           | the data  |
| <i>table</i>          | table name  |
| <i>attribute_name</i> | attribute name  |
| <i>element</i>        | element after we which insert the new element   |
| <i>constraint</i>     | NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for |

**Returns**

No return value

#### 5.19.2.7 Ak\_insert\_row()

```
int Ak_insert_row (
    struct list_node * row_root )
```

Function inserts a one row into table. Firstly it is checked whether inserted row would violate reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_DIRTY.

##### Author

Matija Novak, updated by Matija Šestak (function now uses caching), updated by Dejan Frankovic (added reference check), updated by Dino Laktašić (removed variable AK\_free, variable table initialized using memset)

##### Parameters

|                 |  |
|-----------------|--|
| <i>row_root</i> | list of elements which contain data of one row |
|-----------------|--|

##### Returns

EXIT\_SUCCESS if success else EXIT\_ERROR

#### 5.19.2.8 Ak\_insert\_row\_to\_block()

```
int Ak_insert_row_to_block (
    struct list_node * row_root,
    AK_block * temp_block )
```

Function inserts one row into some block. Firstly it checks whether block contains attributes from the list. Then data, type, size and last\_tuple\_id are put in temp\_block.

##### Author

Matija Novak, updated by Dino Laktašić

##### Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>row_root</i>   | list of elements to insert    |
| <i>temp_block</i> | block in which we insert data |

##### Returns

EXIT\_SUCCESS if success

### 5.19.2.9 Ak\_update\_row()

```
int Ak_update_row (
    struct list_node * row_root )
```

Function updates rows of some table.

#### Author

Matija Novak, Dejan Frankovic (added referential integrity)

#### Parameters

|                 |                     |
|-----------------|---------------------|
| <i>row_root</i> | elements of one row |
|-----------------|---------------------|

#### Returns

EXIT\_SUCCESS if success

### 5.19.2.10 Ak\_update\_row\_from\_block()

```
void Ak_update_row_from_block (
    AK_block * temp_block,
    struct list_node * row_root )
```

Function updates row from table in given block.

#### Author

Matija Novak, updated by Dino Laktašić, updated by Mario Peroković - separated from deletion

#### Parameters

|                   |  |
|-------------------|--|
| <i>temp_block</i> | block to work with                                       |
| <i>row_list</i>   | list of elements which contain data for delete or update |

#### Returns

No return value

## 5.20 file/files.c File Reference

```
#include "files.h"
#include <pthread.h>
```



### 5.20.2.2 AK\_initialize\_new\_index\_segment()

```
int AK_initialize_new_index_segment (
    char * name,
    char * table_id,
    int attr_id,
    AK_header * header )
```

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

#### Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching), reused by Lovro Predovan

#### Parameters

|               |   |
|---------------|---|
| <i>name</i>   | segment name  |
| <i>type</i>   | segment type  |
| <i>header</i> | pointer to header that should be written to the new extent (all blocks) |

#### Returns

start address of new segment

### 5.20.2.3 AK\_initialize\_new\_segment()

```
int AK_initialize_new_segment (
    char * name,
    int type,
    AK_header * header )
```

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

#### Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching)

#### Parameters

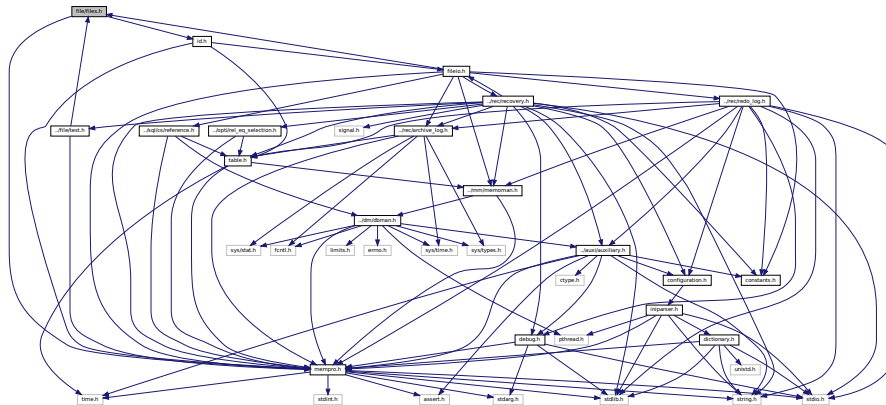
|               |   |
|---------------|---|
| <i>name</i>   | segment name  |
| <i>type</i>   | segment type  |
| <i>header</i> | pointer to header that should be written to the new extent (all blocks) |

## Returns

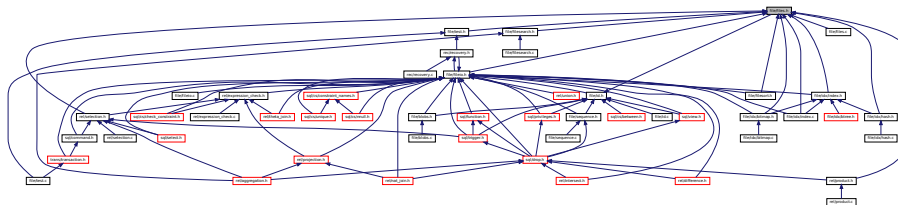
start address of new segment

## 5.21 file/files.h File Reference

```
#include "id.h"
#include "../auxi/mempro.h"
Include dependency graph for files.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- int [AK\\_initialize\\_new\\_segment](#) (char \*name, int type, [AK\\_header](#) \*header)  
*Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.*
- int [AK\\_initialize\\_new\\_index\\_segment](#) (char \*name, char \*table\_id, int attr\_id, [AK\\_header](#) \*header)  
*Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.*
- void [Ak\\_files\\_test](#) ()  
*Test function.*

### 5.21.1 Detailed Description

Header file that provides data structures for file management

## 5.21.2 Function Documentation

### 5.21.2.1 `Ak_files_test()`

```
void Ak_files_test ( )
```

Test function.

#### Author

Unknown

#### Returns

No return value

### 5.21.2.2 `AK_initialize_new_index_segment()`

```
int AK_initialize_new_index_segment (
    char * name,
    char * table_id,
    int attr_id,
    AK_header * header )
```

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

#### Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching), reused by Lovro Predovan

#### Parameters

|               |   |
|---------------|---|
| <i>name</i>   | segment name  |
| <i>type</i>   | segment type  |
| <i>header</i> | pointer to header that should be written to the new extent (all blocks) |

#### Returns

start address of new segment



### 5.21.2.3 AK\_initialize\_new\_segment()

```
int AK_initialize_new_segment (
    char * name,
    int type,
    AK_header * header )
```

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

### Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching)

## Parameters

|               |   |
|---------------|---|
| <i>name</i>   | segment name  |
| <i>type</i>   | segment type  |
| <i>header</i> | pointer to header that should be written to the new extent (all blocks) |

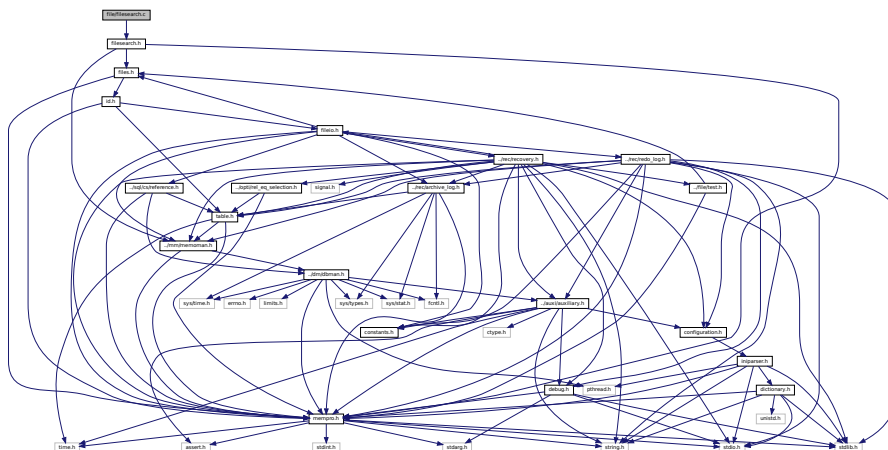
## Returns

start address of new segment

## 5.22 file/filesearch.c File Reference

```
#include "filesearch.h"
```

Include dependency graph for filesearch.c:



## Functions

- `search_result AK_search_unsorted` (char \*szRelation, `search_params` \*aspParams, int iNum\_search\_params)

Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned ( $A == 1$  AND  $B == 7$  AND ...). `SEARCH_RANGE` is inclusive. Only one value (or range) per attribute allowed - use [search\\_params.pData\\_lower](#) for `SEARCH_PARTICULAR`. Supported types for `SEARCH_RANGE`: `TYPE_INT`, `TYPE_FLOAT`, `TYPE_NUMBER`, `TYPE_DATE`, `TYPE_DATETIME`, `TYPE_TIME`. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.

- void [AK\\_deallocate\\_search\\_result](#) ([search\\_result](#) srResult)

Function deallocates memory used by search result returned by `AK_search_unsorted`.

- void [Ak\\_filesearch\\_test](#) ()

Function for testing file search.

### 5.22.1 Detailed Description

Provides functions for file searching

### 5.22.2 Function Documentation

#### 5.22.2.1 AK\_deallocate\_search\_result()

```
void AK_deallocate_search_result (
    search\_result srResult )
```

Function deallocates memory used by search result returned by `AK_search_unsorted`.

#### Author

Miroslav Policki

#### Parameters

|                 |               |
|-----------------|---------------|
| <i>srResult</i> | search result |
|-----------------|---------------|

#### Returns

No return value

#### 5.22.2.2 Ak\_filesearch\_test()

```
void Ak_filesearch_test ( )
```

Function for testing file search.

**Author**

Miroslav Policki

**Returns**

No return value

**5.22.2.3 AK\_search\_unsorted()**

```
search_result AK_search_unsorted (
    char * szRelation,
    search_params * aspParams,
    int iNum_search_params )
```

Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH\_RANGE is inclusive. Only one value (or range) per attribute allowed - use [search\\_params.pData\\_lower](#) for SEARCH\_PARTICULAR. Supported types for SEARCH\_RANGE: TYPE\_INT, TYPE\_FLOAT, TYPE\_NUMBER, TYPE\_DATE, TYPE\_DATETIME, TYPE\_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.

**Author**

Miroslav Policki

**Parameters**

|                           |                             |
|---------------------------|-----------------------------|
| <i>szRelation</i>         | relation name               |
| <i>aspParams</i>          | array of search parameters  |
| <i>iNum_search_params</i> | number of search parameters |

**Returns**

[search\\_result](#) structure defined in [filesearch.h](#). Use [AK\\_deallocate\\_search\\_result](#) to deallocate.

iterate through all the blocks

count number of attributes in segment/relation

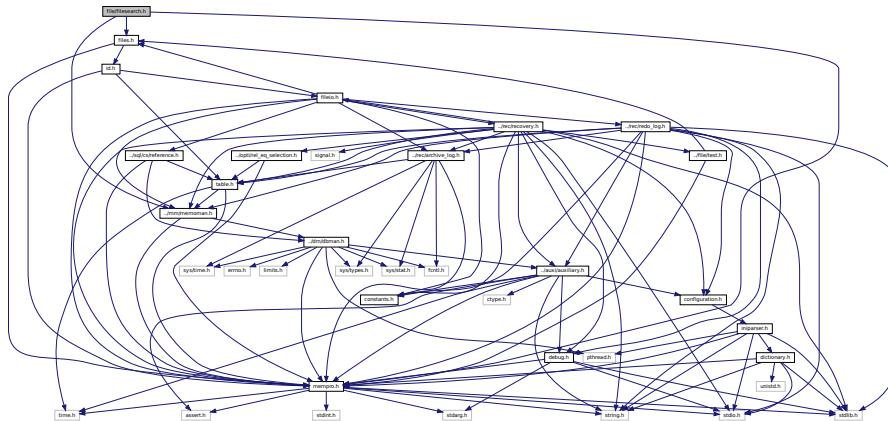
determine index of attributes on which search will be performed

if any of the provided attributes are not found in the relation, return empty result

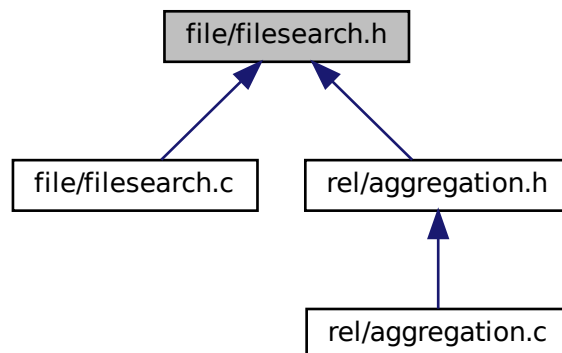
in every tuple, for all required attributes, compare attribute value with searched-for value and store matched tuple addresses

## 5.23 file/filesearch.h File Reference

```
#include "../mm/memoman.h"
#include "files.h"
#include "../aux/mempro.h"
Include dependency graph for filesearch.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [search\\_params](#)

Structure that contains attribute name, lower and upper data value, special(NULL or \*) which is input for AK\_↔equisearch\_unsorted and AK\_rangesearch\_unsorted.

- struct [search\\_result](#)

Structure which represents search result of AK\_equisearch\_unsorted and AK\_rangesearch\_unsorted.

## Macros

- `#define SEARCH_NULL 0`
- `#define SEARCH_ALL 1`
- `#define SEARCH_PARTICULAR 2`
- `#define SEARCH_RANGE 3`

## Functions

- [search\\_result AK\\_search\\_unsorted](#) (char \*szRelation, [search\\_params](#) \*aspParams, int iNum\_search\_params)
 

*Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH\_RANGE is inclusive. Only one value (or range) per attribute allowed - use [search\\_params.pData\\_lower](#) for SEARCH\_PARTICULAR. Supported types for SEARCH\_RANGE: TYPE\_INT, TYPE\_FLOAT, TYPE\_NUMBER, TYPE\_DATE, TYPE\_DATETIME, TYPE\_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.*
- void [AK\\_deallocate\\_search\\_result](#) ([search\\_result](#) srResult)
 

*Function deallocates memory used by search result returned by AK\_search\_unsorted.*
- void [Ak\\_filesearch\\_test](#) ()
 

*Function for testing file search.*

### 5.23.1 Detailed Description

Header file provides data structures for file searching

### 5.23.2 Function Documentation

#### 5.23.2.1 AK\_deallocate\_search\_result()

```
void AK_deallocate_search_result (
    search\_result srResult )
```

Function deallocates memory used by search result returned by AK\_search\_unsorted.

#### Author

Miroslav Policki

#### Parameters

|                 |               |
|-----------------|---------------|
| <i>srResult</i> | search result |
|-----------------|---------------|

#### Returns

No return value

### 5.23.2.2 Ak\_filesearch\_test()

```
void Ak_filesearch_test ( )
```

Function for testing file search.

#### Author

Miroslav Policki

#### Returns

No return value

### 5.23.2.3 AK\_search\_unsorted()

```
search_result AK_search_unsorted (
    char * szRelation,
    search_params * aspParams,
    int iNum_search_params )
```

Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH\_RANGE is inclusive. Only one value (or range) per attribute allowed - use [search\\_params.pData\\_lower](#) for SEARCH\_PARTICULAR. Supported types for SEARCH\_RANGE: TYPE\_INT, TYPE\_FLOAT, TYPE\_NUMBER, TYPE\_DATE, TYPE\_DATETIME, TYPE\_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.

#### Author

Miroslav Policki

#### Parameters

|                           |                             |
|---------------------------|-----------------------------|
| <i>szRelation</i>         | relation name               |
| <i>aspParams</i>          | array of search parameters  |
| <i>iNum_search_params</i> | number of search parameters |

#### Returns

[search\\_result](#) structure defined in [filesearch.h](#). Use [AK\\_deallocate\\_search\\_result](#) to deallocate.

iterate through all the blocks

count number of attributes in segment/relation



### 5.24.1 Detailed Description

Header filr provides data structures for file sorting

### 5.24.2 Function Documentation

#### 5.24.2.1 AK\_block\_sort()

```
void AK_block_sort (
    AK_block * iBlock,
    char * atr_name )
```

Function sorts the given block.

#### Author

Bakoš Nikola

#### Version

v1.0

#### Parameters

|               |                    |
|---------------|--------------------|
| <i>iBlock</i> | block to be sorted |
|---------------|--------------------|

#### Returns

No return value

#### 5.24.2.2 Ak\_get\_header\_number()

```
int Ak_get_header_number (
    AK_block * iBlock,
    char * attribute_name )
```

Function returns number of header in the block which to sort.

#### Author

Unknown

#### Returns

number of attribute in header (0 - MAX\_ATTRIBUTES). USE in tuple\_dict[num]...





## Functions

- int [AK\\_get\\_id](#) ()  
*Function for getting unique ID for any object, stored in sequence.*
- char [AK\\_get\\_table\\_id](#) (char \*tableName)  
*Function for getting unique ID for any object, stored in sequence based on table name.*
- void [Ak\\_id\\_test](#) ()  
*Function for testing getting ID's.*

### 5.25.1 Detailed Description

Provides functions for creating id of objects

### 5.25.2 Function Documentation

#### 5.25.2.1 [AK\\_get\\_id\(\)](#)

```
int AK_get_id ( )
```

Function for getting unique ID for any object, stored in sequence.

#### Author

Saša Vukšić, updated by Mislav Čakarić, changed by Mario Peroković, now uses Ak\_update\_row, updated by Nenad Makar

#### Returns

objectID

#### 5.25.2.2 [AK\\_get\\_table\\_id\(\)](#)

```
char AK_get_table_id (
    char * tableName )
```

Function for getting unique ID for any object, stored in sequence based on table name.

#### Author

Lovro Predovan

#### Returns

objectID in string(char) format



## Functions

- int [AK\\_get\\_id](#) ()  
*Function for getting unique ID for any object, stored in sequence.*
- void [Ak\\_id\\_test](#) ()  
*Function for testing getting ID's.*

### 5.26.1 Detailed Description

Provides functions, data structures and constants for creating id of objects

### 5.26.2 Function Documentation

#### 5.26.2.1 AK\_get\_id()

```
int AK_get_id ( )
```

Function for getting unique ID for any object, stored in sequence.

#### Author

Saša Vukšić, updated by Mislav Čakarić, changed by Mario Peroković, now uses Ak\_update\_row, updated by Nenad Makar

#### Returns

objectID

#### 5.26.2.2 Ak\_id\_test()

```
void Ak_id_test ( )
```

Function for testing getting ID's.

#### Author

Mislav Čakarić, updated by Nenad Makar

#### Returns

No return value



- void [AK\\_delete\\_bitmap\\_index](#) (char \*indexName)  
*Function that deletes bitmap index based on name of index.*
- void [Ak\\_bitmap\\_test](#) ()  
*Function that creates test table and makes index on test table, also prints original tables indexes tables and indexes,tests updating into tables.*

### 5.27.1 Detailed Description

Provides functions for bitmap indexes

### 5.27.2 Function Documentation

#### 5.27.2.1 AK\_add\_to\_bitmap\_index()

```
void AK_add_to_bitmap_index (
    char * tableName,
    char * attributeName )
```

Function for updating the index,function deletes and recreates index values again if different number of params is detected.

#### Author

Lovro Predovan

#### Parameters

|                          |                                |
|--------------------------|--------------------------------|
| <i>tableName</i>         | name of table                  |
| <i>attributeName</i>     | name of attribute              |
| <i>newAttributeValue</i> | new value of updated attribute |

#### Returns

No return value

#### 5.27.2.2 Ak\_bitmap\_test()

```
void Ak_bitmap_test ( )
```

Function that creates test table and makes index on test table, also prints original tables indexes tables and indexes,tests updating into tables.

**Author**

Saša Vukšić updated by Lovro Predovan

**Returns**

No return value

**5.27.2.3 Ak\_create\_Index()**

```
void Ak_create_Index (
    char * tblName,
    char * tblNameIndex,
    char * attributeName,
    int positionTbl,
    int numAttributes,
    AK_header * headerIndex )
```

Function that loads index table with value of particular attribute.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

|                      |  |
|----------------------|--|
| <i>tblName</i>       | source table                             |
| <i>tblNameIndex</i>  | new name of index table                  |
| <i>attributeName</i> | attribute on which we make index         |
| <i>positionTbl</i>   | position of attribute in header of table |
| <i>numAttributes</i> | number of attributes in table            |
| <i>headerIndex</i>   | header of index table                    |

**Returns**

No return value

**5.27.2.4 AK\_create\_Index\_Table()**

```
void AK_create_Index_Table (
    char * tblName,
    struct list_node * attributes )
```

Function reads table on which we create index and call functions for creating index Elements that will be in index are put in list indexLista and headerAttributes. According to those elements new indexes are created.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

|                   |  |
|-------------------|--|
| <i>tblName</i>    | name of table                                      |
| <i>attributes</i> | list of attributes on which we will create indexes |

**Returns**

No return value

**5.27.2.5 AK\_delete\_bitmap\_index()**

```
void AK_delete_bitmap_index (
    char * indexName )
```

Function that deletes bitmap index based on name of index.

**Author**

Lovro Predovan

**Parameters**

|               |                  |
|---------------|------------------|
| <i>Bitmap</i> | index table name |
|---------------|------------------|

**Returns**

No return value

**5.27.2.6 Ak\_get\_Attribute()**

```
list_ad* Ak_get_Attribute (
    char * indexName,
    char * attribute )
```

Function gets addresses of the particuliary attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. Those data are put in list called add\_root.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

|                  |                   |
|------------------|-------------------|
| <i>indexName</i> | name of index     |
| <i>attribute</i> | name of attribute |



**Returns**

list of adresses

**5.27.2.7 AK\_get\_Attribute()**

```
list_ad* AK_get_Attribute (
    char * tableName,
    char * attributeName,
    char * attributeValue )
```

Function for getting values from the bitmap index if there is one for given table. It should be started when we are making selection on the table with bitmap index.

**Author**

Saša Vukšić

**Parameters**

|                       |                    |
|-----------------------|--------------------|
| <i>tableName</i>      | name of table      |
| <i>attributeValue</i> | value of attribute |

**Returns**

list of adresses

**5.27.2.8 Ak\_If\_ExistOp()**

```
int Ak_If_ExistOp (
    struct list_node * L,
    char * ele )
```

Function examines whether list L contains operator ele.

**Author**

Saša Vukšić

**Parameters**

|            |                              |
|------------|------------------------------|
| <i>L</i>   | list of elements             |
| <i>ele</i> | operator to be found in list |

**Returns**

1 if operator ele is found in list, otherwise 0

**5.27.2.9 Ak\_print\_Att\_Test()**

```
void Ak_print_Att_Test (
    list_ad * list )
```

Function for printing list of addresses.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

|             |                   |
|-------------|-------------------|
| <i>list</i> | list of addresses |
|-------------|-------------------|

**Returns**

No return value

**5.27.2.10 Ak\_print\_Header\_Test()**

```
void Ak_print_Header_Test (
    char * tblName )
```

Function that tests printing header of table.

**Author**

Saša Vukšić

**Parameters**

|                |  |
|----------------|--|
| <i>tblName</i> | name of table who's header we are printing |
|----------------|--|

**Returns**

No return value

## 5.27.2.11 AK\_update()

```
void AK_update (
    int addBlock,
    int addTd,
    char * tableName,
    char * attributeName,
    char * attributeValue,
    char * newAttributeValue )
```

Function for updating the index, only on values that already exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.

## Author

Saša Vukšić

## Parameters

|                          |                                |
|--------------------------|--------------------------------|
| <i>addBlock</i>          | adress of block                |
| <i>addTD</i>             | adress of tuple dict           |
| <i>tableName</i>         | name of table                  |
| <i>attributeName</i>     | name of attribute              |
| <i>attributeValue</i>    | value of attribute             |
| <i>newAttributeValue</i> | new value of updated attribute |

## Returns

No return value

## 5.27.2.12 Ak\_write\_block()

```
int Ak_write_block (
    AK_block * block )
```

Function for writing new value in block when index is updated.

## Author

Saša Vukšić

## Parameters

|              |                   |
|--------------|-------------------|
| <i>block</i> | block to write on |
|--------------|-------------------|

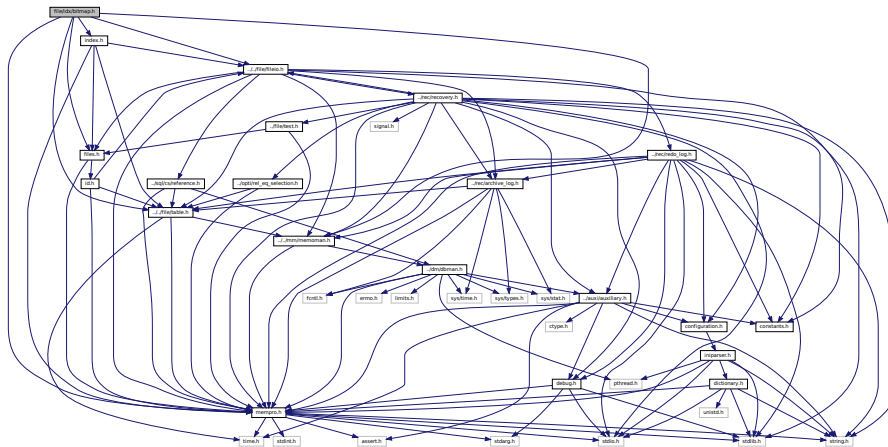
## Returns

EXIT\_SUCESS when write operation is successful, otherwise EXIT\_ERROR

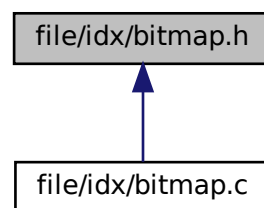
## 5.28 file/idx/bitmap.h File Reference

```
#include "../mm/memoman.h"
#include "index.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../file/files.h"
#include "../aux/mempro.h"
```

Include dependency graph for bitmap.h:



This graph shows which files directly or indirectly include this file:



### Functions

- int [Ak\\_If\\_ExistOp](#) (struct [list\\_node](#) \*L, char \*ele)  
*Function examines whether list L contains operator ele.*
- void [AK\\_create\\_Index\\_Table](#) (char \*tblName, struct [list\\_node](#) \*attributes)  
*Function reads table on which we create index and call functions for creating index Elements that will be in index are put in list indexList and headerAttributes. According to those elements new indexes are created.*
- void [Ak\\_print\\_Header\\_Test](#) (char \*tblName)  
*Function that tests printing header of table.*

- void [Ak\\_create\\_Index](#) (char \*tblName, char \*tblNameIndex, char \*attributeName, int positionTbl, int numAttributes, [AK\\_header](#) \*headerIndex)  
*Function that loads index table with value of particular attribute.*
- [list\\_ad](#) \* [Ak\\_get\\_Attribute](#) (char \*indexName, char \*attribute)  
*Function gets addresses of the particular attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. Those data are put in list called add\_root.*
- void [Ak\\_create\\_List\\_Address\\_Test](#) ()
- void [Ak\\_print\\_Att\\_Test](#) ([list\\_ad](#) \*list)  
*Function for printing list of addresses.*
- [list\\_ad](#) \* [AK\\_get\\_Attribute](#) (char \*tableName, char \*attributeName, char \*attributeValue)  
*Function for getting values from the bitmap index if there is one for given table. It should be started when we are making selection on the table with bitmap index.*
- void [AK\\_update](#) (int addBlock, int addTd, char \*tableName, char \*attributeName, char \*attributeValue, char \*newAttributeValue)  
*Function for updating the index, only on values that already exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.*
- int [AK\\_write\\_block](#) ([AK\\_block](#) \*block)  
*Function for writing new value in block when index is updated.*
- void [AK\\_bitmap\\_test](#) ()  
*Function that creates test table and makes index on test table, also prints original tables indexes tables and indexes, tests updating into tables.*
- void [AK\\_delete\\_bitmap\\_index](#) (char \*indexName)  
*Function that deletes bitmap index based on name of index.*
- void [AK\\_add\\_to\\_bitmap\\_index](#) (char \*tableName, char \*attributeName)  
*Function for updating the index, function deletes and recreates index values again if different number of params is detected.*

## 5.28.1 Detailed Description

Header file that provides data structures for bitmap index

## 5.28.2 Function Documentation

### 5.28.2.1 AK\_add\_to\_bitmap\_index()

```
void AK_add_to_bitmap_index (
    char * tableName,
    char * attributeName )
```

Function for updating the index, function deletes and recreates index values again if different number of params is detected.

Author

Lovro Predovan

**Parameters**

|                          |                                |
|--------------------------|--------------------------------|
| <i>tableName</i>         | name of table                  |
| <i>attributeName</i>     | name of attribute              |
| <i>newAttributeValue</i> | new value of updated attribute |

**Returns**

No return value

**5.28.2.2 Ak\_bitmap\_test()**

```
void Ak_bitmap_test ( )
```

Function that creates test table and makes index on test table, also prints original tables indexes tables and indexes,tests updating into tables.

**Author**

Saša Vukšić updated by Lovro Predovan

**Returns**

No return value

**5.28.2.3 Ak\_create\_Index()**

```
void Ak_create_Index (
    char * tblName,
    char * tblNameIndex,
    char * attributeName,
    int positionTbl,
    int numAttributes,
    AK_header * headerIndex )
```

Function that loads index table with value of particular attribute.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

|                      |  |
|----------------------|--|
| <i>tblName</i>       | source table                             |
| <i>tblNameIndex</i>  | new name of index table                  |
| <i>attributeName</i> | attribute on which we make index         |
| <i>positionTbl</i>   | position of attribute in header of table |
| <i>numAttributes</i> | number of attributes in table            |
| <i>headerIndex</i>   | header of index table                    |

**Returns**

No return value

**5.28.2.4 AK\_create\_Index\_Table()**

```
void AK_create_Index_Table (
    char * tblName,
    struct list_node * attributes )
```

Function reads table on which we create index and call functions for creating index Elements that will be in index are put in list indexLista and headerAttributes. According to those elements new indexes are created.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

|                   |  |
|-------------------|--|
| <i>tblName</i>    | name of table                                      |
| <i>attributes</i> | list of attributes on which we will create indexes |

**Returns**

No return value

**5.28.2.5 AK\_delete\_bitmap\_index()**

```
void AK_delete_bitmap_index (
    char * indexName )
```

Function that deletes bitmap index based on name of index.

**Author**

Lovro Predovan

**Parameters**

|               |                  |
|---------------|------------------|
| <i>Bitmap</i> | index table name |
|---------------|------------------|

**Returns**

No return value

#### 5.28.2.6 Ak\_get\_Attribute()

```
list_ad* Ak_get_Attribute (
    char * indexName,
    char * attribute )
```

Function gets addresses of the particuliary attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. Those data are put in list called add\_root.

##### Author

Saša Vukšić, Lovro Predovan

##### Parameters

|                  |                   |
|------------------|-------------------|
| <i>indexName</i> | name of index     |
| <i>attribute</i> | name of attribute |

##### Returns

list of adresses

#### 5.28.2.7 AK\_get\_Attribute()

```
list_ad* AK_get_Attribute (
    char * tableName,
    char * attributeName,
    char * attributeValue )
```

Function for getting values from the bitmap index if there is one for given table. It should be started when we are making selection on the table with bitmap index.

##### Author

Saša Vukšić

##### Parameters

|                       |                    |
|-----------------------|--------------------|
| <i>tableName</i>      | name of table      |
| <i>attributeValue</i> | value of attribute |

##### Returns

list of adresses



### 5.28.2.8 Ak\_If\_ExistOp()

```
int Ak_If_ExistOp (
    struct list_node * L,
    char * ele )
```

Function examines whether list L contains operator ele.

#### Author

Saša Vukšić

#### Parameters

|            |                              |
|------------|------------------------------|
| <i>L</i>   | list of elements             |
| <i>ele</i> | operator to be found in list |

#### Returns

1 if operator ele is found in list, otherwise 0

### 5.28.2.9 Ak\_print\_Att\_Test()

```
void Ak_print_Att_Test (
    list_ad * list )
```

Function for printing list of addresses.

#### Author

Saša Vukšić, Lovro Predovan

#### Parameters

|             |                   |
|-------------|-------------------|
| <i>list</i> | list of addresses |
|-------------|-------------------|

#### Returns

No return value

### 5.28.2.10 Ak\_print\_Header\_Test()

```
void Ak_print_Header_Test (
    char * tblName )
```

Function that tests printing header of table.

**Author**

Saša Vukšić

**Parameters**

|                |  |
|----------------|--|
| <i>tblName</i> | name of table who's header we are printing |
|----------------|--|

**Returns**

No return value

**5.28.2.11 AK\_update()**

```
void AK_update (
    int addBlock,
    int addTd,
    char * tableName,
    char * attributeName,
    char * attributeValue,
    char * newAttributeValue )
```

Function for updating the index, only on values that already exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.

**Author**

Saša Vukšić

**Parameters**

|                          |                                |
|--------------------------|--------------------------------|
| <i>addBlock</i>          | adress of block                |
| <i>addTD</i>             | adress of tuple dict           |
| <i>tableName</i>         | name of table                  |
| <i>attributeName</i>     | name of attribute              |
| <i>attributeValue</i>    | value of attribute             |
| <i>newAttributeValue</i> | new value of updated attribute |

**Returns**

No return value

**5.28.2.12 Ak\_write\_block()**

```
int Ak_write_block (
    AK_block * block )
```

Function for writing new value in block when index is updated.

## Author

Saša Vukšić

## Parameters

|              |                   |
|--------------|-------------------|
| <i>block</i> | block to write on |
|--------------|-------------------|

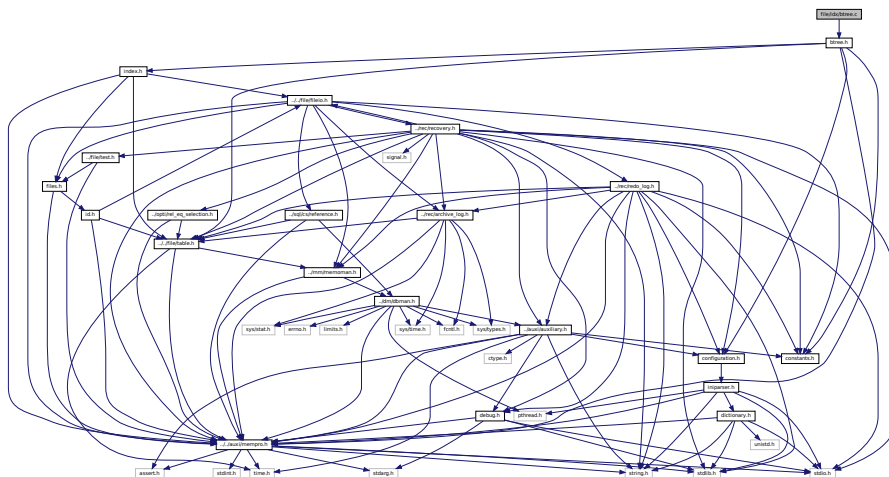
## Returns

EXIT\_SUCESS when write operation is successful, otherwise EXIT\_ERROR

## 5.29 file/idx/btree.c File Reference

```
#include "btree.h"
```

Include dependency graph for btree.c:



## Functions

- int **AK\_btree\_create** (char \*tblName, struct **list\_node** \*attributes, char \*indexName)  
*Function for creating new btree index on integer attribute in table.*
- int **AK\_btree\_delete** (char \*indexName)
- void **AK\_btree\_search\_delete** (char \*indexName, int \*searchValue, int \*endRange, int \*toDo)  
*Function for searching or deleting a value in btree index.*
- int **AK\_btree\_insert** (char \*indexName, int \*insertValue, int \*insertTd, int \*insertBlock)
- void **Ak\_btree\_test** ()

### 5.29.1 Detailed Description

Header file that provides functions for BTree indices

## 5.29.2 Function Documentation

### 5.29.2.1 AK\_btree\_create()

```
int AK_btree_create (
    char * tblName,
    struct list_node * attributes,
    char * indexName )
```

Function for creating new btree index on integer attribute in table.

#### Author

Andelko Spevec

#### Parameters

|                   |  |
|-------------------|--|
| <i>tblName</i>    | - name of the table on which we are creating index |
| <i>attributes</i> | - attribute on which we are creating index         |
| <i>indexName</i>  | - name of the index                                |

### 5.29.2.2 AK\_btree\_search\_delete()

```
void AK_btree_search_delete (
    char * indexName,
    int * searchValue,
    int * endRange,
    int * todo )
```

Function for searching or deleting a value in btree index.

#### Author

Andelko Spevec

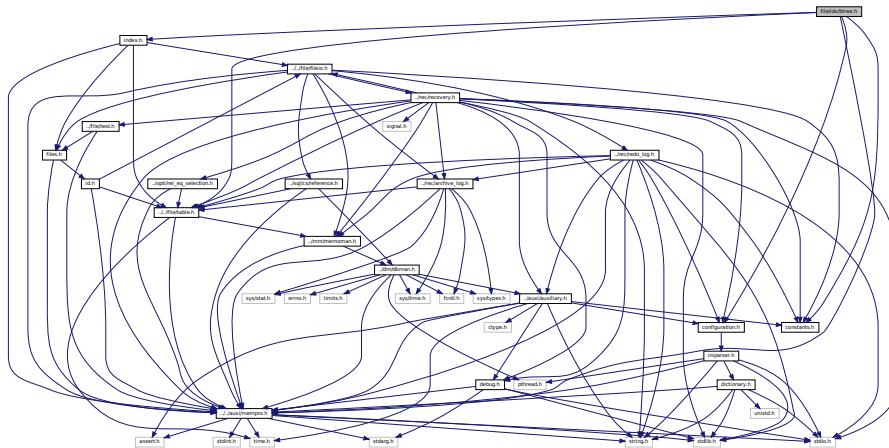
#### Parameters

|                    |  |
|--------------------|--|
| <i>indexName</i>   | - name of the index  |
| <i>searchValue</i> | - value that we are searching in the index                     |
| <i>endRange</i>    | - if 0 search is for 0 value, else searching in range          |
| <i>todo</i>        | - if 0 we just search else we delete the element if we find it |

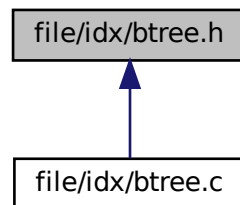
## 5.30 file/idx/btree.h File Reference

```
#include "index.h"
#include "../../file/table.h"
#include "../../auxi/constants.h"
#include "../../auxi/configuration.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for btree.h:



This graph shows which files directly or indirectly include this file:



### Classes

- struct [btree\\_\\_node](#)
- struct [root\\_info](#)

### Macros

- `#define B 3`
- `#define ORDER 6`
- `#define LEAF 0`
- `#define NODE 1`

## Functions

- int [AK\\_btree\\_create](#) (char \*tblName, struct [list\\_node](#) \*attributes, char \*indexName)  
*Function for creating new btree index on integer attribute in table.*
- int [AK\\_btree\\_delete](#) (char \*indexName)
- void [AK\\_btree\\_search\\_delete](#) (char \*indexName, int \*searchValue, int \*endRange, int \*toDo)  
*Function for searching or deleting a value in btree index.*
- int [AK\\_btree\\_insert](#) (char \*indexName, int \*insertValue, int \*insertTd, int \*insertBlock)
- void [Ak\\_btree\\_test](#) ()

### 5.30.1 Detailed Description

Header file that provides data structures for BTree indices

### 5.30.2 Function Documentation

#### 5.30.2.1 AK\_btree\_create()

```
int AK_btree_create (
    char * tblName,
    struct list\_node * attributes,
    char * indexName )
```

Function for creating new btree index on integer attribute in table.

#### Author

Andelko Spevec

#### Parameters

|                   |  |
|-------------------|--|
| <i>tblName</i>    | - name of the table on which we are creating index |
| <i>attributes</i> | - attribute on which we are creating index         |
| <i>indexName</i>  | - name of the index                                |

#### 5.30.2.2 AK\_btree\_search\_delete()

```
void AK_btree_search_delete (
    char * indexName,
    int * searchValue,
    int * endRange,
    int * toDo )
```

Function for searching or deleting a value in btree index.

## Author

Andelko Spevec

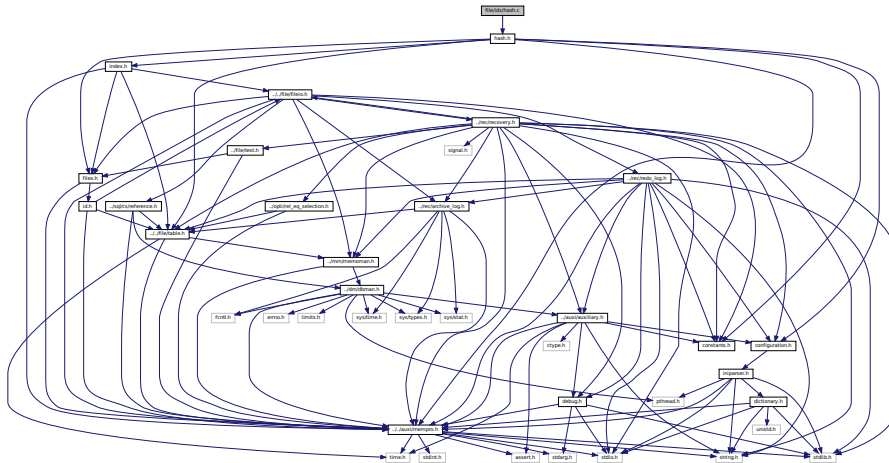
## Parameters

|                    |  |
|--------------------|--|
| <i>indexName</i>   | - name of the index  |
| <i>searchValue</i> | - value that we are searching in the index                     |
| <i>endRange</i>    | - if 0 search is for 0 value, else searching in range          |
| <i>toDo</i>        | - if 0 we just search else we delete the element if we find it |

## 5.31 file/idx/hash.c File Reference

```
#include "hash.h"
```

Include dependency graph for hash.c:



## Functions

- `int AK_elem_hash_value (struct list_node *elem)`  
*Function for computing a hash value from varchar or integer.*
- `struct_add * Ak_insert_bucket_to_block (char *indexName, char *data, int type)`  
*Function for inserting bucket to block.*
- `void Ak_update_bucket_in_block (struct_add *add, char *data)`  
*Function for update bucket in block.*
- `void AK_change_hash_info (char *indexName, int modulo, int main_bucket_num, int hash_bucket_num)`  
*Function for changing info of hash index.*
- `hash_info * AK_get_hash_info (char *indexName)`  
*Function for fetching info for hash index.*
- `struct_add * Ak_get_nth_main_bucket_add (char *indexName, int n)`  
*Function for fetching nth main bucket.*
- `void AK_insert_in_hash_index (char *indexName, int hashValue, struct_add *add)`  
*Function for inserting record in hash bucket.*
- `struct_add * AK_find_delete_in_hash_index (char *indexName, struct list_node *values, int delete)`

*Function for fetching or deleting record from hash index.*

- `struct_add * AK_find_in_hash_index` (char \*indexName, struct `list_node` \*values)

*Function for fetching record from hash index.*

- void `AK_delete_in_hash_index` (char \*indexName, struct `list_node` \*values)

*Function for deleting record from hash index.*

- int `AK_create_hash_index` (char \*tblName, struct `list_node` \*attributes, char \*indexName)

*Function for creating hash index.*

- void `AK_delete_hash_index` (char \*indexName)

- void `Ak_hash_test` ()

*Function for testing hash index.*

### 5.31.1 Detailed Description

Provides functions for Hash indices

### 5.31.2 Function Documentation

#### 5.31.2.1 AK\_change\_hash\_info()

```
void AK_change_hash_info (
    char * indexName,
    int modulo,
    int main_bucket_num,
    int hash_bucket_num )
```

Function for changing info of hash index.

#### Author

Mislav Čakarić

#### Parameters

|                        |                                |
|------------------------|--------------------------------|
| <i>indexName</i>       | name of index                  |
| <i>modulo</i>          | value for modulo hash function |
| <i>main_bucket_num</i> | number of main buckets         |
| <i>hash_bucket_num</i> | number of hash buckets         |

#### Returns

No return value



### 5.31.2.2 AK\_create\_hash\_index()

```
int AK_create_hash_index (
    char * tblName,
    struct list_node * attributes,
    char * indexName )
```

Function for creating hash index.

#### Author

Mislav Čakarić

#### Parameters

|                   |  |
|-------------------|--|
| <i>tblName</i>    | name of table for which the index is being created       |
| <i>indexName</i>  | name of index  |
| <i>attributes</i> | list of attributes over which the index is being created |

#### Returns

success or error

### 5.31.2.3 AK\_delete\_in\_hash\_index()

```
void AK_delete_in_hash_index (
    char * indexName,
    struct list_node * values )
```

Function for deleting record from hash index.

#### Author

Mislav Čakarić

#### Parameters

|                  |  |
|------------------|--|
| <i>indexName</i> | name of index                                    |
| <i>values</i>    | list of values (one row) to search in hash index |

#### Returns

No return value

#### 5.31.2.4 AK\_elem\_hash\_value()

```
int AK_elem_hash_value (
    struct list_node * elem )
```

Function for computing a hash value from varchar or integer.

##### Author

Mislav Čakarić

##### Parameters

|             |   |
|-------------|---|
| <i>elem</i> | element of row for wich value is to be computed |
|-------------|---|

##### Returns

hash value

#### 5.31.2.5 AK\_find\_delete\_in\_hash\_index()

```
struct_add* AK_find_delete_in_hash_index (
    char * indexName,
    struct list_node * values,
    int delete )
```

Function for fetching or deleting record from hash index.

##### Author

Mislav Čakarić

##### Parameters

|                  |  |
|------------------|--|
| <i>indexName</i> | name of index  |
| <i>values</i>    | list of values (one row) to search in hash index                               |
| <i>delete</i>    | if delete is 0 then record is only read otherwise it's deleted from hash index |

##### Returns

address structure with data where the record is in table

#### 5.31.2.6 AK\_find\_in\_hash\_index()

```
struct_add* AK_find_in_hash_index (
    char * indexName,
    struct list_node * values )
```

Function for fetching record from hash index.

**Author**

Mislav Čakarić

**Parameters**

|                  |  |
|------------------|--|
| <i>indexName</i> | name of index                                    |
| <i>values</i>    | list of values (one row) to search in hash index |

**Returns**

address structure with data where the record is in table

**5.31.2.7 AK\_get\_hash\_info()**

```
hash_info* AK_get_hash_info (  
    char * indexName )
```

Function for fetching info for hash index.

**Author**

Mislav Čakarić

**Parameters**

|                  |               |
|------------------|---------------|
| <i>indexName</i> | name of index |
|------------------|---------------|

**Returns**

info bucket with info data for hash index

**5.31.2.8 Ak\_get\_nth\_main\_bucket\_add()**

```
struct_add* Ak_get_nth_main_bucket_add (  
    char * indexName,  
    int n )
```

Function for fetching nth main bucket.

**Author**

Mislav Čakarić

**Parameters**

|                  |                       |
|------------------|-----------------------|
| <i>indexName</i> | name of index         |
| <i>n</i>         | number of main bucket |

**Returns**

address structure with data where the bucket is stored

**5.31.2.9 Ak\_hash\_test()**

```
void Ak_hash_test ( )
```

Function for testing hash index.

**Author**

Mislav Čakarić

**Returns**

No return value

**5.31.2.10 Ak\_insert\_bucket\_to\_block()**

```
struct_add* Ak_insert_bucket_to_block (
    char * indexName,
    char * data,
    int type )
```

Function for inserting bucket to block.

**Author**

Mislav Čakarić

**Parameters**

|                  |   |
|------------------|---|
| <i>indexName</i> | name of index                               |
| <i>data</i>      | content of bucket stored in char array      |
| <i>type</i>      | type of bucket (MAIN_BUCKET or HASH_BUCKET) |

**Returns**

address structure with data where the bucket is stored

**5.31.2.11 AK\_insert\_in\_hash\_index()**

```
void AK_insert_in_hash_index (
    char * indexName,
    int hashValue,
    struct_add * add )
```

Function for inserting record in hash bucket.

**Author**

Mislav Čakarić

**Parameters**

|                  |   |
|------------------|---|
| <i>indexName</i> | name of index   |
| <i>hashValue</i> | hash value of record that is being inserted                 |
| <i>add</i>       | address structure with data where the hash bucket is stored |

**Returns**

No return value

**5.31.2.12 Ak\_update\_bucket\_in\_block()**

```
void Ak_update_bucket_in_block (
    struct_add * add,
    char * data )
```

Function for update bucket in block.

**Author**

Mislav Čakarić

**Parameters**

|             |  |
|-------------|--|
| <i>add</i>  | address of where the bucket is stored  |
| <i>data</i> | content of bucket stored in char array |

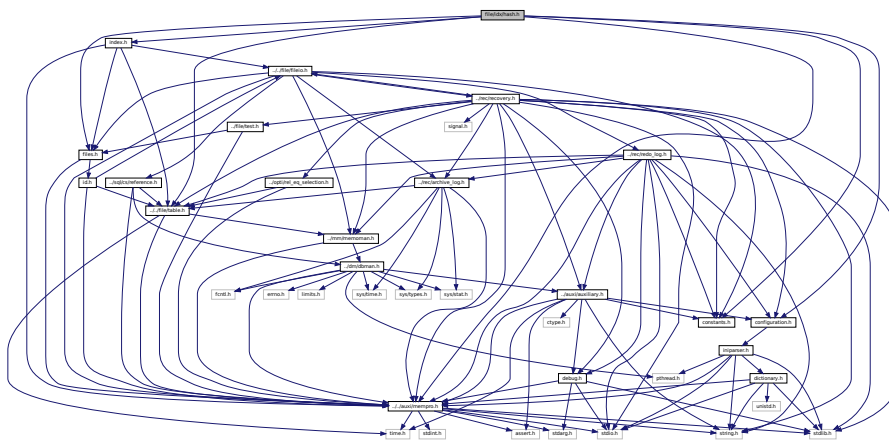
## Returns

No return value

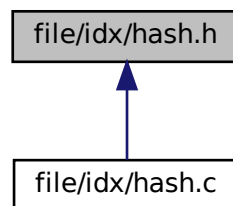
### 5.32 file/idx/hash.h File Reference

```
#include "index.h"
#include "../../file/table.h"
#include "../../aux/constants.h"
#include "../../aux/configuration.h"
#include "../../files.h"
#include "../../aux/mempro.h"
```

Include dependency graph for hash.h:



This graph shows which files directly or indirectly include this file:



### Classes

- struct [hash\\_info](#)  
Structure for defining a hash info element.
- struct [bucket\\_elem](#)  
Structure for defining a single bucket element.

- struct [main\\_bucket](#)  
*Structure for defining main bucket for table hashing.*
- struct [hash\\_bucket](#)  
*Structure for hash bucket for table hashing.*

## Functions

- int [AK\\_elem\\_hash\\_value](#) (struct [list\\_node](#) \*elem)  
*Function for computing a hash value from varchar or integer.*
- struct [add](#) \* [Ak\\_insert\\_bucket\\_to\\_block](#) (char \*indexName, char \*data, int type)  
*Function for inserting bucket to block.*
- void [Ak\\_update\\_bucket\\_in\\_block](#) (struct [add](#) \*add, char \*data)  
*Function for update bucket in block.*
- void [AK\\_change\\_hash\\_info](#) (char \*indexName, int modulo, int main\_bucket\_num, int hash\_bucket\_num)  
*Function for changing info of hash index.*
- hash\_info \* [AK\\_get\\_hash\\_info](#) (char \*indexName)  
*Function for fetching info for hash index.*
- struct [add](#) \* [Ak\\_get\\_nth\\_main\\_bucket\\_add](#) (char \*indexName, int n)  
*Function for fetching nth main bucket.*
- void [AK\\_insert\\_in\\_hash\\_index](#) (char \*indexName, int hashValue, struct [add](#) \*add)  
*Function for inserting record in hash bucket.*
- struct [add](#) \* [AK\\_find\\_delete\\_in\\_hash\\_index](#) (char \*indexName, struct [list\\_node](#) \*values, int delete)  
*Function for fetching or deleting record from hash index.*
- struct [add](#) \* [AK\\_find\\_in\\_hash\\_index](#) (char \*indexName, struct [list\\_node](#) \*values)  
*Function for fetching record from hash index.*
- void [AK\\_delete\\_in\\_hash\\_index](#) (char \*indexName, struct [list\\_node](#) \*values)  
*Function for deleting record from hash index.*
- int [AK\\_create\\_hash\\_index](#) (char \*tblName, struct [list\\_node](#) \*attributes, char \*indexName)  
*Function for creating hash index.*
- void [AK\\_delete\\_hash\\_index](#) (char \*indexName)
- void [Ak\\_hash\\_test](#) ()  
*Function for testing hash index.*

### 5.32.1 Detailed Description

Header file that provides data structures for Hash indices

### 5.32.2 Function Documentation

#### 5.32.2.1 [AK\\_change\\_hash\\_info\(\)](#)

```
void AK_change_hash_info (
    char * indexName,
    int modulo,
    int main_bucket_num,
    int hash_bucket_num )
```

Function for changing info of hash index.

Author

Mislav Čakarić

**Parameters**

|                        |                                |
|------------------------|--------------------------------|
| <i>indexName</i>       | name of index                  |
| <i>modulo</i>          | value for modulo hash function |
| <i>main_bucket_num</i> | number of main buckets         |
| <i>hash_bucket_num</i> | number of hash buckets         |

**Returns**

No return value

**5.32.2.2 AK\_create\_hash\_index()**

```
int AK_create_hash_index (
    char * tblName,
    struct list_node * attributes,
    char * indexName )
```

Function for creating hash index.

**Author**

Mislav Čakarić

**Parameters**

|                   |  |
|-------------------|--|
| <i>tblName</i>    | name of table for which the index is being created       |
| <i>indexName</i>  | name of index  |
| <i>attributes</i> | list of attributes over which the index is being created |

**Returns**

success or error

**5.32.2.3 AK\_delete\_in\_hash\_index()**

```
void AK_delete_in_hash_index (
    char * indexName,
    struct list_node * values )
```

Function for deleting record from hash index.

**Author**

Mislav Čakarić



## Parameters

|                  |  |
|------------------|--|
| <i>indexName</i> | name of index                                    |
| <i>values</i>    | list of values (one row) to search in hash index |

## Returns

No return value

## 5.32.2.4 AK\_elem\_hash\_value()

```
int AK_elem_hash_value (
    struct list_node * elem )
```

Function for computing a hash value from varchar or integer.

## Author

Mislav Čakarić

## Parameters

|             |   |
|-------------|---|
| <i>elem</i> | element of row for wich value is to be computed |
|-------------|---|

## Returns

hash value

## 5.32.2.5 AK\_find\_delete\_in\_hash\_index()

```
struct_add* AK_find_delete_in_hash_index (
    char * indexName,
    struct list_node * values,
    int delete )
```

Function for fetching or deleting record from hash index.

## Author

Mislav Čakarić

## Parameters

|                  |  |
|------------------|--|
| <i>indexName</i> | name of index  |
| <i>values</i>    | list of values (one row) to search in hash index                               |
| <i>delete</i>    | if delete is 0 then record is only read otherwise it's deleted from hash index |

**Returns**

address structure with data where the record is in table

**5.32.2.6 AK\_find\_in\_hash\_index()**

```
struct_add* AK_find_in_hash_index (
    char * indexName,
    struct list_node * values )
```

Function for fetching record from hash index.

**Author**

Mislav Čakarić

**Parameters**

|                  |  |
|------------------|--|
| <i>indexName</i> | name of index                                    |
| <i>values</i>    | list of values (one row) to search in hash index |

**Returns**

address structure with data where the record is in table

**5.32.2.7 AK\_get\_hash\_info()**

```
hash_info* AK_get_hash_info (
    char * indexName )
```

Function for fetching info for hash index.

**Author**

Mislav Čakarić

**Parameters**

|                  |               |
|------------------|---------------|
| <i>indexName</i> | name of index |
|------------------|---------------|

**Returns**

info bucket with info data for hash index

#### 5.32.2.8 Ak\_get\_nth\_main\_bucket\_add()

```
struct_add* Ak_get_nth_main_bucket_add (
    char * indexName,
    int n )
```

Function for fetching nth main bucket.

##### Author

Mislav Čakarić

##### Parameters

|                  |                       |
|------------------|-----------------------|
| <i>indexName</i> | name of index         |
| <i>n</i>         | number of main bucket |

##### Returns

address structure with data where the bucket is stored

#### 5.32.2.9 Ak\_hash\_test()

```
void Ak_hash_test ( )
```

Function for testing hash index.

##### Author

Mislav Čakarić

##### Returns

No return value

#### 5.32.2.10 Ak\_insert\_bucket\_to\_block()

```
struct_add* Ak_insert_bucket_to_block (
    char * indexName,
    char * data,
    int type )
```

Function for inserting bucket to block.

##### Author

Mislav Čakarić

**Parameters**

|                  |   |
|------------------|---|
| <i>indexName</i> | name of index                               |
| <i>data</i>      | content of bucket stored in char array      |
| <i>type</i>      | type of bucket (MAIN_BUCKET or HASH_BUCKET) |

**Returns**

address structure with data where the bucket is stored

**5.32.2.11 AK\_insert\_in\_hash\_index()**

```
void AK_insert_in_hash_index (
    char * indexName,
    int hashValue,
    struct_add * add )
```

Function for inserting record in hash bucket.

**Author**

Mislav Čakarić

**Parameters**

|                  |   |
|------------------|---|
| <i>indexName</i> | name of index   |
| <i>hashValue</i> | hash value of record that is being inserted                 |
| <i>add</i>       | address structure with data where the hash bucket is stored |

**Returns**

No return value

**5.32.2.12 Ak\_update\_bucket\_in\_block()**

```
void Ak_update_bucket_in_block (
    struct_add * add,
    char * data )
```

Function for update bucket in block.

**Author**

Mislav Čakarić

## Parameters

|             |  |
|-------------|--|
| <i>add</i>  | address of where the bucket is stored  |
| <i>data</i> | content of bucket stored in char array |

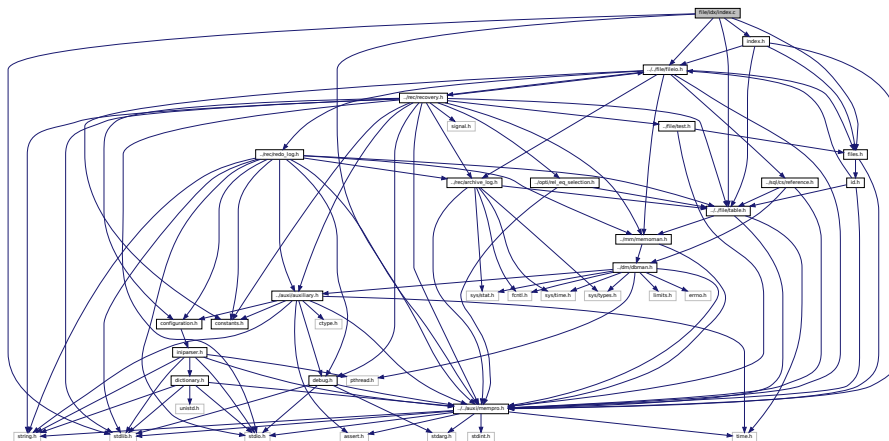
## Returns

No return value

## 5.33 file/idx/index.c File Reference

```
#include "index.h"
#include <stdlib.h>
#include "../aux/mempro.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../file/files.h"
```

Include dependency graph for index.c:



## Functions

- void [Ak\\_InitializelistAd](#) ([list\\_ad](#) \*L)  
*Function for initializing linked list.*
- [element\\_ad](#) [Ak\\_Get\\_First\\_elementAd](#) ([list\\_ad](#) \*L)  
*Function for finding first node of linked list.*
- [element\\_ad](#) [Ak\\_Get\\_Last\\_elementAd](#) ([list\\_ad](#) \*L)  
*Function for finding last node of linked list.*
- [element\\_ad](#) [Ak\\_Get\\_Next\\_elementAd](#) ([element\\_ad](#) Currentelement\_op)  
*Function for finding the next node of a node in linked list.*
- [element\\_ad](#) [Ak\\_Get\\_Previus\\_elementAd](#) ([element\\_ad](#) Currentelement\_op, [element\\_ad](#) L)  
*Function for finding the previous node of a node in linked list.*
- int [Ak\\_Get\\_Position\\_Of\\_elementAd](#) ([element\\_ad](#) Searchedelement\_op, [list\\_ad](#) \*L)  
*Function for finding the position of a node in linked list.*
- void [Ak\\_Delete\\_elementAd](#) ([element\\_ad](#) Deletedelement\_op, [list\\_ad](#) \*L)

- Function for deleting a node in linked list.*

  - void [Ak\\_Delete\\_All\\_elementsAd](#) ([list\\_ad](#) \*L)

*Function for deleting all nodes in linked list.*
- void [Ak\\_Insert\\_NewelementAd](#) (int addBlock, int indexTd, char \*attName, [element\\_ad](#) elementBefore)

*Function for inserting a new element into linked list.*
- int [AK\\_num\\_index\\_attr](#) (char \*indexTblName)

*Function for getting number of elements in index table.*
- int [AK\\_get\\_index\\_num\\_records](#) (char \*indexTblName)

*Determine number of rows in the table.*
- struct [list\\_node](#) \* [AK\\_get\\_index\\_tuple](#) (int row, int column, char \*indexTblName)

*Function that gets value in some row and column.*
- int [AK\\_index\\_table\\_exist](#) (char \*indexTblName)

*Function examines whether there is a table with the name "tblName" in the system catalog (AK\_relation)*
- [AK\\_header](#) \* [AK\\_get\\_index\\_header](#) (char \*indexTblName)

*Function that gets index table header.*
- void [AK\\_print\\_index\\_table](#) (char \*indexTblName)

*Function for printing index table.*
- void [AK\\_index\\_test](#) ()

*Test funtion for index structures(list) and printing table.*

### 5.33.1 Detailed Description

Provides functions for indexes

### 5.33.2 Function Documentation

#### 5.33.2.1 Ak\_Delete\_All\_elementsAd()

```
void Ak_Delete_All_elementsAd (
    list\_ad * L )
```

Function for deleting all nodes in linked list.

#### Author

Unknown

#### Parameters

|                   |           |
|-------------------|-----------|
| <a href="#">L</a> | list head |
|-------------------|-----------|

#### Returns

No return value

#### 5.33.2.2 Ak\_Delete\_elementAd()

```
void Ak_Delete_elementAd (
    element_ad Deletedelement_op,
    list_ad * L )
```

Function for deleting a node in linked list.

##### Author

Unknown

##### Parameters

|                          |                             |
|--------------------------|-----------------------------|
| <i>Deletedelement_op</i> | - address of node to delete |
| <i>list_ad</i>           | *L - list head              |

##### Returns

No return value

#### 5.33.2.3 Ak\_Get\_First\_elementAd()

```
element_ad Ak_Get_First_elementAd (
    list_ad * L )
```

Function for finding first node of linked list.

##### Author

Unknown

##### Parameters

|                |                     |
|----------------|---------------------|
| <i>list_ad</i> | *L linked list head |
|----------------|---------------------|

##### Returns

Address of first node

#### 5.33.2.4 AK\_get\_index\_header()

```
AK_header* AK_get_index_header (
    char * indexTblName )
```

Function that getts index table header.

**Author**

Matija Šestak, modified for indexes by Lovro Predovan

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. else read the first block
4. allocate array
5. copy table header to the array

**Parameters**

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

**Returns**

array of table header

**5.33.2.5 AK\_get\_index\_num\_records()**

```
int AK_get_index_num_records (
    char * indexTblName )
```

Determine number of rows in the table.

**Author**

Matija Šestak, modified for indexes by Lovro Predovan

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. For each extent from table
4. For each block in the extent
5. Get a block
6. Exit if there is no records in block
7. Count tuples in block
8. Return the number of tuples divided by number of attributes

**Parameters**

|                   |            |
|-------------------|------------|
| <i>*tableName</i> | table name |
|-------------------|------------|

**Returns**

number of rows in the table



#### 5.33.2.6 AK\_get\_index\_tuple()

```
struct list_node * AK_get_index_tuple (
    int row,
    int column,
    char * indexTblName )
```

Function that gets value in some row and column.

##### Author

Matija Šestak, modified for indexes by Lovro Predovan

##### Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>row</i>      | zero-based row index    |
| <i>column</i>   | zero-based column index |
| <i>*tblName</i> | table name              |

##### Returns

value in the list

#### 5.33.2.7 Ak\_Get\_Last\_elementAd()

```
element_ad Ak_Get_Last_elementAd (
    list_ad * L )
```

Function for finding last node of linked list.

##### Author

Unknown

##### Parameters

|                |                     |
|----------------|---------------------|
| <i>list_ad</i> | *L linked list head |
|----------------|---------------------|

##### Returns

Address of last node or 0 if list is empty

#### 5.33.2.8 Ak\_Get\_Next\_elementAd()

```
element_ad Ak_Get_Next_elementAd (
    element_ad Currentelement_op )
```

Function for finding the next node of a node in linked list.

**Author**

Unknown

**Parameters**

|                          |                         |
|--------------------------|-------------------------|
| <i>Currentelement_op</i> | address of current node |
|--------------------------|-------------------------|

**Returns**

Address of next node or 0 if current node is last in list

**5.33.2.9 Ak\_Get\_Position\_Of\_elementAd()**

```
int Ak_Get_Position_Of_elementAd (  
    element_ad Searchedelement_op,  
    list_ad * L )
```

Function for finding the position of a node in linked list.

**Author**

Unknown

**Parameters**

|                           |                         |
|---------------------------|-------------------------|
| <i>Searchedelement_op</i> | address of current note |
| <i>*L</i>                 | linked list head        |

**Returns**

Integer value of current node's order in the list

**5.33.2.10 Ak\_Get\_Previous\_elementAd()**

```
element_ad Ak_Get_Previous_elementAd (  
    element_ad Currentelement_op,  
    element_ad L )
```

Function for finding the previous node of a node in linked list.

**Author**

Unknown

**Parameters**

|                          |                         |
|--------------------------|-------------------------|
| <i>Currentelement_op</i> | Address of current node |
| <i>L</i>                 | previous element        |

**Returns**

Address of previous node or 0 if the current node is the head or the list is empty

**5.33.2.11 AK\_index\_table\_exist()**

```
int AK_index_table_exist (
    char * indexTblName )
```

Function examines whether there is a table with the name "tblName" in the system catalog (AK\_relation)

**Author**

Matija Šestak, modified for indexes by Lovro Predovan

**Parameters**

|                |            |
|----------------|------------|
| <i>tblName</i> | table name |
|----------------|------------|

**Returns**

returns 1 if table exist or returns 0 if table does not exist

**5.33.2.12 AK\_index\_test()**

```
void AK_index_test ( )
```

Test funtion for index structures(list) and printing table.

**Author**

Lovro Predovan

**Returns**

No return value

### 5.33.2.13 Ak\_InitializelistAd()

```
void Ak_InitializelistAd (
    list_ad * L )
```

Function for initalizing linked list.

#### Author

Unknown

#### Parameters

|                |                     |
|----------------|---------------------|
| <i>list_ad</i> | *L linked list head |
|----------------|---------------------|

#### Returns

No return value

### 5.33.2.14 Ak\_Insert\_NewelementAd()

```
void Ak_Insert_NewelementAd (
    int addBlock,
    int indexTd,
    char * attName,
    element_ad elementBefore )
```

Function for inserting a new element into linked list.

#### Author

Unknown

#### Parameters

|                      |   |
|----------------------|---|
| <i>addBlock</i>      | address block   |
| <i>indexTd</i>       | index table destination                                       |
| <i>*attname</i>      | attribute name  |
| <i>elementBefore</i> | address of the node after which the new node will be inserted |

#### Returns

No return value

## 5.33.2.15 AK\_num\_index\_attr()

```
int AK_num_index_attr (
    char * indexTblName )
```

Function for getting number of elements in index table.

## Author

Lovro Predovan

## Parameters

|              |            |
|--------------|------------|
| <i>index</i> | table name |
|--------------|------------|

## Returns

No return value

## 5.33.2.16 AK\_print\_index\_table()

```
void AK_print_index_table (
    char * indexTblName )
```

Function for printing index table.

## Author

Matija Šestak, modified for indexes by Lovro Predovan

## Parameters

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

## Returns

No return value

## 5.34 file/idx/index.h File Reference

```
#include "../aux/mempro.h"
#include "../file/table.h"
#include "../file/fileio.h"
```



## Functions

- int [AK\\_index\\_table\\_exist](#) (char \*indexTblName)  
*Function examines whether there is a table with the name "tblName" in the system catalog (AK\_relation)*
- void [AK\\_print\\_index\\_table](#) (char \*indexTblName)  
*Function for printing index table.*
- struct [list\\_node](#) \* [AK\\_get\\_index\\_tuple](#) (int row, int column, char \*indexTblName)  
*Function that gets value in some row and column.*
- int [AK\\_get\\_index\\_num\\_records](#) (char \*indexTblName)  
*Determine number of rows in the table.*
- int [AK\\_num\\_index\\_attr](#) (char \*indexTblName)  
*Function for getting number of elements in index table.*
- void [Ak\\_InitializelistAd](#) ([list\\_ad](#) \*L)  
*Function for initializing linked list.*
- [element\\_ad](#) [Ak\\_Get\\_First\\_elementAd](#) ([list\\_ad](#) \*L)  
*Function for finding first node of linked list.*
- [element\\_ad](#) [Ak\\_Get\\_Last\\_elementAd](#) ([list\\_ad](#) \*L)  
*Function for finding last node of linked list.*
- [element\\_ad](#) [Ak\\_Get\\_Next\\_elementAd](#) ([element\\_ad](#) Currentelement\_op)  
*Function for finding the next node of a node in linked list.*
- [element\\_ad](#) [Ak\\_Get\\_Previous\\_elementAd](#) ([element\\_ad](#) Currentelement\_op, [element\\_ad](#) L)  
*Function for finding the previous node of a node in linked list.*
- int [Ak\\_Get\\_Position\\_Of\\_elementAd](#) ([element\\_ad](#) Searchedelement\_op, [list\\_ad](#) \*L)  
*Function for finding the position of a node in linked list.*
- void [Ak\\_Delete\\_elementAd](#) ([element\\_ad](#) Deletedelement\_op, [list\\_ad](#) \*L)  
*Function for deleting a node in linked list.*
- void [Ak\\_Delete\\_All\\_elementsAd](#) ([list\\_ad](#) \*L)  
*Function for deleting all nodes in linked list.*
- void [Ak\\_Insert\\_NewelementAd](#) (int addBlock, int indexTd, char \*attName, [element\\_ad](#) elementBefore)  
*Function for inserting a new element into linked list.*
- void [AK\\_index\\_test](#) ()  
*Test funtion for index structures(list) and printing table.*

### 5.34.1 Detailed Description

Header file that provides data structures for bitmap index

### 5.34.2 Function Documentation

#### 5.34.2.1 Ak\_Delete\_All\_elementsAd()

```
void Ak_Delete_All_elementsAd (
    list\_ad * L )
```

Function for deleting all nodes in linked list.

#### Author

Unknown

**Parameters**

|          |           |
|----------|-----------|
| <i>L</i> | list head |
|----------|-----------|

**Returns**

No return value

**5.34.2.2 Ak\_Delete\_elementAd()**

```
void Ak_Delete_elementAd (
    element_ad Deletedelement_op,
    list_ad * L )
```

Function for deleting a node in linked list.

**Author**

Unknown

**Parameters**

|                          |                             |
|--------------------------|-----------------------------|
| <i>Deletedelement_op</i> | - address of node to delete |
| <i>list_ad</i>           | *L - list head              |

**Returns**

No return value

**5.34.2.3 Ak\_Get\_First\_elementAd()**

```
element_ad Ak_Get_First_elementAd (
    list_ad * L )
```

Function for finding first node of linked list.

**Author**

Unknown

**Parameters**

|                |                     |
|----------------|---------------------|
| <i>list_ad</i> | *L linked list head |
|----------------|---------------------|



**Returns**

Address of first node

**5.34.2.4 AK\_get\_index\_num\_records()**

```
int AK_get_index_num_records (
    char * indexTblName )
```

Determine number of rows in the table.

**Author**

Matija Šestak, modified for indexes by Lovro Predovan

1. Read addresses of extents
2. If there is no extents in the table, return -1
3. For each extent from table
4. For each block in the extent
5. Get a block
6. Exit if there is no records in block
7. Count tuples in block
8. Return the number of tuples divided by number of attributes

**Parameters**

|                   |            |
|-------------------|------------|
| <i>*tableName</i> | table name |
|-------------------|------------|

**Returns**

number of rows in the table

**5.34.2.5 AK\_get\_index\_tuple()**

```
struct list_node* AK_get_index_tuple (
    int row,
    int column,
    char * indexTblName )
```

Function that gets value in some row and column.

**Author**

Matija Šestak, modified for indexes by Lovro Predovan

**Parameters**

|                 |                         |
|-----------------|-------------------------|
| <i>row</i>      | zero-based row index    |
| <i>column</i>   | zero-based column index |
| <i>*tblName</i> | table name              |

**Returns**

value in the list

**5.34.2.6 Ak\_Get\_Last\_elementAd()**

```
element_ad Ak_Get_Last_elementAd (
    list_ad * L )
```

Function for finding last node of linked list.

**Author**

Unknown

**Parameters**

|                |                     |
|----------------|---------------------|
| <i>list_ad</i> | *L linked list head |
|----------------|---------------------|

**Returns**

Address of last node or 0 if list is empty

**5.34.2.7 Ak\_Get\_Next\_elementAd()**

```
element_ad Ak_Get_Next_elementAd (
    element_ad Currentelement_op )
```

Function for finding the next node of a node in linked list.

**Author**

Unknown

**Parameters**

|                          |                         |
|--------------------------|-------------------------|
| <i>Currentelement_op</i> | address of current node |
|--------------------------|-------------------------|

**Returns**

Address of next node or 0 if current node is last in list

**5.34.2.8 Ak\_Get\_Position\_Of\_elementAd()**

```
int Ak_Get_Position_Of_elementAd (
    element_ad Searchedelement_op,
    list_ad * L )
```

Function for finding the position of a node in linked list.

**Author**

Unknown

**Parameters**

|                           |                         |
|---------------------------|-------------------------|
| <i>Searchedelement_op</i> | address of current note |
| <i>*L</i>                 | linked list head        |

**Returns**

Integer value of current node's order in the list

**5.34.2.9 Ak\_Get\_Previous\_elementAd()**

```
element_ad Ak_Get_Previous_elementAd (
    element_ad Currentelement_op,
    element_ad L )
```

Function for finding the previous node of a node in linked list.

**Author**

Unknown

**Parameters**

|                          |                         |
|--------------------------|-------------------------|
| <i>Currentelement_op</i> | Address of current node |
| <i>L</i>                 | previous element        |

**Returns**

Address of previous node or 0 if the current node is the head or the list is empty

**5.34.2.10 AK\_index\_table\_exist()**

```
int AK_index_table_exist (
    char * indexTblName )
```

Function examines whether there is a table with the name "tblName" in the system catalog (AK\_relation)

**Author**

Matija Šestak, modified for indexes by Lovro Predovan

**Parameters**

|                |            |
|----------------|------------|
| <i>tblName</i> | table name |
|----------------|------------|

**Returns**

returns 1 if table exist or returns 0 if table does not exist

**5.34.2.11 AK\_index\_test()**

```
void AK_index_test ( )
```

Test funtion for index structures(list) and printing table.

**Author**

Lovro Predovan

**Returns**

No return value

**5.34.2.12 Ak\_InitializelistAd()**

```
void Ak_InitializelistAd (
    list_ad * L )
```

Function for initalizing linked list.

**Author**

Unknown

## Parameters

|                |                     |
|----------------|---------------------|
| <i>list_ad</i> | *L linked list head |
|----------------|---------------------|

## Returns

No return value

## 5.34.2.13 Ak\_Insert\_NewelementAd()

```
void Ak_Insert_NewelementAd (
    int addBlock,
    int indexTd,
    char * attName,
    element_ad elementBefore )
```

Function for inserting a new element into linked list.

## Author

Unknown

## Parameters

|                      |   |
|----------------------|---|
| <i>addBlock</i>      | address block   |
| <i>indexTd</i>       | index table destination                                       |
| <i>*attname</i>      | attribute name  |
| <i>elementBefore</i> | address of the node after which the new node will be inserted |

## Returns

No return value

## 5.34.2.14 AK\_num\_index\_attr()

```
int AK_num_index_attr (
    char * indexTblName )
```

Function for getting number of elements in index table.

## Author

Lovro Predovan



## Functions

- int [AK\\_sequence\\_add](#) (char \*name, int start\_value, int increment, int max\_value, int min\_value, int cycle)  
*Function for adding sequence.*
- int [AK\\_sequence\\_remove](#) (char \*name)  
*Function for removing sequence.*
- int [AK\\_sequence\\_current\\_value](#) (char \*name)  
*Function returns the current value of the sequence.*
- int [AK\\_sequence\\_next\\_value](#) (char \*name)  
*Function returns the next value of the sequence and writes it in a system table as current value.*
- int [AK\\_sequence\\_get\\_id](#) (char \*name)  
*Function gets sequence id.*
- int [AK\\_sequence\\_rename](#) (char \*old\_name, char \*new\_name)  
*Function renames the sequence.*
- int [AK\\_sequence\\_modify](#) (char \*name, int start\_value, int increment, int max\_value, int min\_value, int cycle)  
*Function for modifying sequence.*
- void [AK\\_sequence\\_test](#) ()  
*Function for sequences testing.*

### 5.35.1 Detailed Description

Provides functions for sequences

### 5.35.2 Function Documentation

#### 5.35.2.1 AK\_sequence\_add()

```
int AK_sequence_add (
    char * name,
    int start_value,
    int increment,
    int max_value,
    int min_value,
    int cycle )
```

Function for adding sequence.

#### Author

Boris Kišić

#### Parameters

|                    |  |
|--------------------|--|
| <i>name</i>        | name of the sequence                     |
| <i>start_value</i> | start value of the sequence              |
| <i>increment</i>   | increment of the sequence                |
| <i>max_value</i>   | maximum value of the sequence            |
| <i>min_value</i>   | minimum value of the sequence            |
| <i>cycle</i>       | 0:non-cyclic sequence, 1:cyclic sequence |

**Returns**

sequence\_id or EXIT\_ERROR

**5.35.2.2 AK\_sequence\_current\_value()**

```
int AK_sequence_current_value (
    char * name )
```

Function returns the current value of the sequence.

**Author**

Boris Kišić

**Parameters**

|             |                      |
|-------------|----------------------|
| <i>name</i> | name of the sequence |
|-------------|----------------------|

**Returns**

current\_value or EXIT\_ERROR

**5.35.2.3 AK\_sequence\_get\_id()**

```
int AK_sequence_get_id (
    char * name )
```

Function gets sequence id.

**Author**

Ljubo Barać

**Parameters**

|             |                      |
|-------------|----------------------|
| <i>name</i> | Name of the sequence |
|-------------|----------------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR



#### 5.35.2.4 AK\_sequence\_modify()

```
int AK_sequence_modify (
    char * name,
    int start_value,
    int increment,
    int max_value,
    int min_value,
    int cycle )
```

Function for modifying sequence.

##### Author

Boris Kišić fixed by Ljubo Barać

##### Parameters

|                    |  |
|--------------------|--|
| <i>name</i>        | Name of the sequence                     |
| <i>start_value</i> | start value of the sequence              |
| <i>increment</i>   | increment of the sequence                |
| <i>max_value</i>   | maximum value of the sequence            |
| <i>min_value</i>   | minimum value of the sequence            |
| <i>cycle</i>       | 0:non-cyclic sequence, 1:cyclic sequence |

##### Returns

EXIT\_SUCCESS or EXIT\_ERROR

#### 5.35.2.5 AK\_sequence\_next\_value()

```
int AK_sequence_next_value (
    char * name )
```

Function returns the next value of the sequence and writes it in a system table as current value.

##### Author

Boris Kišić

##### Parameters

|             |                      |
|-------------|----------------------|
| <i>name</i> | name of the sequence |
|-------------|----------------------|

##### Returns

next\_value or EXIT\_ERROR

#### 5.35.2.6 AK\_sequence\_remove()

```
int AK_sequence_remove (
    char * name )
```

Function for removing sequence.

##### Author

Boris Kišić

##### Parameters

|             |                      |
|-------------|----------------------|
| <i>name</i> | name of the sequence |
|-------------|----------------------|

##### Returns

EXIT\_SUCCESS or EXIT\_ERROR

#### 5.35.2.7 AK\_sequence\_rename()

```
int AK_sequence_rename (
    char * old_name,
    char * new_name )
```

Function renames the sequence.

##### Author

Boris Kišić

##### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>old_name</i> | Name of the sequence to be renamed |
| <i>new_name</i> | New name of the sequence           |

##### Returns

EXIT\_SUCCESS or EXIT\_ERROR

### 5.35.2.8 AK\_sequence\_test()

```
void AK_sequence_test ( )
```

Function for sequences testing.

**Author**

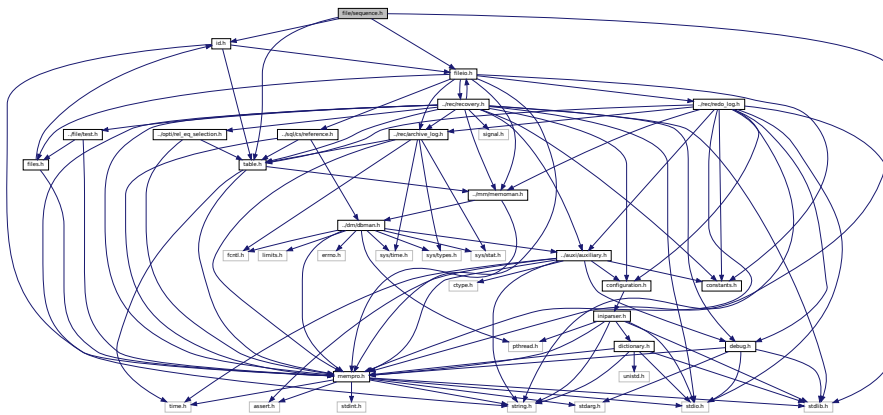
Boris Kišić fixed by Ljubo Barać

## Returns

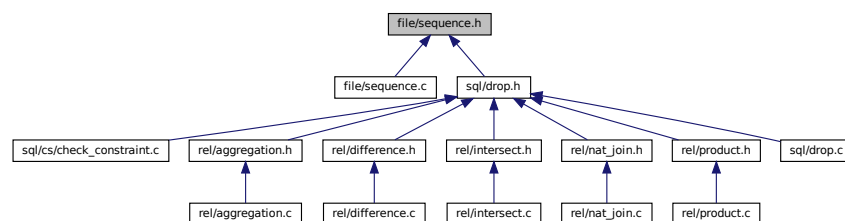
No return value

### 5.36 file/sequence.h File Reference

```
#include "table.h"
#include "id.h"
#include "fileio.h"
#include "../auxi/mempro.h"
Include dependency graph for sequence.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- int [AK\\_sequence\\_add](#) (char \*name, int start\_value, int increment, int max\_value, int min\_value, int cycle)  
*Function for adding sequence.*
- int [AK\\_sequence\\_remove](#) (char \*name)  
*Function for removing sequence.*
- int [AK\\_sequence\\_current\\_value](#) (char \*name)  
*Function returns the current value of the sequence.*
- int [AK\\_sequence\\_next\\_value](#) (char \*name)  
*Function returns the next value of the sequence and writes it in a system table as current value.*
- int [AK\\_sequence\\_rename](#) (char \*old\_name, char \*new\_name)  
*Function renames the sequence.*
- int [AK\\_sequence\\_modify](#) (char \*name, int start\_value, int increment, int max\_value, int min\_value, int cycle)  
*Function for modifying sequence.*
- int [AK\\_sequence\\_get\\_id](#) (char \*name)  
*Function gets sequence id.*
- void [AK\\_sequence\\_test](#) ()  
*Function for sequences testing.*

### 5.36.1 Detailed Description

Header file that provides data structures for sequences

### 5.36.2 Function Documentation

#### 5.36.2.1 AK\_sequence\_add()

```
int AK_sequence_add (
    char * name,
    int start_value,
    int increment,
    int max_value,
    int min_value,
    int cycle )
```

Function for adding sequence.

#### Author

Boris Kišić

#### Parameters

|                    |  |
|--------------------|--|
| <i>name</i>        | name of the sequence                     |
| <i>start_value</i> | start value of the sequence              |
| <i>increment</i>   | increment of the sequence                |
| <i>max_value</i>   | maximum value of the sequence            |
| <i>min_value</i>   | minimum value of the sequence            |
| <i>cycle</i>       | 0:non-cyclic sequence, 1:cyclic sequence |

**Returns**

sequence\_id or EXIT\_ERROR

**5.36.2.2 AK\_sequence\_current\_value()**

```
int AK_sequence_current_value (
    char * name )
```

Function returns the current value of the sequence.

**Author**

Boris Kišić

**Parameters**

|             |                      |
|-------------|----------------------|
| <i>name</i> | name of the sequence |
|-------------|----------------------|

**Returns**

current\_value or EXIT\_ERROR

**5.36.2.3 AK\_sequence\_get\_id()**

```
int AK_sequence_get_id (
    char * name )
```

Function gets sequence id.

**Author**

Ljubo Barać

**Parameters**

|             |                      |
|-------------|----------------------|
| <i>name</i> | Name of the sequence |
|-------------|----------------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

#### 5.36.2.4 AK\_sequence\_modify()

```
int AK_sequence_modify (
    char * name,
    int start_value,
    int increment,
    int max_value,
    int min_value,
    int cycle )
```

Function for modifying sequence.

##### Author

Boris Kišić fixed by Ljubo Barać

##### Parameters

|                    |  |
|--------------------|--|
| <i>name</i>        | Name of the sequence                     |
| <i>start_value</i> | start value of the sequence              |
| <i>increment</i>   | increment of the sequence                |
| <i>max_value</i>   | maximum value of the sequence            |
| <i>min_value</i>   | minimum value of the sequence            |
| <i>cycle</i>       | 0:non-cyclic sequence, 1:cyclic sequence |

##### Returns

EXIT\_SUCCESS or EXIT\_ERROR

#### 5.36.2.5 AK\_sequence\_next\_value()

```
int AK_sequence_next_value (
    char * name )
```

Function returns the next value of the sequence and writes it in a system table as current value.

##### Author

Boris Kišić

##### Parameters

|             |                      |
|-------------|----------------------|
| <i>name</i> | name of the sequence |
|-------------|----------------------|

##### Returns

next\_value or EXIT\_ERROR

### 5.36.2.6 AK\_sequence\_remove()

```
int AK_sequence_remove (
    char * name )
```

Function for removing sequence.

#### Author

Boris Kišić

#### Parameters

|             |                      |
|-------------|----------------------|
| <i>name</i> | name of the sequence |
|-------------|----------------------|

#### Returns

EXIT\_SUCCESS or EXIT\_ERROR

### 5.36.2.7 AK\_sequence\_rename()

```
int AK_sequence_rename (
    char * old_name,
    char * new_name )
```

Function renames the sequence.

#### Author

Boris Kišić

#### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>old_name</i> | Name of the sequence to be renamed |
| <i>new_name</i> | New name of the sequence           |

#### Returns

EXIT\_SUCCESS or EXIT\_ERROR

### 5.36.2.8 AK\_sequence\_test()

```
void AK_sequence_test ( )
```

Function for sequences testing.

#### Author

Boris Kišić fixed by Ljubo Barać

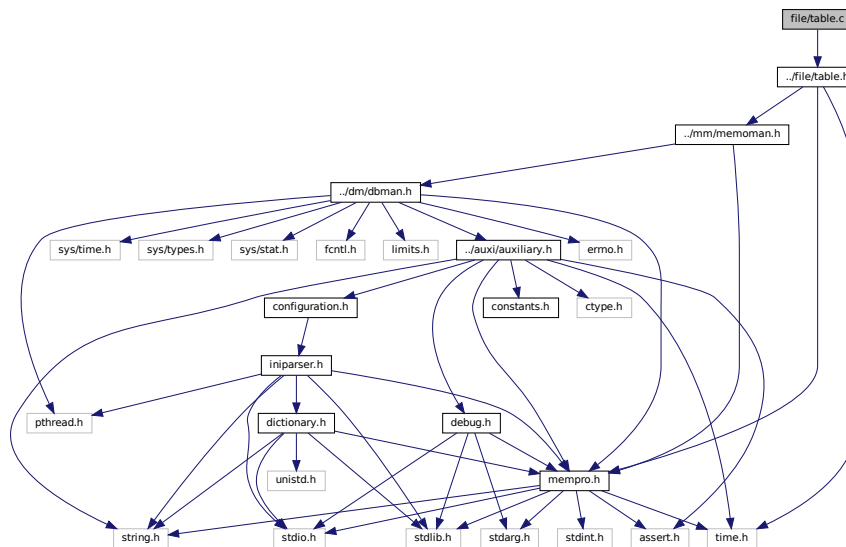
#### Returns

No return value

## 5.37 file/table.c File Reference

```
#include "../file/table.h"
```

Include dependency graph for table.c:



## Functions

- [AK\\_create\\_table\\_parameter](#) \* **AK\_create\_create\_table\_parameter** (int type, char \*name)
- void **AK\_create\_table** (char \*tblName, [AK\\_create\\_table\\_parameter](#) \*parameters, int attribute\_count)
- int [AK\\_num\\_attr](#) (char \*tblName)  
Determine the number of attributes in the table.
- int [AK\\_get\\_num\\_records](#) (char \*tblName)  
Determine number of rows in the table.
- [AK\\_header](#) \* [AK\\_get\\_header](#) (char \*tblName)  
Function that gets table header.
- char \* [AK\\_get\\_attr\\_name](#) (char \*tblName, int index)



- Function that gets attribute name for some zero-based index.*
- int [AK\\_get\\_attr\\_index](#) (char \*tblName, char \*attrName)
- Function that gets zero-based index for attribute.*
- struct [list\\_node](#) \* [AK\\_get\\_column](#) (int num, char \*tblName)
- Function that gets all values in some column and put on the list.*
- struct [list\\_node](#) \* [AK\\_get\\_row](#) (int num, char \*tblName)
- Function that gets all values in some row and put on the list.*
- struct [list\\_node](#) \* [AK\\_get\\_tuple](#) (int row, int column, char \*tblName)
- Function that gets value in some row and column.*
- char \* [AK\\_tuple\\_to\\_string](#) (struct [list\\_node](#) \*tuple)
- Function that converts tuple value to string.*
- void [AK\\_print\\_row\\_spacer](#) (int col\_len[], int length)
- Function that prints row spacer.*
- void [AK\\_print\\_row](#) (int col\_len[], struct [list\\_node](#) \*row)
- Function that prints table row.*
- int [AK\\_table\\_exist](#) (char \*tblName)
- Function examines whether there is a table with the name "tblName" in the system catalog (AK\_relation)*
- void [AK\\_print\\_table](#) (char \*tblName)
- Function for printing table.*
- void [AK\\_print\\_row\\_spacer\\_to\\_file](#) (int col\_len[], int length)
- Function that prints row spacer update by Luka Rajcevic.*
- char \* [get\\_row\\_attr\\_data](#) (int column, struct [list\\_node](#) \*node)
- Function that returns value of attribute from row.*
- void [AK\\_print\\_row\\_to\\_file](#) (int col\_len[], struct [list\\_node](#) \*row)
- Function that prints table row update by Luka Rajcevic.*
- void [AK\\_print\\_table\\_to\\_file](#) (char \*tblName)
- Function for printing table.*
- int [AK\\_table\\_empty](#) (char \*tblName)
- Function that check whether table is empty.*
- int [AK\\_get\\_table\\_obj\\_id](#) (char \*table)
- Function that gets obj\_id of named table from AK\_relation system table.*
- int [AK\\_check\\_tables\\_scheme](#) ([AK\\_mem\\_block](#) \*tbl1\_temp\_block, [AK\\_mem\\_block](#) \*tbl2\_temp\_block, char \*operator\_name)
- Function to check if tables have the same relation schema.*
- int [AK\\_rename](#) (char \*old\_table\_name, char \*old\_attr, char \*new\_table\_name, char \*new\_attr)
- Function for renaming table and/or attribute in table (moved from rename.c)*
- void [AK\\_table\\_test](#) ()
- Function for testing table abstraction.*
- void [AK\\_op\\_rename\\_test](#) ()
- Function for rename operator testing (moved from rename.c)*

### 5.37.1 Detailed Description

Provides functions for table abstraction

### 5.37.2 Function Documentation

### 5.37.2.1 AK\_check\_tables\_scheme()

```
int AK_check_tables_scheme (
    AK_mem_block * tbl1_temp_block,
    AK_mem_block * tbl2_temp_block,
    char * operator_name )
```

Function to check if tables have the same relation schema.

#### Author

Dino Laktašić, abstracted from [difference.c](#) for use in [difference.c](#), [intersect.c](#) and [union.c](#) by Tomislav Mikulček

#### Parameters

|                        |   |
|------------------------|---|
| <i>tbl1_temp_block</i> | first cache block of the first table                    |
| <i>tbl2_temp_block</i> | first cache block of the second table                   |
| <i>operator_name</i>   | the name of operator, used for displaying error message |

#### Returns

if success returns num of attributes in schema, else returns EXIT\_ERROR

### 5.37.2.2 AK\_get\_attr\_index()

```
int AK_get_attr_index (
    char * tblName,
    char * attrName )
```

Function that gets zero-based index for attribute.

#### Author

Matija Šestak.

#### Parameters

|                  |                |
|------------------|----------------|
| <i>*tblName</i>  | table name     |
| <i>*attrName</i> | attribute name |

#### Returns

zero-based index

### 5.37.2.3 AK\_get\_attr\_name()

```
char* AK_get_attr_name (
    char * tblName,
    int index )
```

Function that gets attribute name for some zero-based index.

#### Author

Matija Šestak.

#### Parameters

|                 |                  |
|-----------------|------------------|
| <i>*tblName</i> | table name       |
| <i>index</i>    | zero-based index |

#### Returns

attribute name

### 5.37.2.4 AK\_get\_column()

```
struct list_node* AK_get_column (
    int num,
    char * tblName )
```

Function that gets all values in some column and put on the list.

#### Author

Matija Šestak.

#### Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>num</i>      | zero-based column index |
| <i>*tblName</i> | table name              |

#### Returns

column values list

### 5.37.2.5 AK\_get\_header()

```
AK_header* AK_get_header (
    char * tblName )
```

Function that getts table header.

#### Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return 0
3. else read the first block
4. allocate array
5. copy table header to the array

#### Parameters

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

#### Returns

array of table header

#### 5.37.2.6 AK\_get\_num\_records()

```
int AK_get_num_records (
    char * tblName )
```

Determine number of rows in the table.

#### Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return EXIT\_WARNING
3. For each extent from table
4. For each block in the extent
5. Get a block
6. Exit if there is no records in block
7. Count tuples in block
8. Return the number of tuples divided by number of attributes

#### Parameters

|                   |            |
|-------------------|------------|
| <i>*tableName</i> | table name |
|-------------------|------------|

#### Returns

number of rows in the table

### 5.37.2.7 AK\_get\_row()

```
struct list_node* AK_get_row (
    int num,
    char * tblName )
```

Function that gets all values in some row and put on the list.

#### Author

Markus Schatten, Matija Šestak.

#### Parameters

|            |                      |
|------------|----------------------|
| <i>num</i> | zero-based row index |
| *          | tblName table name   |

#### Returns

row values list

### 5.37.2.8 AK\_get\_table\_obj\_id()

```
int AK_get_table_obj_id (
    char * table )
```

Function that gets obj\_id of named table from AK\_relation system table.

#### Author

Dejan Frankovic

#### Parameters

|               |            |
|---------------|------------|
| <i>*table</i> | table name |
|---------------|------------|

#### Returns

obj\_id of the table or EXIT\_ERROR if there is no table with that name

### 5.37.2.9 AK\_get\_tuple()

```
struct list_node* AK_get_tuple (
    int row,
    int column,
    char * tblName )
```

Function that gets value in some row and column.

#### Author

Matija Šestak.

#### Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>row</i>      | zero-based row index    |
| <i>column</i>   | zero-based column index |
| <i>*tblName</i> | table name              |

#### Returns

value in the list

### 5.37.2.10 AK\_num\_attr()

```
int AK_num_attr (
    char * tblName )
```

Determine the number of attributes in the table.

#### Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return EXIT\_WARNING
3. else read the first block
4. while header tuple exists in the block, increment num\_attr

#### Parameters

|   |                    |
|---|--------------------|
| * | tblName table name |
|---|--------------------|

#### Returns

number of attributes in the table

#### 5.37.2.11 AK\_op\_rename\_test()

```
void AK_op_rename_test ( )
```

Function for rename operator testing (moved from rename.c)

##### Author

Mislav Čakarić, edited by Ljubo Barać

##### Returns

No return value

#### 5.37.2.12 AK\_print\_row()

```
void AK_print_row (
    int col_len[],
    struct list_node * row )
```

Function that prints table row.

##### Author

Dino Laktašić

##### Parameters

|                  |   |
|------------------|---|
| <i>col_len[]</i> | array of max lengths for each attribute |
| <i>*row</i>      | list with row elements                  |

##### Returns

No return value

#### 5.37.2.13 AK\_print\_row\_spacer()

```
void AK_print_row_spacer (
    int col_len[],
    int length )
```

Function that prints row spacer.

##### Author

Dino Laktašić.

**Parameters**

|                  |                                     |
|------------------|-------------------------------------|
| <i>col_len[]</i> | max lengths for each attribute cell |
| <i>length</i>    | total table width                   |

**Returns**

printed row spacer

**5.37.2.14 AK\_print\_row\_spacer\_to\_file()**

```
void AK_print_row_spacer_to_file (
    int col_len[],
    int length )
```

Function that prints row spacer update by Luka Rajcevic.

**Author**

Dino Laktašić.

**Parameters**

|                  |                                     |
|------------------|-------------------------------------|
| <i>col_len[]</i> | max lengths for each attribute cell |
| <i>length</i>    | total table width                   |

**Returns**

printed row spacer

**5.37.2.15 AK\_print\_row\_to\_file()**

```
void AK_print_row_to_file (
    int col_len[],
    struct list_node * row )
```

Function that prints table row update by Luka Rajcevic.

**Author**

Dino Laktašić



**Parameters**

|                  |   |
|------------------|---|
| <i>col_len[]</i> | array of max lengths for each attribute |
| <i>*row</i>      | list with row elements                  |

**Returns**

No return value

**5.37.2.16 AK\_print\_table()**

```
void AK_print_table (
    char * tblName )
```

Function for printing table.

**Author**

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one)

**Parameters**

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

**Returns**

No return value

**5.37.2.17 AK\_print\_table\_to\_file()**

```
void AK_print_table_to_file (
    char * tblName )
```

Function for printing table.

**Author**

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one) update by Luka Rajcevic

**Parameters**

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

**Returns**

No return value update by Anto Tomaš (corrected the Ak\_DeleteAll\_L3 function)

**5.37.2.18 AK\_rename()**

```
int AK_rename (
    char * old_table_name,
    char * old_attr,
    char * new_table_name,
    char * new_attr )
```

Function for renaming table and/or attribute in table (moved from rename.c)

**Author**

Mislav Čakarić edited by Ljubo Barać

**Parameters**

|                       |                                      |
|-----------------------|--------------------------------------|
| <i>old_table_name</i> | old name of the table                |
| <i>new_table_name</i> | new name of the table                |
| <i>old_attr</i>       | name of the attribute to rename      |
| <i>new_attr</i>       | new name for the attribute to rename |

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**5.37.2.19 AK\_table\_empty()**

```
int AK_table_empty (
    char * tblName )
```

Function that check whether table is empty.

**Author**

Matija Šestak.

**Parameters**

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

**Returns**

true/false

**5.37.2.20 AK\_table\_exist()**

```
int AK_table_exist (
    char * tblName )
```

Function examines whether there is a table with the name "tblName" in the system catalog (AK\_relation)

**Author**

Jurica Hlevnjak

**Parameters**

|                |            |
|----------------|------------|
| <i>tblName</i> | table name |
|----------------|------------|

**Returns**

returns 1 if table exist or returns 0 if table does not exist

**5.37.2.21 AK\_table\_test()**

```
void AK_table_test ( )
```

Function for testing table abstraction.

**Author**

Unknown

**Returns**

No return value

by Ana-Marija Balen - added getRow function to the test

**5.37.2.22 AK\_tuple\_to\_string()**

```
char* AK_tuple_to_string (
    struct list_node * tuple )
```

Function that converts tuple value to string.

**Author**

Matija Šestak.

**Parameters**

|               |                   |
|---------------|-------------------|
| <i>*tuple</i> | tuple in the list |
|---------------|-------------------|

**Returns**

tuple value as a string

**5.37.2.23 get\_row\_attr\_data()**

```
char* get_row_attr_data (
    int column,
    struct list_node * node )
```

Function that returns value of attribute from row.

**Author**

Leon Palać

**Parameters**

|               |                           |
|---------------|---------------------------|
| <i>column</i> | index of column attribute |
| <i>*row</i>   | list with row elements    |

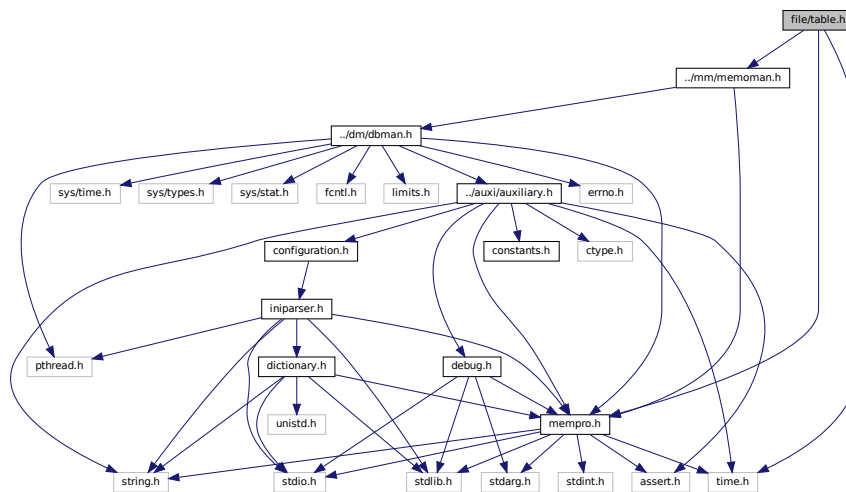
**Returns**

atribute data

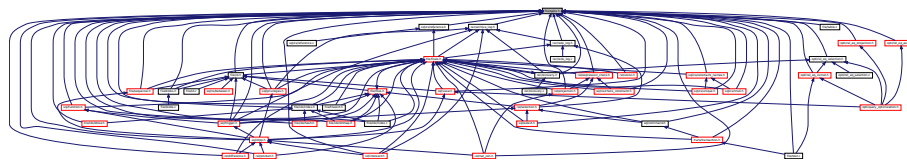
**5.38 file/table.h File Reference**

```
#include "../mm/memoman.h"
#include "../auxi/mempro.h"
#include <time.h>
```

Include dependency graph for table.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [AK\\_create\\_table\\_struct](#)

## Typedefs

- typedef struct [AK\\_create\\_table\\_struct](#) **AK\_create\_table\_parameter**

## Functions

- [AK\\_create\\_table\\_parameter](#) \* **AK\_create\_create\_table\_parameter** (int type, char \*name)
- void **AK\_create\_table** (char \*tblName, [AK\\_create\\_table\\_parameter](#) \*parameters, int attribute\_count)
- int [AK\\_num\\_attr](#) (char \*tblName)
  - Determine the number of attributes in the table.
- int [AK\\_get\\_num\\_records](#) (char \*tblName)
  - Determine number of rows in the table.
- [AK\\_header](#) \* [AK\\_get\\_header](#) (char \*tblName)
  - Function that gets table header.
- char \* [AK\\_get\\_attr\\_name](#) (char \*tblName, int index)
  - Function that gets attribute name for some zero-based index.
- int [AK\\_get\\_attr\\_index](#) (char \*tblName, char \*attrName)

- Function that gets zero-based index for attribute.*
- struct [list\\_node](#) \* [AK\\_get\\_column](#) (int num, char \*tblName)
- Function that gets all values in some column and put on the list.*
- struct [list\\_node](#) \* [AK\\_get\\_row](#) (int num, char \*tblName)
- Function that gets all values in some row and put on the list.*
- struct [list\\_node](#) \* [AK\\_get\\_tuple](#) (int row, int column, char \*tblName)
- Function that gets value in some row and column.*
- char \* [AK\\_tuple\\_to\\_string](#) (struct [list\\_node](#) \*tuple)
- Function that converts tuple value to string.*
- void [AK\\_print\\_row\\_spacer](#) (int col\_len[], int length)
- Function that prints row spacer.*
- void [AK\\_print\\_row](#) (int col\_len[], struct [list\\_node](#) \*row)
- Function that prints table row.*
- void [AK\\_print\\_table](#) (char \*tblName)
- Function for printing table.*
- void [AK\\_print\\_row\\_spacer\\_to\\_file](#) (int col\_len[], int length)
- Function that prints row spacer update by Luka Rajcevic.*
- void [AK\\_print\\_row\\_to\\_file](#) (int col\_len[], struct [list\\_node](#) \*row)
- Function that prints table row update by Luka Rajcevic.*
- void [AK\\_print\\_table\\_to\\_file](#) (char \*tblName)
- Function for printing table.*
- int [AK\\_table\\_empty](#) (char \*tblName)
- Function that check whether table is empty.*
- int [AK\\_get\\_table\\_obj\\_id](#) (char \*table)
- Function that gets obj\_id of named table from AK\_relation system table.*
- int [AK\\_check\\_tables\\_scheme](#) ([AK\\_mem\\_block](#) \*tbl1\_temp\_block, [AK\\_mem\\_block](#) \*tbl2\_temp\_block, char \*operator\_name)
- Function to check if tables have the same relation schema.*
- char \* [get\\_row\\_attr\\_data](#) (int column, struct [list\\_node](#) \*node)
- Function that returns value of attribute from row.*
- void [AK\\_table\\_test](#) ()
- Function for testing table abstraction.*
- int [AK\\_rename](#) (char \*old\_table\_name, char \*old\_attr, char \*new\_table\_name, char \*new\_attr)
- Function for renaming table and/or attribute in table (moved from rename.c)*
- void [AK\\_op\\_rename\\_test](#) ()
- Function for rename operator testing (moved from rename.c)*

### 5.38.1 Detailed Description

Header file that provides data structures for table abstraction

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor Boston, MA 02110-1301, USA

## 5.38.2 Function Documentation

### 5.38.2.1 AK\_check\_tables\_scheme()

```
int AK_check_tables_scheme (
    AK_mem_block * tbl1_temp_block,
    AK_mem_block * tbl2_temp_block,
    char * operator_name )
```

Function to check if tables have the same relation schema.

#### Author

Dino Laktašić, abstracted from [difference.c](#) for use in [difference.c](#), [intersect.c](#) and [union.c](#) by Tomislav Mikulček

#### Parameters

|                        |   |
|------------------------|---|
| <i>tbl1_temp_block</i> | first cache block of the first table                    |
| <i>tbl2_temp_block</i> | first cache block of the second table                   |
| <i>operator_name</i>   | the name of operator, used for displaying error message |

#### Returns

if success returns num of attributes in schema, else returns EXIT\_ERROR

### 5.38.2.2 AK\_get\_attr\_index()

```
int AK_get_attr_index (
    char * tblName,
    char * attrName )
```

Function that gets zero-based index for attribute.

#### Author

Matija Šestak.

#### Parameters

|                  |                |
|------------------|----------------|
| <i>*tblName</i>  | table name     |
| <i>*attrName</i> | attribute name |

**Returns**

zero-based index

**5.38.2.3 AK\_get\_attr\_name()**

```
char* AK_get_attr_name (
    char * tblName,
    int index )
```

Function that gets attribute name for some zero-based index.

**Author**

Matija Šestak.

**Parameters**

|                 |                  |
|-----------------|------------------|
| <i>*tblName</i> | table name       |
| <i>index</i>    | zero-based index |

**Returns**

attribute name

**5.38.2.4 AK\_get\_column()**

```
struct list_node* AK_get_column (
    int num,
    char * tblName )
```

Function that gets all values in some column and put on the list.

**Author**

Matija Šestak.

**Parameters**

|                 |                         |
|-----------------|-------------------------|
| <i>num</i>      | zero-based column index |
| <i>*tblName</i> | table name              |

**Returns**

column values list



### 5.38.2.5 AK\_get\_header()

```
AK_header* AK_get_header (
    char * tblName )
```

Function that getts table header.

#### Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return 0
3. else read the first block
4. allocate array
5. copy table header to the array

#### Parameters

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

#### Returns

array of table header

### 5.38.2.6 AK\_get\_num\_records()

```
int AK_get_num_records (
    char * tblName )
```

Determine number of rows in the table.

#### Author

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return EXIT\_WARNING
3. For each extent from table
4. For each block in the extent
5. Get a block
6. Exit if there is no records in block
7. Count tuples in block
8. Return the number of tuples divided by number of attributes

**Parameters**

|                   |            |
|-------------------|------------|
| <i>*tableName</i> | table name |
|-------------------|------------|

**Returns**

number of rows in the table

**5.38.2.7 AK\_get\_row()**

```
struct list_node* AK_get_row (
    int num,
    char * tblName )
```

Function that gets all values in some row and put on the list.

**Author**

Markus Schatten, Matija Šestak.

**Parameters**

|            |                      |
|------------|----------------------|
| <i>num</i> | zero-based row index |
| <i>*</i>   | tblName table name   |

**Returns**

row values list

**5.38.2.8 AK\_get\_table\_obj\_id()**

```
int AK_get_table_obj_id (
    char * table )
```

Function that gets obj\_id of named table from AK\_relation system table.

**Author**

Dejan Frankovic

**Parameters**

|               |            |
|---------------|------------|
| <i>*table</i> | table name |
|---------------|------------|

**Returns**

obj\_id of the table or EXIT\_ERROR if there is no table with that name

**5.38.2.9 AK\_get\_tuple()**

```
struct list_node* AK_get_tuple (
    int row,
    int column,
    char * tblName )
```

Function that gets value in some row and column.

**Author**

Matija Šestak.

**Parameters**

|                 |                         |
|-----------------|-------------------------|
| <i>row</i>      | zero-based row index    |
| <i>column</i>   | zero-based column index |
| <i>*tblName</i> | table name              |

**Returns**

value in the list

**5.38.2.10 AK\_num\_attr()**

```
int AK_num_attr (
    char * tblName )
```

Determine the number of attributes in the table.

**Author**

Matija Šestak.

1. Read addresses of extents
2. If there is no extents in the table, return EXIT\_WARNING
3. else read the first block
4. while header tuple exists in the block, increment num\_attr

**Parameters**

|   |                    |
|---|--------------------|
| * | tblName table name |
|---|--------------------|

**Returns**

number of attributes in the table

**5.38.2.11 AK\_op\_rename\_test()**

```
void AK_op_rename_test ( )
```

Function for rename operator testing (moved from rename.c)

**Author**

Mislav Čakarić, edited by Ljubo Barać

**Returns**

No return value

**5.38.2.12 AK\_print\_row()**

```
void AK_print_row (
    int col_len[],
    struct list_node * row )
```

Function that prints table row.

**Author**

Dino Laktašić

**Parameters**

|                  |   |
|------------------|---|
| <i>col_len[]</i> | array of max lengths for each attribute |
| <i>*row</i>      | list with row elements                  |

**Returns**

No return value

**5.38.2.13 AK\_print\_row\_spacer()**

```
void AK_print_row_spacer (
    int col_len[],
    int length )
```

Function that prints row spacer.

**Author**

Dino Laktašić.

**Parameters**

|                  |                                     |
|------------------|-------------------------------------|
| <i>col_len[]</i> | max lengths for each attribute cell |
| <i>length</i>    | total table width                   |

**Returns**

printed row spacer

**5.38.2.14 AK\_print\_row\_spacer\_to\_file()**

```
void AK_print_row_spacer_to_file (
    int col_len[],
    int length )
```

Function that prints row spacer update by Luka Rajcevic.

**Author**

Dino Laktašić.

**Parameters**

|                  |                                     |
|------------------|-------------------------------------|
| <i>col_len[]</i> | max lengths for each attribute cell |
| <i>length</i>    | total table width                   |

**Returns**

printed row spacer

**5.38.2.15 AK\_print\_row\_to\_file()**

```
void AK_print_row_to_file (
    int col_len[],
    struct list_node * row )
```

Function that prints table row update by Luka Rajcevic.

**Author**

Dino Laktašić

**Parameters**

|                  |   |
|------------------|---|
| <i>col_len[]</i> | array of max lengths for each attribute |
| <i>*row</i>      | list with row elements                  |

**Returns**

No return value

**5.38.2.16 AK\_print\_table()**

```
void AK_print_table (
    char * tblName )
```

Function for printing table.

**Author**

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one)

**Parameters**

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

**Returns**

No return value

**5.38.2.17 AK\_print\_table\_to\_file()**

```
void AK_print_table_to_file (
    char * tblName )
```

Function for printing table.

**Author**

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one) update by Luka Rajcevic

**Parameters**

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

**Returns**

No return value update by Anto Tomaš (corrected the Ak\_DeleteAll\_L3 function)

**5.38.2.18 AK\_rename()**

```
int AK_rename (
    char * old_table_name,
    char * old_attr,
    char * new_table_name,
    char * new_attr )
```

Function for renaming table and/or attribute in table (moved from rename.c)

**Author**

Mislav Čakarić edited by Ljubo Barać

**Parameters**

|                       |                                      |
|-----------------------|--------------------------------------|
| <i>old_table_name</i> | old name of the table                |
| <i>new_table_name</i> | new name of the table                |
| <i>old_attr</i>       | name of the attribute to rename      |
| <i>new_attr</i>       | new name for the attribute to rename |

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**5.38.2.19 AK\_table\_empty()**

```
int AK_table_empty (
    char * tblName )
```

Function that check whether table is empty.

**Author**

Matija Šestak.

**Parameters**

|                 |            |
|-----------------|------------|
| <i>*tblName</i> | table name |
|-----------------|------------|

**Returns**

true/false

**5.38.2.20 AK\_table\_test()**

```
void AK_table_test ( )
```

Function for testing table abstraction.

**Author**

Unknown

**Returns**

No return value

by Ana-Marija Balen - added getRow function to the test

**5.38.2.21 AK\_tuple\_to\_string()**

```
char* AK_tuple_to_string (
    struct list_node * tuple )
```

Function that converts tuple value to string.

**Author**

Matija Šestak.

**Parameters**

|               |                   |
|---------------|-------------------|
| <i>*tuple</i> | tuple in the list |
|---------------|-------------------|

**Returns**

tuple value as a string

**5.38.2.22 get\_row\_attr\_data()**

```
char* get_row_attr_data (
    int column,
    struct list_node * node )
```

Function that returns value of attribute from row.



**Author**

Leon Palaić

**Parameters**

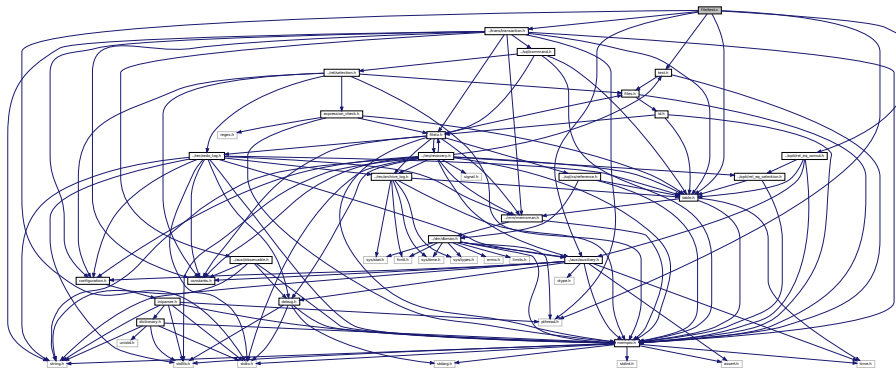
|               |                           |
|---------------|---------------------------|
| <i>column</i> | index of column attribute |
| <i>*row</i>   | list with row elements    |

**Returns**

atribute data

**5.39 file/test.c File Reference**

```
#include <pthread.h>
#include <stdio.h>
#include "test.h"
#include "../trans/transaction.h"
#include "../file/table.h"
#include "../auxiliary/auxiliary.h"
#include "../opti/rel_eq_comut.h"
Include dependency graph for test.c:
```

**Functions**

- char \* [AK\\_get\\_table\\_attribute\\_types](#) (char \*tblName)  
*returns a string containing attribute types for supplied table name, seperated by ATTR\_DELIMITER*
- int [create\\_header\\_test](#) (char \*tbl\_name, char \*\*attr\_name, int \_num, int \*\_type)  
*Function for creating test table header.*
- int [insert\\_data\\_test](#) (char \*tbl\_name, char \*\*attr\_name, char \*\*attr\_value, int \_num, int \*\_type)  
*Function for inserting test data into table (needed for python testing)*
- int [selection\\_test](#) (char \*src\_table, char \*dest\_table, char \*\*sel\_query, int \_num, int \*\_type)  
*Function for selection operator on one table.*
- int [get\\_column\\_test](#) (int num, char \*tbl)  
*prints requested column*
- int [get\\_row\\_test](#) (int num, char \*tbl)  
*prints requested row*
- void [AK\\_create\\_test\\_tables](#) ()  
*Function for creating test tables.*

### 5.39.1 Detailed Description

Provides functions for testing purposes

### 5.39.2 Function Documentation

#### 5.39.2.1 AK\_create\_test\_tables()

```
void AK_create_test_tables ( )
```

Function for creating test tables.

##### Author

Dino Laktašić

##### Returns

No return value

#### 5.39.2.2 AK\_get\_table\_attribute\_types()

```
char* AK_get_table_attribute_types (
    char * tblName )
```

returns a string containing attribute types for supplied table name, seperated by ATTR\_DELIMITER

##### Author

Goran Štok

##### Parameters

|                |  |
|----------------|--|
| <i>tblName</i> | name of the table for which the attribute types will be returned |
|----------------|--|

#### 5.39.2.3 create\_header\_test()

```
int create_header_test (
    char * tbl_name,
```

```
char ** attr_name,  
int _num,  
int * _type )
```

Function for creating test table header.

#### Author

Luka Rajcevic

#### Parameters

|                  |   |
|------------------|---|
| <i>tbl_name</i>  | - name of the table for which the header will be created      |
| <i>attr_name</i> | - array of attribute names                                    |
| <i>_num</i>      | - number of attributes  |
| <i>_type</i>     | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

#### Returns

1 if ok, 0 otherwise

#### 5.39.2.4 get\_column\_test()

```
int get_column_test (  
    int num,  
    char * tbl )
```

prints requested column

#### Author

Luka Rajcevic

#### Returns

1 if column is found, 0 otherwise

#### Parameters

|            |                           |
|------------|---------------------------|
| <i>num</i> | - 0 based index of column |
| <i>tbl</i> | - name of the table       |

#### 5.39.2.5 get\_row\_test()

```
int get_row_test (
```

```
int num,  
char * tbl )
```

prints requested row

#### Author

Luka Rajcevic

#### Returns

1 if row is found, 0 otherwise

#### Parameters

|            |                        |
|------------|------------------------|
| <i>num</i> | - 0 based index of row |
| <i>tbl</i> | - name of the table    |

#### 5.39.2.6 insert\_data\_test()

```
int insert_data_test (  
    char * tbl_name,  
    char ** attr_name,  
    char ** attr_value,  
    int _num,  
    int * _type )
```

Function for inserting test data into table (needed for python testing)

#### Author

Luka Rajcevic

#### Parameters

|                   |   |
|-------------------|---|
| <i>tbl_name</i>   | - name of the table for which the header will be created      |
| <i>attr_name</i>  | - array of attribute names                                    |
| <i>attr_value</i> | - values of attributes to be inserted                         |
| <i>_num</i>       | - number of attributes  |
| <i>_type</i>      | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

#### Returns

EXIT\_SUCCESS if ok, EXIT\_ERROR otherwise

### 5.39.2.7 selection\_test()

```
int selection_test (
    char * src_table,
    char * dest_table,
    char ** sel_query,
    int _num,
    int * _type )
```

Function for selection operator on one table.

## Author

Luka Rajcevic

•

## Parameters

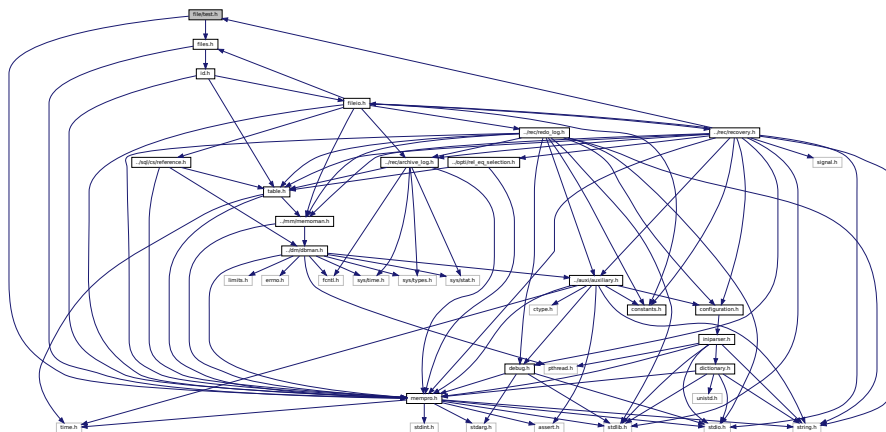
|                   |   |
|-------------------|---|
| <i>src_table</i>  | - name of the source table<br>•                               |
| <i>dest_table</i> | - table in which selection will be stored                     |
| <i>sel_query</i>  | - array of operators, operands and attributes (postfix query) |
| <i>_num</i>       | - number of attributes  |
| <i>_type</i>      | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

## Returns

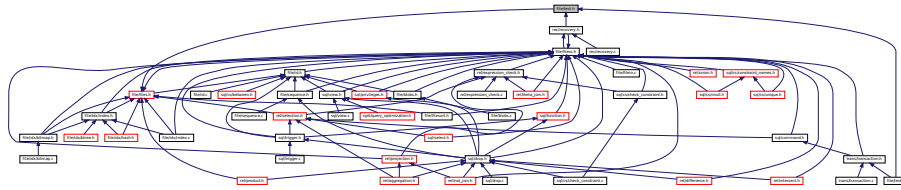
EXIT\_SUCCESS if ok, EXIT\_ERROR otherwise

## 5.40 file/test.h File Reference

```
#include "files.h"
#include "../auxi/mempro.h"
Include dependency graph for test.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- char \* [AK\\_get\\_table\\_attribute\\_types](#) (char \*tblName)  
*returns a string containing attribute types for supplied table name, seperated by ATTR\_DELIMITER*
- int [create\\_header\\_test](#) (char \*tbl\_name, char \*\*attr\_name, int \_num, int \*\_type)  
*Function for creating test table header.*
- int [insert\\_data\\_test](#) (char \*tbl\_name, char \*\*attr\_name, char \*\*attr\_value, int \_num, int \*\_type)  
*Function for inserting test data into table (needed for python testing)*
- int [selection\\_test](#) (char \*src\_table, char \*dest\_table, char \*\*sel\_query, int \_num, int \*\_type)  
*Function for selection operator on one table.*
- int [get\\_column\\_test](#) (int num, char \*tbl)  
*prints requested column*
- int [get\\_row\\_test](#) (int num, char \*tbl)  
*prints requested row*
- void [AK\\_create\\_test\\_tables](#) ()  
*Function for creating test tables.*

### 5.40.1 Detailed Description

Header file that provides functions for testing purposes

### 5.40.2 Function Documentation

#### 5.40.2.1 AK\_create\_test\_tables()

```
void AK_create_test_tables ( )
```

Function for creating test tables.

#### Author

Dino Laktašić

#### Returns

No return value

### 5.40.2.2 AK\_get\_table\_attribute\_types()

```
char* AK_get_table_attribute_types (
    char * tblName )
```

returns a string containing attribute types for supplied table name, seperated by ATTR\_DELIMITER

#### Author

Goran Štok

#### Parameters

|                |  |
|----------------|--|
| <i>tblName</i> | name of the table for which the attribute types will be returned |
|----------------|--|

### 5.40.2.3 create\_header\_test()

```
int create_header_test (
    char * tbl_name,
    char ** attr_name,
    int _num,
    int * _type )
```

Function for creating test table header.

#### Author

Luka Rajcevic

#### Parameters

|                  |   |
|------------------|---|
| <i>tbl_name</i>  | - name of the table for which the header will be created      |
| <i>attr_name</i> | - array of attribute names                                    |
| <i>_num</i>      | - number of attributes  |
| <i>_type</i>     | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

#### Returns

1 if ok, 0 otherwise

### 5.40.2.4 get\_column\_test()

```
int get_column_test (
    int num,
    char * tbl )
```

prints requested column

**Author**

Luka Rajcevic

**Returns**

1 if column is found, 0 otherwise

**Parameters**

|            |                           |
|------------|---------------------------|
| <i>num</i> | - 0 based index of column |
| <i>tbl</i> | - name of the table       |

**5.40.2.5 get\_row\_test()**

```
int get_row_test (
    int num,
    char * tbl )
```

prints requested row

**Author**

Luka Rajcevic

**Returns**

1 if row is found, 0 otherwise

**Parameters**

|            |                        |
|------------|------------------------|
| <i>num</i> | - 0 based index of row |
| <i>tbl</i> | - name of the table    |

**5.40.2.6 insert\_data\_test()**

```
int insert_data_test (
    char * tbl_name,
    char ** attr_name,
    char ** attr_value,
    int _num,
    int * _type )
```

Function for inserting test data into table (needed for python testing)



**Author**

Luka Rajcevic

**Parameters**

|                   |   |
|-------------------|---|
| <i>tbl_name</i>   | - name of the table for which the header will be created      |
| <i>attr_name</i>  | - array of attribute names                                    |
| <i>attr_value</i> | - values of attributes to be inserted                         |
| <i>_num</i>       | - number of attributes  |
| <i>_type</i>      | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

**Returns**

EXIT\_SUCCESS if ok, EXIT\_ERROR otherwise

**5.40.2.7 selection\_test()**

```
int selection_test (
    char * src_table,
    char * dest_table,
    char ** sel_query,
    int _num,
    int * _type )
```

Function for selection operator on one table.

**Author**

Luka Rajcevic

•

**Parameters**

|                   |   |
|-------------------|---|
| <i>src_table</i>  | - name of the source table<br>•                               |
| <i>dest_table</i> | - table in which selection will be stored                     |
| <i>sel_query</i>  | - array of operators, operands and attributes (postfix query) |
| <i>_num</i>       | - number of attributes  |
| <i>_type</i>      | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

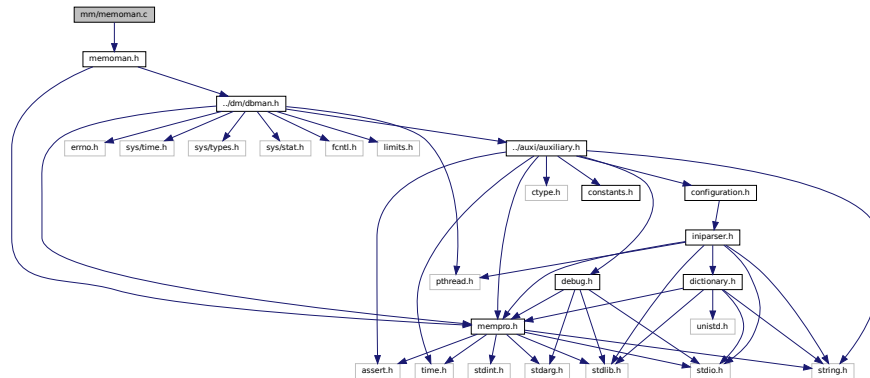
**Returns**

EXIT\_SUCCESS if ok, EXIT\_ERROR otherwise

## 5.41 mm/memoman.c File Reference

```
#include "memoman.h"
```

Include dependency graph for memoman.c:



## Functions

- `int AK_cache_block (int num, AK_mem_block *mem_block)`  
Function caches block into memory.
- `int AK_cache_AK_malloc ()`  
Function initializes the global cache memory (variable `db_cache`)
- `int AK_redo_log_AK_malloc ()`  
Function initializes the global redo log memory (variable `redo_log`)
- `int AK_find_available_result_block ()`  
Function find available block for result caching in circular array.
- `unsigned long AK_generate_result_id (unsigned char *str)`  
Generate unique hash identifier for each cached result by using djb2 algorithm.
- `void AK_cache_result (char *srcTable, AK_block *temp_block, AK_header header[])`  
Cache fetched result block in memory.
- `int AK_query_mem_AK_malloc ()`  
Function initializes the global query memory (variable `query_mem`)
- `int AK_memoman_init ()`  
Function initializes memory manager (cache, redo log and query memory)
- `AK_mem_block * AK_get_block (int num)`  
Function reads a block from memory. If the block is cached returns the cached block. Else uses `AK_cache_block` to read the block to cache and then returns it.
- `void AK_mem_block_modify (AK_mem_block *mem_block, int dirty)`  
Modify the "dirty" bit of a block, and update timestamps accordingly.
- `int AK_refresh_cache ()`  
Function re-read all the blocks from disk.
- `table_addresses * AK_get_index_segment_addresses (char *segmentName)`  
Function for getting addresses of some table.
- `table_addresses * AK_get_segment_addresses (char *segmentName)`  
Function for getting addresses of some table.
- `table_addresses * AK_get_table_addresses (char *table)`  
function for getting addresses of some table

- [table\\_addresses](#) \* [AK\\_get\\_index\\_addresses](#) (char \*index)  
*Function for getting addresses of some index.*
- int [AK\\_find\\_AK\\_free\\_space](#) ([table\\_addresses](#) \*addresses)  
*Function to find AK\_free space in some block between block addresses. It's made for insert\_row()*
- int [AK\\_init\\_new\\_extent](#) (char \*table\_name, int extent\_type)  
*Function that extends the segment.*
- int [AK\\_flush\\_cache](#) ()  
*Function that flushes memory blocks to disk file.*
- void [AK\\_memoman\\_test](#) ()
- void [AK\\_memoman\\_test2](#) ()

### 5.41.1 Detailed Description

Defines functions for the memory manager of Kalashnikov DB

### 5.41.2 Function Documentation

#### 5.41.2.1 [AK\\_cache\\_AK\\_malloc\(\)](#)

```
int AK_cache_AK_malloc ( )
```

Function initializes the global cache memory (variable db\_cache)

#### Author

Markus Schatten, Matija Šestak(revised)

#### Returns

EXIT\_SUCCESS if the cache memory has been initialized, EXIT\_ERROR otherwise

#### 5.41.2.2 [AK\\_cache\\_block\(\)](#)

```
int AK_cache_block (
    int num,
    AK\_mem\_block * mem_block )
```

Function caches block into memory.

#### Author

Nikola Bakoš, Matija Šestak(revised)

**Parameters**

|                  |                          |
|------------------|--------------------------|
| <i>num</i>       | block number (address)   |
| <i>mem_block</i> | address of memmory block |

**Returns**

EXIT\_SUCCESS if the block has been successfully read into memory, EXIT\_ERROR otherwise

read the block from the given address

set dirty bit in mem\_block struct

get the timestamp

set timestamp\_read

set timestamp\_last\_change

**5.41.2.3 AK\_cache\_result()**

```
void AK_cache_result (
    char * srcTable,
    AK_block * temp_block,
    AK_header header[] )
```

Cache fetched result block in memory.

**Author**

Mario Novoselec

**5.41.2.4 AK\_find\_AK\_free\_space()**

```
int AK_find_AK_free_space (
    table_addresses * addresses )
```

Function to find AK\_free space in some block between block addresses. It's made for insert\_row()

**Author**

Matija Novak, updated by Matija Šestak( function now uses caching)

**Parameters**

|                |                      |
|----------------|----------------------|
| <i>address</i> | addresses of extents |
|----------------|----------------------|

**Returns**

address of the block to write in

**5.41.2.5 AK\_find\_available\_result\_block()**

```
int AK_find_available_result_block ( )
```

Function find available block for result caching in circular array.

**Author**

Mario Novoselec

**Returns**

available\_index

**5.41.2.6 AK\_flush\_cache()**

```
int AK_flush_cache ( )
```

Function that flushes memory blocks to disk file.

**Author**

Matija Šestak

**Returns**

EXIT\_SUCCESS

if block form cache can not be writed to DB file -> EXIT\_ERROR

**5.41.2.7 AK\_generate\_result\_id()**

```
unsigned long AK_generate_result_id (
    unsigned char * str )
```

Generate unique hash identifier for each cached result by using djb2 algorithm.

**Author**

Mario Novoselec

**Returns**

hash

#### 5.41.2.8 AK\_get\_block()

```
AK_mem_block* AK_get_block (
    int num )
```

Function reads a block from memory. If the block is cached returns the cached block. Else uses AK\_cache\_block to read the block to cache and then returns it.

##### Author

Tomislav Fotak, updated by Matija Šestak

##### Parameters

|            |                        |
|------------|------------------------|
| <i>num</i> | block number (address) |
|------------|------------------------|

##### Returns

segment start address

if block form cache can not be writed to DB file -> EXIT\_ERROR

if block form cache can not be writed to DB file -> EXIT\_ERROR

#### 5.41.2.9 AK\_get\_index\_addresses()

```
table_addresses* AK_get_index_addresses (
    char * index )
```

Function for geting addresses of some index.

##### Author

Mislav Čakarić

##### Parameters

|              |                                |
|--------------|--------------------------------|
| <i>index</i> | index name that you search for |
|--------------|--------------------------------|

##### Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

#### 5.41.2.10 AK\_get\_index\_segment\_addresses()

```
table_addresses* AK_get_index_segment_addresses (
    char * segmentName )
```

Function for geting addresses of some table.

**Author**

Matija Novak, updated by Matija Šestak(function now uses caching), modified and renamed by Mislav Čakarić,Lovro Predovan

**Parameters**

|              |                                |
|--------------|--------------------------------|
| <i>table</i> | table name that you search for |
|--------------|--------------------------------|

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

**5.41.2.11 AK\_get\_segment\_addresses()**

```
table_addresses* AK_get_segment_addresses (
    char * segmentName )
```

Function for geting addresses of some table.

**Author**

Matija Novak, updated by Matija Šestak(function now uses caching), modified and renamed by Mislav Čakarić

**Parameters**

|              |                                |
|--------------|--------------------------------|
| <i>table</i> | table name that you search for |
|--------------|--------------------------------|

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

**5.41.2.12 AK\_get\_table\_addresses()**

```
table_addresses* AK_get_table_addresses (
    char * table )
```

function for geting addresses of some table

**Author**

Mislav Čakarić

**Parameters**

|              |                                |
|--------------|--------------------------------|
| <i>table</i> | table name that you search for |
|--------------|--------------------------------|

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when from and to are 0 you are on the end of addresses

**5.41.2.13 AK\_init\_new\_extent()**

```
int AK_init_new_extent (
    char * table_name,
    int extent_type )
```

Function that extends the segment.

**Author**

Nikola Bakoš, updated by Matija Šestak (function now uses caching), updated by Mislav Čakarić, updated by Dino Laktašić

**Parameters**

|                    |   |
|--------------------|---|
| <i>table_name</i>  | name of segment to extent   |
| <i>extent_type</i> | type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP |

**Returns**

address of new extent, otherwise EXIT\_ERROR

!! to correct header BUG iterate through header from 0 to N-th block while there is

**5.41.2.14 AK\_mem\_block\_modify()**

```
void AK_mem_block_modify (
    AK_mem_block * mem_block,
    int dirty )
```

Modify the "dirty" bit of a block, and update timestamps accordingly.

**Author**

Alen Novosel.



#### 5.41.2.15 AK\_memoman\_init()

```
int AK_memoman_init ( )
```

Function initializes memory manager (cache, redo log and query memory)

##### Author

Miroslav Policki

##### Returns

EXIT\_SUCCESS if the query memory manager has been initialized, EXIT\_ERROR otherwise

#### 5.41.2.16 AK\_query\_mem\_AK\_malloc()

```
int AK_query_mem_AK_malloc ( )
```

Function initializes the global query memory (variable query\_mem)

##### Author

Matija Novak

##### Returns

EXIT\_SUCCESS if the query memory has been initialized, EXIT\_ERROR otherwise

allocate memory for global variable query\_mem

allocate memory for variable query\_mem\_lib which is used in query\_mem->parsed

allocate memory for variable query\_mem\_dict which is used in query\_mem->dictionary

allocate memory for variable query\_mem\_result which is used in query\_mem->result

allocate memory for variable tuple\_dict which is used in query\_mem->dictionary->dictionary[]

#### 5.41.2.17 AK\_redo\_log\_AK\_malloc()

```
int AK_redo_log_AK_malloc ( )
```

Function initializes the global redo log memory (variable redo\_log)

##### Author

Dejan Sambolić updated by Dražen Bandić, updated by Tomislav Turek

##### Returns

EXIT\_SUCCESS if the redo log memory has been initialized, EXIT\_ERROR otherwise

### 5.41.2.18 AK\_refresh\_cache()

```
int AK_refresh_cache ( )
```

Function re-read all the blocks from disk.

#### Author

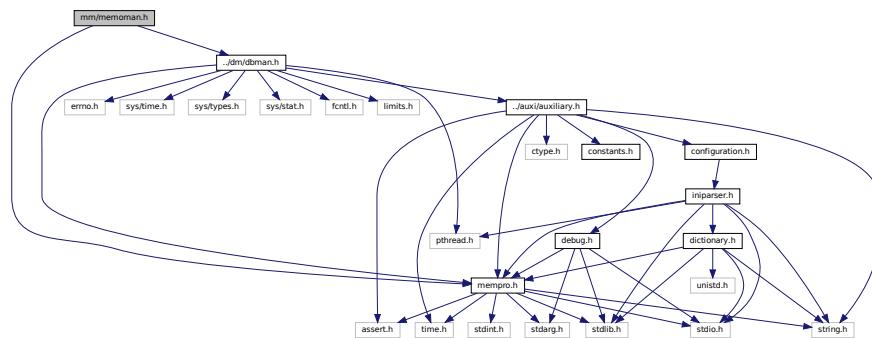
Matija Šestak.

#### Returns

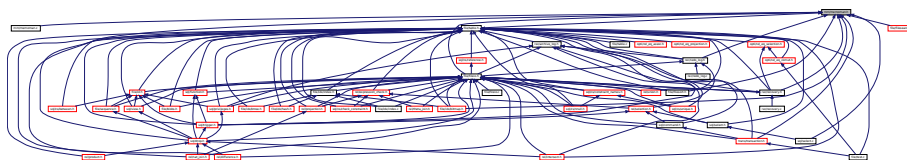
EXIT\_SUCCESS

## 5.42 mm/memoman.h File Reference

```
#include "../dm/dbman.h"
#include "../aux/mempro.h"
Include dependency graph for memoman.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [AK\\_mem\\_block](#)  
Structure that defines a block of data in memory.
- struct [AK\\_db\\_cache](#)  
Structure that defines global cache memory.
- struct [AK\\_command\\_recovery\\_struct](#)  
recovery structure used to recover commands from binary file

- struct [AK\\_redo\\_log](#)  
*Structure that defines global redo log.*
- struct [AK\\_query\\_mem\\_lib](#)  
*Structure that defines global query memory for libraries.*
- struct [AK\\_query\\_mem\\_dict](#)  
*Structure that defines global query memory for data dictionaries.*
- struct [AK\\_results](#)  
*Structure used for in-memory result caching.*
- struct [AK\\_query\\_mem\\_result](#)  
*Structure that defines global query memory for results.*
- struct [AK\\_query\\_mem](#)  
*Structure that defines global query memory.*

## Functions

- void [AK\\_cache\\_result](#) (char \*srcTable, [AK\\_block](#) \*temp\_block, [AK\\_header](#) header[])  
*Cache fetched result block in memory.*
- int [AK\\_find\\_available\\_result\\_block](#) ()  
*Function find available block for result caching in circular array.*
- unsigned long [AK\\_generate\\_result\\_id](#) (unsigned char \*str)  
*Generate unique hash identifier for each cached result by using djb2 algorithm.*
- int [AK\\_cache\\_block](#) (int num, [AK\\_mem\\_block](#) \*mem\_block)  
*Function caches block into memory.*
- int [AK\\_cache\\_AK\\_malloc](#) ()  
*Function initializes the global cache memory (variable db\_cache)*
- int [AK\\_redo\\_log\\_AK\\_malloc](#) ()  
*Function initializes the global redo log memory (variable redo\_log)*
- int [AK\\_query\\_mem\\_AK\\_malloc](#) ()  
*Function initializes the global query memory (variable query\_mem)*
- int [AK\\_memoman\\_init](#) ()  
*Function initializes memory manager (cache, redo log and query memory)*
- [AK\\_mem\\_block](#) \* [AK\\_get\\_block](#) (int num)  
*Function reads a block from memory. If the block is cached returns the cached block. Else uses AK\_cache\_block to read the block to cache and then returns it.*
- void [AK\\_mem\\_block\\_modify](#) ([AK\\_mem\\_block](#) \*mem\_block, int dirty)  
*Modify the "dirty" bit of a block, and update timestamps accordingly.*
- int [AK\\_refresh\\_cache](#) ()  
*Function re-read all the blocks from disk.*
- [table\\_addresses](#) \* [AK\\_get\\_segment\\_addresses](#) (char \*segmentName)  
*Function for getting addresses of some table.*
- [table\\_addresses](#) \* [AK\\_get\\_index\\_segment\\_addresses](#) (char \*segmentName)  
*Function for getting addresses of some table.*
- [table\\_addresses](#) \* [AK\\_get\\_table\\_addresses](#) (char \*table)  
*function for getting addresses of some table*
- [table\\_addresses](#) \* [AK\\_get\\_index\\_addresses](#) (char \*index)  
*Function for getting addresses of some index.*
- int [AK\\_find\\_AK\\_free\\_space](#) ([table\\_addresses](#) \*addresses)  
*Function to find AK\_free space in some block between block addresses. It's made for insert\_row()*
- int [AK\\_init\\_new\\_extent](#) (char \*table\_name, int extent\_type)  
*Function that extends the segment.*
- int [AK\\_flush\\_cache](#) ()  
*Function that flushes memory blocks to disk file.*
- void [AK\\_memoman\\_test](#) ()
- void [AK\\_memoman\\_test2](#) ()

## Variables

- [AK\\_db\\_cache](#) \* [db\\_cache](#)  
*Variable that defines the db cache.*
- [AK\\_redo\\_log](#) \* [redo\\_log](#)  
*Variable that defines the global redo log.*
- [AK\\_query\\_mem](#) \* [query\\_mem](#)  
*Variable that defines the global query memory.*

### 5.42.1 Detailed Description

Header file that defines includes and datastructures for the memory manager of Kalashnikov DB

### 5.42.2 Function Documentation

#### 5.42.2.1 AK\_cache\_AK\_malloc()

```
int AK_cache_AK_malloc ( )
```

Function initializes the global cache memory (variable [db\\_cache](#))

#### Author

Markus Schatten, Matija Šestak(revised)

#### Returns

EXIT\_SUCCESS if the cache memory has been initialized, EXIT\_ERROR otherwise

#### 5.42.2.2 AK\_cache\_block()

```
int AK_cache_block (
    int num,
    AK\_mem\_block * mem_block )
```

Function caches block into memory.

#### Author

Nikola Bakoš, Matija Šestak(revised)

## Parameters

|                  |                         |
|------------------|-------------------------|
| <i>num</i>       | block number (address)  |
| <i>mem_block</i> | address of memory block |

## Returns

EXIT\_SUCCESS if the block has been successfully read into memory, EXIT\_ERROR otherwise

read the block from the given address

set dirty bit in mem\_block struct

get the timestamp

set timestamp\_read

set timestamp\_last\_change

## 5.42.2.3 AK\_cache\_result()

```
void AK_cache_result (
    char * srcTable,
    AK_block * temp_block,
    AK_header header[] )
```

Cache fetched result block in memory.

## Author

Mario Novoselec

## 5.42.2.4 AK\_find\_AK\_free\_space()

```
int AK_find_AK_free_space (
    table_addresses * addresses )
```

Function to find AK\_free space in some block between block addresses. It's made for insert\_row()

## Author

Matija Novak, updated by Matija Šestak( function now uses caching)

## Parameters

|                |                      |
|----------------|----------------------|
| <i>address</i> | addresses of extents |
|----------------|----------------------|

**Returns**

address of the block to write in

**5.42.2.5 AK\_find\_available\_result\_block()**

```
int AK_find_available_result_block ( )
```

Function find available block for result caching in circular array.

**Author**

Mario Novoselec

**Returns**

available\_index

**5.42.2.6 AK\_flush\_cache()**

```
int AK_flush_cache ( )
```

Function that flushes memory blocks to disk file.

**Author**

Matija Šestak

**Returns**

EXIT\_SUCCESS

if block form cache can not be writed to DB file -> EXIT\_ERROR

**5.42.2.7 AK\_generate\_result\_id()**

```
unsigned long AK_generate_result_id (
    unsigned char * str )
```

Generate unique hash identifier for each cached result by using djb2 algorithm.

**Author**

Mario Novoselec

**Returns**

hash

#### 5.42.2.8 AK\_get\_block()

```
AK_mem_block* AK_get_block (
    int num )
```

Function reads a block from memory. If the block is cached returns the cached block. Else uses AK\_cache\_block to read the block to cache and then returns it.

##### Author

Tomislav Fotak, updated by Matija Šestak

##### Parameters

|            |                        |
|------------|------------------------|
| <i>num</i> | block number (address) |
|------------|------------------------|

##### Returns

segment start address

if block form cache can not be writed to DB file -> EXIT\_ERROR

if block form cache can not be writed to DB file -> EXIT\_ERROR

#### 5.42.2.9 AK\_get\_index\_addresses()

```
table_addresses* AK_get_index_addresses (
    char * index )
```

Function for geting addresses of some index.

##### Author

Mislav Čakarić

##### Parameters

|              |                                |
|--------------|--------------------------------|
| <i>index</i> | index name that you search for |
|--------------|--------------------------------|

##### Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

#### 5.42.2.10 AK\_get\_index\_segment\_addresses()

```
table_addresses* AK_get_index_segment_addresses (
    char * segmentName )
```

Function for geting addresses of some table.

**Author**

Matija Novak, updated by Matija Šestak(function now uses caching), modified and renamed by Mislav Čakarić,Lovro Predovan

**Parameters**

|              |                                |
|--------------|--------------------------------|
| <i>table</i> | table name that you search for |
|--------------|--------------------------------|

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

**5.42.2.11 AK\_get\_segment\_addresses()**

```
table_addresses* AK_get_segment_addresses (
    char * segmentName )
```

Function for geting addresses of some table.

**Author**

Matija Novak, updated by Matija Šestak(function now uses caching), modified and renamed by Mislav Čakarić

**Parameters**

|              |                                |
|--------------|--------------------------------|
| <i>table</i> | table name that you search for |
|--------------|--------------------------------|

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

**5.42.2.12 AK\_get\_table\_addresses()**

```
table_addresses* AK_get_table_addresses (
    char * table )
```

function for geting addresses of some table

**Author**

Mislav Čakarić



## Parameters

|              |                                |
|--------------|--------------------------------|
| <i>table</i> | table name that you search for |
|--------------|--------------------------------|

## Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when from and to are 0 you are on the end of addresses

## 5.42.2.13 AK\_init\_new\_extent()

```
int AK_init_new_extent (
    char * table_name,
    int extent_type )
```

Function that extends the segment.

## Author

Nikola Bakoš, updated by Matija Šestak (function now uses caching), updated by Mislav Čakarić, updated by Dino Laktašić

## Parameters

|                    |   |
|--------------------|---|
| <i>table_name</i>  | name of segment to extent   |
| <i>extent_type</i> | type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP |

## Returns

address of new extent, otherwise EXIT\_ERROR

!! to correct header BUG iterate through header from 0 to N-th block while there is

## 5.42.2.14 AK\_mem\_block\_modify()

```
void AK_mem_block_modify (
    AK_mem_block * mem_block,
    int dirty )
```

Modify the "dirty" bit of a block, and update timestamps accordingly.

## Author

Alen Novosel.

#### 5.42.2.15 AK\_memoman\_init()

```
int AK_memoman_init ( )
```

Function initializes memory manager (cache, redo log and query memory)

##### Author

Miroslav Policki

##### Returns

EXIT\_SUCCESS if the query memory manager has been initialized, EXIT\_ERROR otherwise

#### 5.42.2.16 AK\_query\_mem\_AK\_malloc()

```
int AK_query_mem_AK_malloc ( )
```

Function initializes the global query memory (variable query\_mem)

##### Author

Matija Novak

##### Returns

EXIT\_SUCCESS if the query memory has been initialized, EXIT\_ERROR otherwise

allocate memory for global variable query\_mem

allocate memory for variable query\_mem\_lib which is used in query\_mem->parsed

allocate memory for variable query\_mem\_dict which is used in query\_mem->dictionary

allocate memory for variable query\_mem\_result which is used in query\_mem->result

allocate memory for variable tuple\_dict which is used in query\_mem->dictionary->dictionary[]

#### 5.42.2.17 AK\_redo\_log\_AK\_malloc()

```
int AK_redo_log_AK_malloc ( )
```

Function initializes the global redo log memory (variable redo\_log)

##### Author

Dejan Sambolić updated by Dražen Bandić, updated by Tomislav Turek

##### Returns

EXIT\_SUCCESS if the redo log memory has been initialized, EXIT\_ERROR otherwise

## 5.42.2.18 AK\_refresh\_cache()

```
int AK_refresh_cache ( )
```

Function re-read all the blocks from disk.

## Author

Matija Šestak.

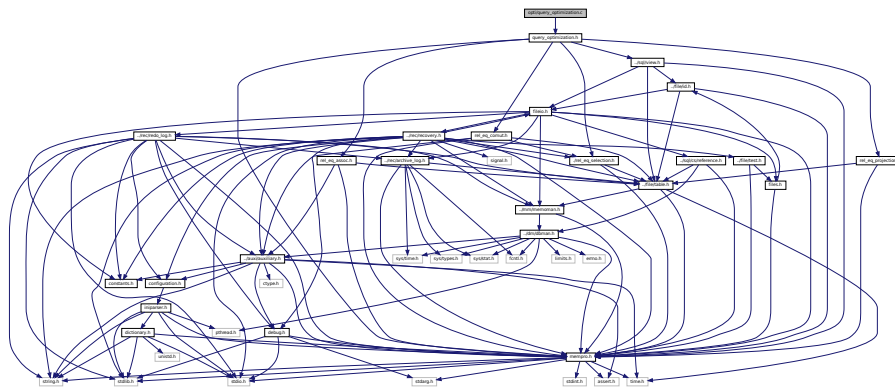
## Returns

EXIT\_SUCCESS

## 5.43 opti/query\_optimization.c File Reference

```
#include "query_optimization.h"
```

Include dependency graph for query\_optimization.c:



## Functions

- void [AK\\_print\\_optimized\\_query](#) (struct [list\\_node](#) \*list\_query)  
*Print optimization table for testing purposes.*
- struct [list\\_node](#) \* [AK\\_execute\\_rel\\_eq](#) (struct [list\\_node](#) \*list\_query, const char rel\_eq, const char \*FLAGS)  
*Call and execute relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection.*
- struct [list\\_node](#) \* [AK\\_query\\_optimization](#) (struct [list\\_node](#) \*list\_query, const char \*FLAGS, const int DIFF↔\_PLANS)  
*Execute all relational equivalences provided by FLAGS (one or more), if DIFF\_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.*
- void [AK\\_query\\_optimization\\_test](#) ()

## 5.43.1 Detailed Description

Provides functions for general query optimization

## 5.43.2 Function Documentation

### 5.43.2.1 AK\_execute\_rel\_eq()

```
struct list_node* AK_execute_rel_eq (
    struct list_node * list_query,
    const char rel_eq,
    const char * FLAGS )
```

Call and execute relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection.

#### Author

Dino Laktašić.

#### Parameters

|                    |  |
|--------------------|--|
| <i>*list_query</i> | RA expresion list where we need to apply relational equivalences rules |
| <i>rel_eq</i>      | rel_eq to execute  |
| <i>*FLAGS</i>      | flags for relation equivalences (execute rel_eq for given flags)       |

#### Returns

returns struct `list_node` (RA expresion list) optimized by given relational equivalence rule

### 5.43.2.2 AK\_print\_optimized\_query()

```
void AK_print_optimized_query (
    struct list_node * list_query )
```

Print optimization table for testing purposes.

#### Author

Dino Laktašić.

#### Parameters

|                    |                             |
|--------------------|-----------------------------|
| <i>*list_query</i> | optimized RA expresion list |
|--------------------|-----------------------------|

#### Returns

list output

### 5.43.2.3 AK\_query\_optimization()

```
struct list_node* AK_query_optimization (
    struct list_node * list_query,
    const char * FLAGS,
    const int DIFF_PLANS )
```

Execute all relational equivalences provided by FLAGS (one or more), if DIFF\_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.

#### Author

Dino Laktašić.

#### Parameters

|             |  |
|-------------|--|
| *list_query | RA expresion list where we need to apply relational equivalences rules |
| *FLAGS      | flags for relation equivalences (execute rel_eq for given flags)       |

#### Returns

returns AK\_list (RA expresion list) optimized by all relational equivalence rules provided by FLAGS (commented code can be edited so AK\_list can return the list of lists (lists of different optimization plans), with permutation switched on (DIFF\_PLANS = 1) time for execution will be significantly increased Current implementation without uncommenting code doesn't produce list of list, it rather apply all permutations on the same list

For futher development consider to implement cost estimation for given plan based on returned heuristicly optimized list

### 5.43.2.4 AK\_query\_optimization\_test()

```
void AK_query_optimization_test ( )
```

#### Author

Dino Laktašić

#### Parameters

|             |                       |
|-------------|-----------------------|
| *list_query | query to be optimized |
|-------------|-----------------------|

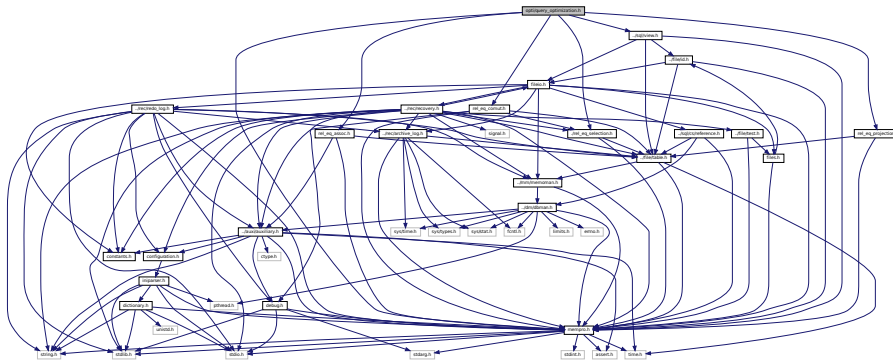
#### Returns

No return value

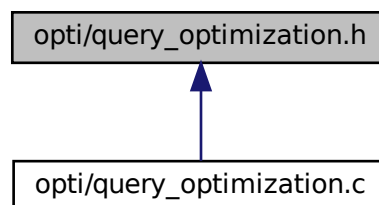
## 5.44 opti/query\_optimization.h File Reference

```
#include "rel_eq_comut.h"
#include "rel_eq_assoc.h"
#include "rel_eq_projection.h"
#include "rel_eq_selection.h"
#include "../auxi/mempro.h"
#include "../sql/view.h"
```

Include dependency graph for query\_optimization.h:



This graph shows which files directly or indirectly include this file:



### Macros

- `#define MAX_PERMUTATION 24`  
*Constant declaring maximum number of permutations.*

### Functions

- `void AK_print_optimized_query (struct list_node *list_query)`  
*Print optimization table for testing purposes.*
- `struct list_node * AK_execute_rel_eq (struct list_node *list_query, const char rel_eq, const char *FLAGS)`  
*Call and execute relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection.*

- struct [list\\_node](#) \* [AK\\_query\\_optimization](#) (struct [list\\_node](#) \*list\_query, const char \*FLAGS, const int DIFF\_PLANS)  
*Execute all relational equivalences provided by FLAGS (one or more), if DIFF\_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.*
- void [AK\\_query\\_optimization\\_test](#) ()

### 5.44.1 Detailed Description

Header file that provides functions for general query optimization

### 5.44.2 Function Documentation

#### 5.44.2.1 AK\_execute\_rel\_eq()

```
struct list\_node* AK_execute_rel_eq (
    struct list\_node * list_query,
    const char rel_eq,
    const char * FLAGS )
```

Call and execute relation equivalence RELATION EQUIVALENCE RULES FLAGS c - commutation a - associativity p - projection s - selection.

#### Author

Dino Laktašić.

#### Parameters

|             |  |
|-------------|--|
| *list_query | RA expresion list where we need to apply relational equivalences rules |
| rel_eq      | rel_eq to execute  |
| *FLAGS      | flags for relation equivalences (execute rel_eq for given flags)       |

#### Returns

returns struct [list\\_node](#) (RA expresion list) optimized by given relational equivalence rule

#### 5.44.2.2 AK\_print\_optimized\_query()

```
void AK_print_optimized_query (
    struct list\_node * list_query )
```

Print optimization table for testing purposes.

#### Author

Dino Laktašić.

**Parameters**

|                    |                             |
|--------------------|-----------------------------|
| <i>*list_query</i> | optimized RA expresion list |
|--------------------|-----------------------------|

**Returns**

list output

**5.44.2.3 AK\_query\_optimization()**

```
struct list_node* AK_query_optimization (
    struct list_node * list_query,
    const char * FLAGS,
    const int DIFF_PLANS )
```

Execute all relational equivalences provided by FLAGS (one or more), if DIFF\_PLANS turned on execute permutations without repetition on given RA list from SQL parser output.

**Author**

Dino Laktašić.

**Parameters**

|                    |  |
|--------------------|--|
| <i>*list_query</i> | RA expresion list where we need to apply relational equivalences rules |
| <i>*FLAGS</i>      | flags for relation equivalences (execute rel_eq for given flags)       |

**Returns**

returns AK\_list (RA expresion list) optimized by all relational equivalence rules provided by FLAGS (commented code can be edited so AK\_list can return the list of lists (lists of different optimization plans), with permutation switched on (DIFF\_PLANS = 1) time for execution will be significantly increased Current implementation without uncommenting code doesn't produce list of list, it rather apply all permutations on the same list

For futher development consider to implement cost estimation for given plan based on returned heuristicly optimized list

**5.44.2.4 AK\_query\_optimization\_test()**

```
void AK_query_optimization_test ( )
```

**Author**

Dino Laktašić



### Parameters

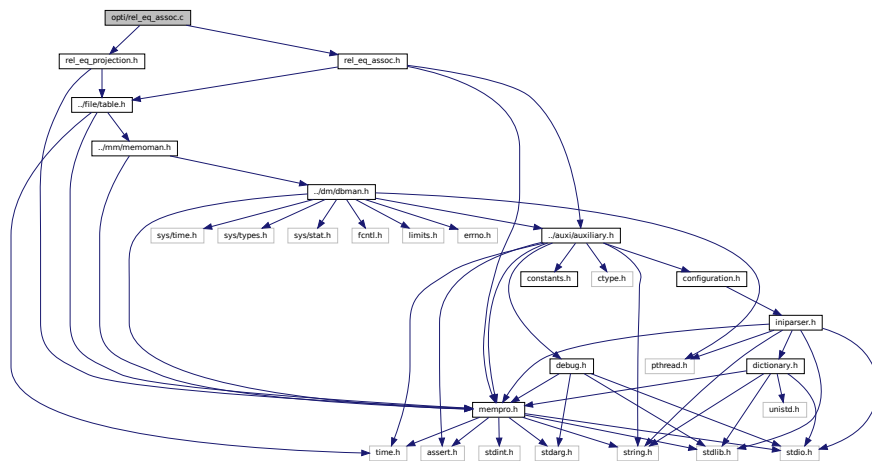
|                     |                       |
|---------------------|-----------------------|
| * <i>list_query</i> | query to be optimized |
|---------------------|-----------------------|

## Returns

No return value

### 5.45 opti/rel\_eq\_assoc.c File Reference

```
#include "rel_eq_assoc.h"
#include "rel_eq_projection.h"
Include dependency graph for rel_eq_assoc.c:
```



## Functions

- int **AK\_compare** (const void \*a, const void \*b)  
*Function for Struct cost\_eval comparison.*
- struct **list\_node** \* **AK\_rel\_eq\_assoc** (struct **list\_node** \*list\_rel\_eq)  
*Main function for generating RA expression according to associativity equivalence rules.*
- void **AK\_print\_rel\_eq\_assoc** (struct **list\_node** \*list\_rel\_eq)  
*Function for printing RA expression struct **list\_node**.*
- void **AK\_rel\_eq\_assoc\_test** ()  
*Function for testing relational equivalences regarding associativity.*

### 5.45.1 Detailed Description

Provides functions for for relational equivalences regarding associativity

### 5.45.2 Function Documentation

#### 5.45.2.1 AK\_compare()

```
int AK_compare (
    const void * a,
    const void * b )
```

Function for Struct cost\_eval comparison.

##### Author

Dino Laktašić

##### Parameters

|           |              |
|-----------|--------------|
| <i>*a</i> | first value  |
| <i>*b</i> | second value |

##### Returns

returns result of comparison

#### 5.45.2.2 AK\_print\_rel\_eq\_assoc()

```
void AK_print_rel_eq_assoc (
    struct list_node * list_rel_eq )
```

Function for printing RA expresion struct [list\\_node](#).

##### Author

Dino Laktašić.

##### Parameters

|                     |  |
|---------------------|--|
| <i>*list_rel_eq</i> | RA expresion as the struct <a href="#">list_node</a> |
|---------------------|--|

##### Returns

optimised RA expresion as the struct [list\\_node](#)

#### 5.45.2.3 AK\_rel\_eq\_assoc()

```
struct list_node* AK_rel_eq_assoc (
    struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to associativity equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

|              |  |
|--------------|--|
| *list_rel_eq | RA expresion as the struct <a href="#">list_node</a> |
|--------------|--|

**Returns**optimised RA expresion as the struct [list\\_node](#)**5.45.2.4 AK\_rel\_eq\_assoc\_test()**

```
void AK_rel_eq_assoc_test ( )
```

Function for testing relational equivalences regarding associativity.

**Author**

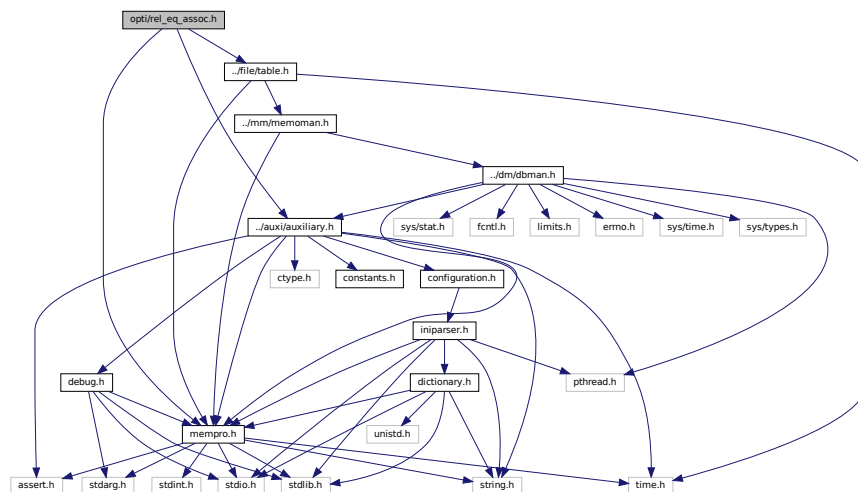
Dino Laktašić.

**Returns**

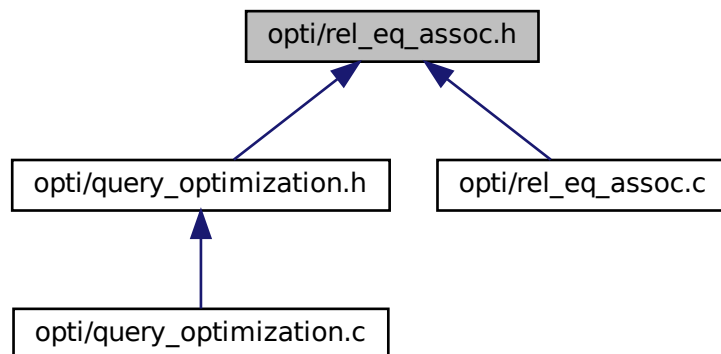
No return value

**5.46 opti/rel\_eq\_assoc.h File Reference**

```
#include "../file/table.h"
#include "../aux/mempro.h"
#include "../aux/auxiliary.h"
Include dependency graph for rel_eq_assoc.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [cost\\_eval\\_t](#)  
*Structure for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name)*

## Typedefs

- typedef struct [cost\\_eval\\_t](#) **cost\_eval**

## Functions

- int [AK\\_compare](#) (const void \*a, const void \*b)  
*Function for Struct cost\_eval comparison.*
- struct [list\\_node](#) \* [AK\\_rel\\_eq\\_assoc](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Main function for generating RA expresion according to associativity equivalence rules.*
- void [AK\\_print\\_rel\\_eq\\_assoc](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Function for printing RA expresion struct list\_node.*
- void [AK\\_rel\\_eq\\_assoc\\_test](#) ()  
*Function for testing relational equivalences regarding associativity.*

### 5.46.1 Detailed Description

Header file that provides data structures for relational equivalences regarding associativity

### 5.46.2 Function Documentation

### 5.46.2.1 AK\_compare()

```
int AK_compare (
    const void * a,
    const void * b )
```

Function for Struct cost\_eval comparison.

#### Author

Dino Laktašić

#### Parameters

|           |              |
|-----------|--------------|
| <i>*a</i> | first value  |
| <i>*b</i> | second value |

#### Returns

returns result of comparison

### 5.46.2.2 AK\_print\_rel\_eq\_assoc()

```
void AK_print_rel_eq_assoc (
    struct list_node * list_rel_eq )
```

Function for printing RA expresion struct [list\\_node](#).

#### Author

Dino Laktašić.

#### Parameters

|                     |  |
|---------------------|--|
| <i>*list_rel_eq</i> | RA expresion as the struct <a href="#">list_node</a> |
|---------------------|--|

#### Returns

optimised RA expresion as the struct [list\\_node](#)

### 5.46.2.3 AK\_rel\_eq\_assoc()

```
struct list_node* AK_rel_eq_assoc (
    struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to associativity equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

|                      |  |
|----------------------|--|
| * <i>list_rel_eq</i> | RA expresion as the struct <a href="#">list_node</a> |
|----------------------|--|

**Returns**optimised RA expression as the struct [list\\_node](#)**5.46.2.4 AK\_rel\_eq\_assoc\_test()**

```
void AK_rel_eq_assoc_test ( )
```

Function for testing relational equivalences regarding associativity.

**Author**

Dino Laktašić.

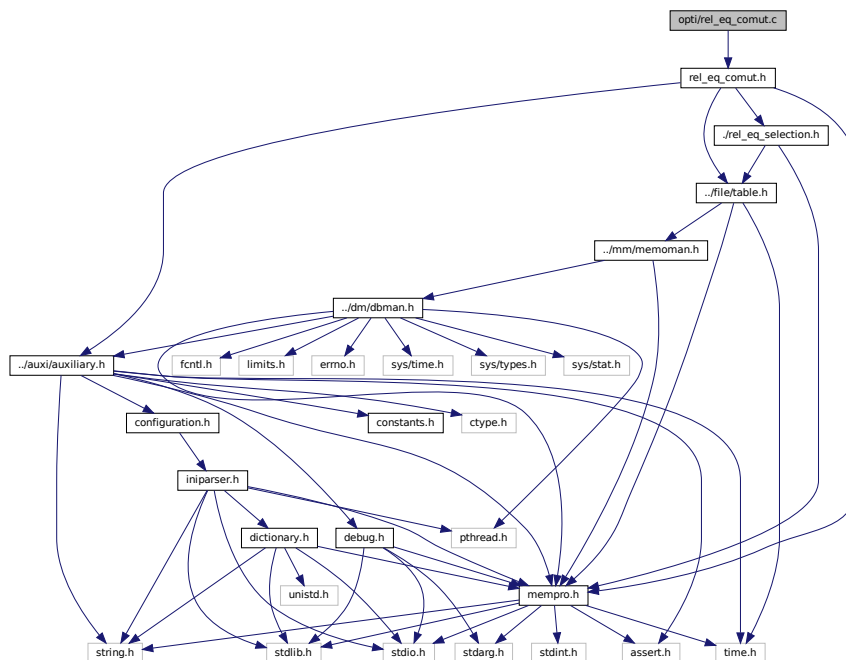
**Returns**

No return value

**5.47 opti/rel\_eq\_comut.c File Reference**

```
#include "rel_eq_comut.h"
```

Include dependency graph for rel\_eq\_comut.c:



## Functions

- void [AK\\_print\\_rel\\_eq\\_comut](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Function for printing optimized relation equivalence expression list regarding commutativity.*
- struct [list\\_node](#) \* [AK\\_rel\\_eq\\_comut](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Main function for generating RA expresion according to commutativity equivalence rules.*
- char \* [AK\\_rel\\_eq\\_commute\\_with\\_theta\\_join](#) (char \*cond, char \*tblName)  
*Check if selection can commute with theta-join or product.*
- void [AK\\_rel\\_eq\\_comut\\_test](#) ()  
*relational equivalences regarding commutativity*

### 5.47.1 Detailed Description

Provides functions for relational equivalences regarding commutativity

### 5.47.2 Function Documentation

#### 5.47.2.1 [AK\\_print\\_rel\\_eq\\_comut\(\)](#)

```
void AK_print_rel_eq_comut (
    struct list\_node * list_rel_eq )
```

Function for printing optimized relation equivalence expression list regarding commutativity.

#### Author

Davor Tomala

#### Parameters

|                     |  |
|---------------------|--|
| <i>*list_rel_eq</i> | RA expresion as the struct <a href="#">list_node</a> |
|---------------------|--|

#### 5.47.2.2 [AK\\_rel\\_eq\\_commute\\_with\\_theta\\_join\(\)](#)

```
char* AK_rel_eq_commute_with_theta_join (
    char * cond,
    char * tblName )
```

Check if selection can commute with theta-join or product.

**Author**

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else set id to 0, else make no changes to variable id
4. if token differs from "AND" and "OR" and id equals to 1 append current token to result condition
5. else if token equals to "AND" or "OR" and id equals to 1 and there are two added tokens add "AND" or "OR" to condition string
6. When exits from loop, return pointer to char array that contains new condition for a given table

**Parameters**

|                 |  |
|-----------------|--|
| <i>*cond</i>    | condition array that contains condition data |
| <i>*tblName</i> | name of the table                            |

**Returns**

pointer to char array that contains new condition for a given table

**5.47.2.3 AK\_rel\_eq\_comut()**

```
struct list_node* AK_rel_eq_comut (
    struct list_node * list_rel_eq )
```

Main function for generating RA expression according to commutativity equivalence rules.

**Author**

Davor Tomala

**Parameters**

|                     |   |
|---------------------|---|
| <i>*list_rel_eq</i> | RA expression as the struct <a href="#">list_node</a> |
|---------------------|---|

**Returns**

optimised RA expression as the struct [list\\_node](#)

**5.47.2.4 AK\_rel\_eq\_comut\_test()**

```
void AK_rel_eq_comut_test ( )
```

relational equivalences regarding commutativity



**Author**

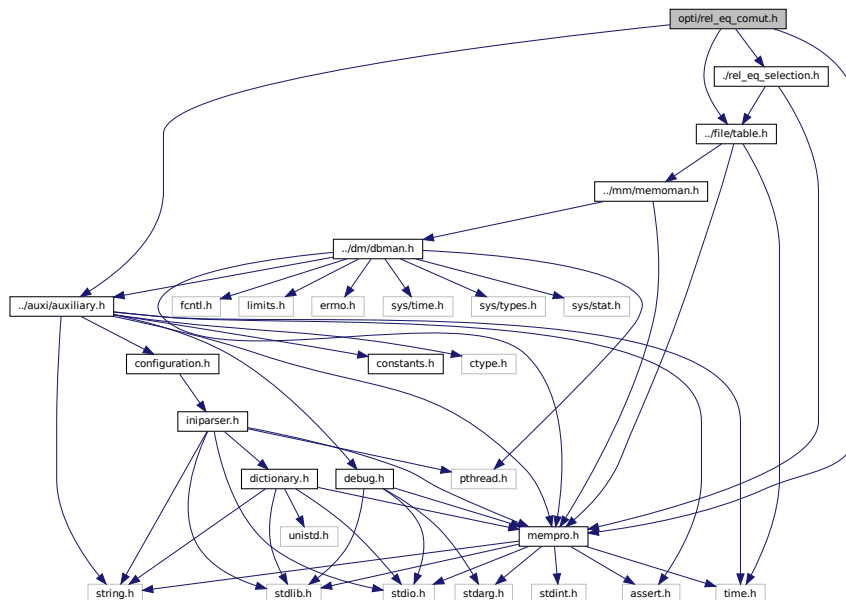
Dino Laktašić (AK\_rel\_eq\_commute\_with\_theta\_join), Davor Tomala (AK\_rel\_eq\_comut)

**Returns**

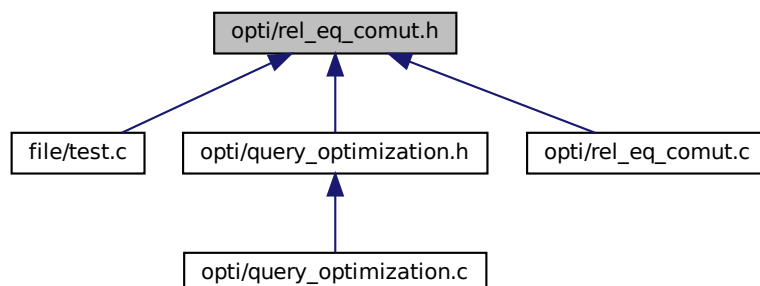
No return vlaue

**5.48 opti/rel\_eq\_comut.h File Reference**

```
#include "../file/table.h"
#include "../rel_eq_selection.h"
#include "../auxi/mempro.h"
#include "../auxi/auxiliary.h"
Include dependency graph for rel_eq_comut.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- void [AK\\_print\\_rel\\_eq\\_comut](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Function for printing optimized relation equivalence expression list regarding commutativity.*
- struct [list\\_node](#) \* [AK\\_rel\\_eq\\_comut](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Main function for generating RA expresion according to commutativity equivalence rules.*
- char \* [AK\\_rel\\_eq\\_commute\\_with\\_theta\\_join](#) (char \*cond, char \*tblName)  
*Check if selection can commute with theta-join or product.*
- void [AK\\_rel\\_eq\\_comut\\_test](#) ()  
*relational equivalences regarding commutativity*

### 5.48.1 Detailed Description

Header file that provides data structures for relational equivalences regarding comutativity

### 5.48.2 Function Documentation

#### 5.48.2.1 [AK\\_print\\_rel\\_eq\\_comut\(\)](#)

```
void AK_print_rel_eq_comut (
    struct list\_node * list_rel_eq )
```

Function for printing optimized relation equivalence expression list regarding commutativity.

#### Author

Davor Tomala

#### Parameters

|                     |  |
|---------------------|--|
| <i>*list_rel_eq</i> | RA expresion as the struct <a href="#">list_node</a> |
|---------------------|--|

#### 5.48.2.2 [AK\\_rel\\_eq\\_commute\\_with\\_theta\\_join\(\)](#)

```
char* AK_rel_eq_commute_with_theta_join (
    char * cond,
    char * tblName )
```

Check if selection can commute with theta-join or product.

**Author**

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else set id to 0, else make no changes to variable id
4. if token differs from "AND" and "OR" and id equals to 1 append current token to result condition
5. else if token equals to "AND" or "OR" and id equals to 1 and there are two added tokens add "AND" or "OR" to condition string
6. When exits from loop, return pointer to char array that contains new condition for a given table

**Parameters**

|                 |  |
|-----------------|--|
| <i>*cond</i>    | condition array that contains condition data |
| <i>*tblName</i> | name of the table                            |

**Returns**

pointer to char array that contains new condition for a given table

**5.48.2.3 AK\_rel\_eq\_comut()**

```
struct list_node* AK_rel_eq_comut (
    struct list_node * list_rel_eq )
```

Main function for generating RA expression according to commutativity equivalence rules.

**Author**

Davor Tomala

**Parameters**

|                     |   |
|---------------------|---|
| <i>*list_rel_eq</i> | RA expression as the struct <a href="#">list_node</a> |
|---------------------|---|

**Returns**

optimised RA expression as the struct [list\\_node](#)

**5.48.2.4 AK\_rel\_eq\_comut\_test()**

```
void AK_rel_eq_comut_test ( )
```

relational equivalences regarding commutativity



## 5.49.1 Detailed Description

Provides functions for for relational equivalences in projection

## 5.49.2 Function Documentation

### 5.49.2.1 AK\_print\_rel\_eq\_projection()

```
void AK_print_rel_eq_projection (
    struct list_node * list_rel_eq )
```

Function for printing AK\_list to the screen.

#### Author

Dino Laktašić.

#### Parameters

|                     |                             |
|---------------------|-----------------------------|
| <i>*list_rel_eq</i> | RA expresion as the AK_list |
|---------------------|-----------------------------|

#### Returns

No return value

### 5.49.2.2 AK\_rel\_eq\_can\_commute()

```
int AK_rel_eq_can_commute (
    struct list_node * list_elem_attribs,
    struct list_node * list_elem_conds )
```

Check if selection uses only attributes retained by the projection before commuting.

#### Author

Dino Laktašić.

1. Tokenize set of projection attributes and store them to the array
2. For each attribute in selection condition check if exists in array of projection attributes
3. if exists increment match variable and break
4. else continue checking until the final attribute is checked
5. if match variable value equals 0 than return 0
6. else if match variable value greater than EXIT\_SUCCESS, return EXIT\_FAILURE

**Parameters**

|                         |  |
|-------------------------|--|
| <i>list_elem_attrbs</i> | list element containing projection data          |
| <i>list_elem_conds</i>  | list element containing selection condition data |

**Returns**

EXIT\_SUCCESS if selection uses only attributes retained by projection, else returns EXIT\_FAILURE

**5.49.2.3 AK\_rel\_eq\_collect\_cond\_attributes()**

```
char* AK_rel_eq_collect_cond_attributes (
    struct list_node * list_elem )
```

Filtering and returning only attributes from selection or theta\_join condition.

**Author**

Dino Laktašić.

**Parameters**

|                  |   |
|------------------|---|
| <i>list_elem</i> | list element that contains selection or theta_join condition data |
|------------------|---|

**Returns**

only attributes from selection or theta\_join condition as the AK\_list

**5.49.2.4 AK\_rel\_eq\_get\_attributes()**

```
struct list_node* AK_rel_eq_get_attributes (
    char * tblName )
```

Get attributes for a given table and store them to the struct [list\\_node](#).

**Author**

Dino Laktašić.

1. Get the number of attributes in a given table
2. Get the table header for a given table
3. Initialize struct [list\\_node](#)
4. For each attribute in table header, insert attribute in struct [list\\_node](#) as new struct [list\\_node](#) element
5. return struct [list\\_node](#)

## Parameters

|                 |                   |
|-----------------|-------------------|
| <i>*tblName</i> | name of the table |
|-----------------|-------------------|

## Returns

struct [list\\_node](#)

## 5.49.2.5 AK\_rel\_eq\_is\_subset()

```
int AK_rel_eq_is_subset (
    struct list\_node * list_elem_set,
    struct list\_node * list_elem_subset )
```

Check if some set of attributes is subset of larger set, used in cascading of the projections.

## Author

Dino Laktašić. =====> Optimization plan using Relational Algebra Equivalences <=====

Equivalence rule that apply on every equivalent expresion generated by Query optimizer

Rules to implement Rule 1. projection comutes with selection that only uses attributes retained by the projection  $p[L](s[L1](R)) = s[L1](p[L](R))$  Rule 2. only the last in a sequence of projection operations is needed, the others can be omitted.  $p[L1] = p[L1](R)$  Rule 3a. distribution according to theta join, only if join includes attributes from  $L1 \cup L2$   $p[L1 \cup L2](R1 \bowtie R2) = (p[L1](R1)) \bowtie (p[L2](R2))$  Rule 3b. Let  $L1 \cup L2$  be attributes from  $R1$  and  $R2$ , respectively. Let  $L3$  be attributes from  $R1$ , but are not in  $L1 \cup L2$  and let  $L4$  be attributes from  $R2$ , but are not in  $L1 \cup L2$ .  $p[L1 \cup L2](R1 \bowtie R2) = p[L1 \cup L2]((p[L1 \cup L3](R1)) \bowtie (p[L2 \cup L4](R2)))$  Rule 4. distribution according to union  $p[L](R1 \cup R2) = (p[L](R1)) \cup (p[L](R2))$

## Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT\_SUCCESS

## Parameters

|                         |  |
|-------------------------|--|
| <i>list_elem_set</i>    | first list element containing projection attributes  |
| <i>list_elem_subset</i> | second list element containing projection attributes |

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**5.49.2.6 AK\_rel\_eq\_projection()**

```
struct list_node* AK_rel_eq_projection (
    struct list_node * list_rel_eq )
```

Main function for generating RA expression according to projection equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

|                     |                              |
|---------------------|------------------------------|
| <i>*list_rel_eq</i> | RA expression as the AK_list |
|---------------------|------------------------------|

**Returns**

optimised RA expression as the AK\_list

**5.49.2.7 AK\_rel\_eq\_projection\_attributes()**

```
char* AK_rel_eq_projection_attributes (
    char * attribs,
    char * tblName )
```

Filtering and returning only those attributes from list of projection attributes that exist in the given table.

**Author**

Dino Laktašić.

1. Get the attributes for a given table and store them to the AK\_list
2. Tokenize set of projection attributes and store them to the array
3. For each attribute in the array check if exists in the previously created AK\_list
4. if exists append attribute to the dynamic attributes char array
5. return pointer to char array with stored attribute/s

**Parameters**

|                 |   |
|-----------------|---|
| <i>*attribs</i> | projection attributes delimited by ";" (ATTR_DELIMITER) |
| <i>*tblName</i> | name of the table                                       |



**Returns**

filtered list of projection attributes as the AK\_list

**5.49.2.8 AK\_rel\_eq\_projection\_test()**

```
void AK_rel_eq_projection_test ( )
```

Function for testing rel\_eq\_selection.

**Author**

Dino Laktašić.

**Returns**

No return value

**5.49.2.9 AK\_rel\_eq\_remove\_duplicates()**

```
char* AK_rel_eq_remove_duplicates (
    char * attrs )
```

Function which removes duplicate attributes from attributes expresion.

**Author**

Dino Laktašić.

**Parameters**

|               |  |
|---------------|--|
| <i>*attrs</i> | attributes from which to remove duplicates |
|---------------|--|

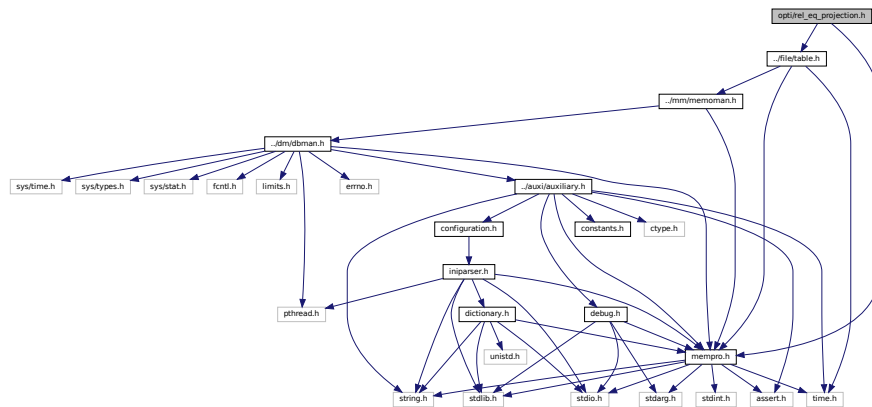
**Returns**

pointer to char array without duplicate attributes

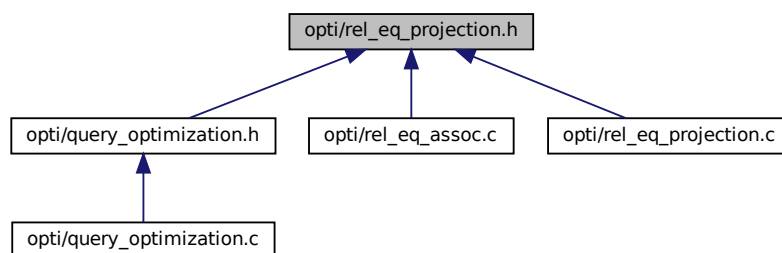
**5.50 opti/rel\_eq\_projection.h File Reference**

```
#include "../file/table.h"
#include "../auxi/mempro.h"
```

Include dependency graph for `rel_eq_projection.h`:



This graph shows which files directly or indirectly include this file:



## Functions

- int [AK\\_rel\\_eq\\_is\\_subset](#) (struct [list\\_node](#) \*list\_elem\_set, struct [list\\_node](#) \*list\_elem\_subset)  
*Check if some set of attributes is subset of larger set, used in cascading of the projections.*
- int [AK\\_rel\\_eq\\_can\\_commute](#) (struct [list\\_node](#) \*list\_elem\_attribs, struct [list\\_node](#) \*list\_elem\_conds)  
*Check if selection uses only attributes retained by the projection before commuting.*
- struct [list\\_node](#) \* [AK\\_rel\\_eq\\_get\\_attributes](#) (char \*tblName)  
*Get attributes for a given table and store them to the struct [list\\_node](#).*
- char \* [AK\\_rel\\_eq\\_projection\\_attributes](#) (char \*attribs, char \*tblName)  
*Filtering and returning only those attributes from list of projection attributes that exist in the given table.*
- char \* [AK\\_rel\\_eq\\_collect\\_cond\\_attributes](#) (struct [list\\_node](#) \*list\_elem)  
*Filtering and returning only attributes from selection or theta\_join condition.*
- char \* [AK\\_rel\\_eq\\_remove\\_duplicates](#) (char \*attribs)  
*Function which removes duplicate attributes from attributes expression.*
- struct [list\\_node](#) \* [AK\\_rel\\_eq\\_projection](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Main function for generating RA expression according to projection equivalence rules.*
- void [AK\\_print\\_rel\\_eq\\_projection](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Function for printing AK\_list to the screen.*
- void [AK\\_rel\\_eq\\_projection\\_test](#) ()  
*Function for testing rel\_eq\_selection.*

### 5.50.1 Detailed Description

Header file that provides data structures for relational equivalences in projection

### 5.50.2 Function Documentation

#### 5.50.2.1 AK\_print\_rel\_eq\_projection()

```
void AK_print_rel_eq_projection (
    struct list_node * list_rel_eq )
```

Function for printing AK\_list to the screen.

#### Author

Dino Laktašić.

#### Parameters

|                     |                             |
|---------------------|-----------------------------|
| <i>*list_rel_eq</i> | RA expresion as the AK_list |
|---------------------|-----------------------------|

#### Returns

No return value

#### 5.50.2.2 AK\_rel\_eq\_can\_commute()

```
int AK_rel_eq_can_commute (
    struct list_node * list_elem_attribs,
    struct list_node * list_elem_conds )
```

Check if selection uses only attributes retained by the projection before commuting.

#### Author

Dino Laktašić.

1. Tokenize set of projection attributes and store them to the array
2. For each attribute in selection condition check if exists in array of projection attributes
3. if exists increment match variable and break
4. else continue checking until the final attribute is checked
5. if match variable value equals 0 than return 0
6. else if match variable value greater than EXIT\_SUCCESS, return EXIT\_FAILURE

**Parameters**

|                         |  |
|-------------------------|--|
| <i>list_elem_attrbs</i> | list element containing projection data          |
| <i>list_elem_conds</i>  | list element containing selection condition data |

**Returns**

EXIT\_SUCCESS if selection uses only attributes retained by projection, else returns EXIT\_FAILURE

**5.50.2.3 AK\_rel\_eq\_collect\_cond\_attributes()**

```
char* AK_rel_eq_collect_cond_attributes (
    struct list_node * list_elem )
```

Filtering and returning only attributes from selection or theta\_join condition.

**Author**

Dino Laktašić.

**Parameters**

|                  |   |
|------------------|---|
| <i>list_elem</i> | list element that contains selection or theta_join condition data |
|------------------|---|

**Returns**

only attributes from selection or theta\_join condition as the AK\_list

**5.50.2.4 AK\_rel\_eq\_get\_attributes()**

```
struct list_node* AK_rel_eq_get_attributes (
    char * tblName )
```

Get attributes for a given table and store them to the struct [list\\_node](#).

**Author**

Dino Laktašić.

1. Get the number of attributes in a given table
2. Get the table header for a given table
3. Initialize struct [list\\_node](#)
4. For each attribute in table header, insert attribute in struct [list\\_node](#) as new struct [list\\_node](#) element
5. return struct [list\\_node](#)

## Parameters

|                 |                   |
|-----------------|-------------------|
| <i>*tblName</i> | name of the table |
|-----------------|-------------------|

## Returns

struct [list\\_node](#)

## 5.50.2.5 AK\_rel\_eq\_is\_subset()

```
int AK_rel_eq_is_subset (
    struct list\_node * list_elem_set,
    struct list\_node * list_elem_subset )
```

Check if some set of attributes is subset of larger set, used in cascading of the projections.

## Author

Dino Laktašić. =====> Optimization plan using Relational Algebra Equivalences <=====

Equivalence rule that apply on every equivalent expresion generated by Query optimizer

Rules to implement Rule 1. projection comutes with selection that only uses attributes retained by the projection  $p[L](s[L1](R)) = s[L1](p[L](R))$  Rule 2. only the last in a sequence of projection operations is needed, the others can be omitted.  $p[L1] = p[L1](R)$  Rule 3a. distribution according to theta join, only if join includes attributes from  $L1 \cup L2$   $p[L1 \cup L2](R1 \bowtie R2) = (p[L1](R1)) \bowtie (p[L2](R2))$  Rule 3b. Let  $L1 \cup L2$  be attributes from  $R1$  and  $R2$ , respectively. Let  $L3$  be attributes from  $R1$ , but are not in  $L1 \cup L2$  and let  $L4$  be attributes from  $R2$ , but are not in  $L1 \cup L2$ .  $p[L1 \cup L2](R1 \bowtie R2) = p[L1 \cup L2]((p[L1 \cup L3](R1)) \bowtie (p[L2 \cup L4](R2)))$  Rule 4. distribution according to union  $p[L](R1 \cup R2) = (p[L](R1)) \cup (p[L](R2))$

## Author

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT\_SUCCESS

## Parameters

|                         |  |
|-------------------------|--|
| <i>list_elem_set</i>    | first list element containing projection attributes  |
| <i>list_elem_subset</i> | second list element containing projection attributes |

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**5.50.2.6 AK\_rel\_eq\_projection()**

```
struct list_node* AK_rel_eq_projection (
    struct list_node * list_rel_eq )
```

Main function for generating RA expression according to projection equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

|                     |                              |
|---------------------|------------------------------|
| <i>*list_rel_eq</i> | RA expression as the AK_list |
|---------------------|------------------------------|

**Returns**

optimised RA expression as the AK\_list

**5.50.2.7 AK\_rel\_eq\_projection\_attributes()**

```
char* AK_rel_eq_projection_attributes (
    char * attribs,
    char * tblName )
```

Filtering and returning only those attributes from list of projection attributes that exist in the given table.

**Author**

Dino Laktašić.

1. Get the attributes for a given table and store them to the AK\_list
2. Tokenize set of projection attributes and store them to the array
3. For each attribute in the array check if exists in the previously created AK\_list
4. if exists append attribute to the dynamic attributes char array
5. return pointer to char array with stored attribute/s

**Parameters**

|                 |   |
|-----------------|---|
| <i>*attribs</i> | projection attributes delimited by ";" (ATTR_DELIMITER) |
| <i>*tblName</i> | name of the table                                       |

**Returns**

filtered list of projection attributes as the AK\_list

**5.50.2.8 AK\_rel\_eq\_projection\_test()**

```
void AK_rel_eq_projection_test ( )
```

Function for testing rel\_eq\_selection.

**Author**

Dino Laktašić.

**Returns**

No return value

**5.50.2.9 AK\_rel\_eq\_remove\_duplicates()**

```
char* AK_rel_eq_remove_duplicates (
    char * attrs )
```

Function which removes duplicate attributes from attributes expresion.

**Author**

Dino Laktašić.

**Parameters**

|               |  |
|---------------|--|
| <i>*attrs</i> | attributes from which to remove duplicates |
|---------------|--|

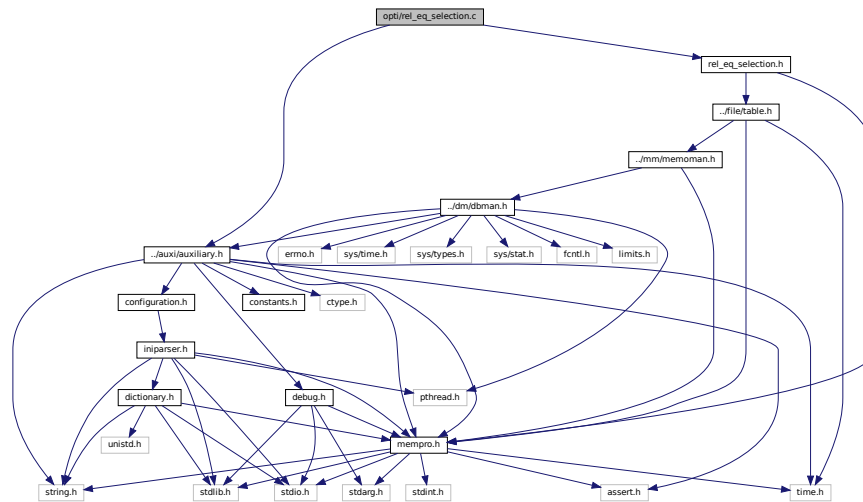
**Returns**

pointer to char array without duplicate attributes

**5.51 opti/rel\_eq\_selection.c File Reference**

```
#include "rel_eq_selection.h"
#include "../auxiliary/auxiliary.h"
```

Include dependency graph for `rel_eq_selection.c`:



## Functions

- `int AK_rel_eq_is_attr_subset (char *set, char *subset)`  
*Check if some set of attributes is subset of larger set.*
- `char * AK_rel_eq_get_attributes_char (char *tblName)`  
*Get attributes for a given table and store them to the char array.*
- `char * AK_rel_eq_cond_attributes (char *cond)`  
*Function for filtering and returning attributes from condition.*
- `int AK_rel_eq_share_attributes (char *set, char *subset)`  
*Check if two sets share one or more of it's attributes.*
- `struct list_node * AK_rel_eq_split_condition (char *cond)`  
*Check if selection can commute with theta-join or product (if working with conditions in infix format use this function instead - also remember to change code at the other places)*
- `struct list_node * AK_rel_eq_selection (struct list_node *list_rel_eq)`  
*Main function for generating RA expresion according to selection equivalence rules.*
- `void AK_print_rel_eq_selection (struct list_node *list_rel_eq)`  
*Function for printing struct `list_node` to the screen.*
- `void AK_rel_eq_selection_test ()`  
*Function for testing `rel_eq_selection`.*

### 5.51.1 Detailed Description

Provides functions for for relational equivalences in selection

### 5.51.2 Function Documentation



### 5.51.2.1 AK\_print\_rel\_eq\_selection()

```
void AK_print_rel_eq_selection (
    struct list_node * list_rel_eq )
```

Function for printing struct [list\\_node](#) to the screen.

#### Author

Dino Laktašić.

#### Parameters

|                     |  |
|---------------------|--|
| <i>*list_rel_eq</i> | RA expresion as the struct <a href="#">list_node</a> |
|---------------------|--|

#### Returns

void

### 5.51.2.2 AK\_rel\_eq\_cond\_attributes()

```
char* AK_rel_eq_cond_attributes (
    char * cond )
```

Function for filtering and returning attributes from condition.

#### Author

Dino Laktašić.

#### Parameters

|              |  |
|--------------|--|
| <i>*cond</i> | condition array that contains condition data |
|--------------|--|

#### Returns

pointer to array that contains attributes for a given condition

### 5.51.2.3 AK\_rel\_eq\_get\_attributes\_char()

```
char* AK_rel_eq_get_attributes_char (
    char * tblName )
```

Get attributes for a given table and store them to the char array.

**Author**

Dino Laktašić.

1. Get the number of attributes in a given table
2. If there is no attributes return NULL
3. Get the table header for a given table
4. Initialize struct `list_node`
5. For each attribute in table header, insert attribute in the array
6. Delimit each new attribute with ";" (ATTR\_DELIMITER)
7. return pointer to char array

**Parameters**

|                       |                   |
|-----------------------|-------------------|
| <code>*tblName</code> | name of the table |
|-----------------------|-------------------|

**Returns**

pointer to char array

**5.51.2.4 AK\_rel\_eq\_is\_attr\_subset()**

```
int AK_rel_eq_is_attr_subset (
    char * set,
    char * subset )
```

Check if some set of attributes is subset of larger set.

**Author**

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is reached
8. on loop exit return EXIT\_SUCCESS

**Parameters**

|                      |              |
|----------------------|--------------|
| <code>*set</code>    | set array    |
| <code>*subset</code> | subset array |

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**5.51.2.5 AK\_rel\_eq\_selection()**

```
struct list_node* AK_rel_eq_selection (
    struct list_node * list_rel_eq )
```

Main function for generating RA expression according to selection equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

|                     |  |
|---------------------|--|
| <i>*list_rel_eq</i> | RA expresion as the struct <a href="#">list_node</a> |
|---------------------|--|

**Returns**

optimised RA expresion as the struct [list\\_node](#)

**5.51.2.6 AK\_rel\_eq\_selection\_test()**

```
void AK_rel_eq_selection_test ( )
```

Function for testing rel\_eq\_selection.

**Author**

Dino Laktašić.

**Returns**

No return value

### 5.51.2.7 AK\_rel\_eq\_share\_attributes()

```
int AK_rel_eq_share_attributes (
    char * set,
    char * subset )
```

Check if two sets share one or more of it's attributes.

#### Author

Dino Laktašić.

1. If is empty set or subset returns EXIT\_FAILURE
2. For each attribute in one set check if there is same attribute in the second set
3. If there is the same attribute return EXIT\_SUCCESS
4. else remove unused pointers and return EXIT\_FAILURE

#### Parameters

|                |  |
|----------------|--|
| <i>*set</i>    | first set of attributes delimited by ";" (ATTR_DELIMITER)  |
| <i>*subset</i> | second set of attributes delimited by ";" (ATTR_DELIMITER) |

#### Returns

EXIT\_SUCCESS if set and subset share at least one attribute, else returns EXIT\_FAILURE

### 5.51.2.8 AK\_rel\_eq\_split\_condition()

```
struct list_node* AK_rel_eq_split_condition (
    char * cond )
```

Check if selection can commute with theta-join or product (if working with conditions in infix format use this function instead - also remember to change code at the other places)

#### Author

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else check if token differs from "OR", and if so, set id to 0, else make no changes to variable id
4. if token equals to "AND" and id equals to 1 append collected conds to result condition
5. else if token equals to "AND" and id equals to 0 discharge collected conds
6. else append token to collected data
7. When exits from loop if id greater then 0, append the last collected data to result
8. return pointer to char array that contains new condition for a given table

### Parameters

|                 |  |
|-----------------|--|
| <i>*cond</i>    | condition array that contains condition data |
| <i>*tblName</i> | name of the table                            |

## Returns

pointer to char array that contains new condition for a given table

**Author**

Dino Laktašić. Break conjunctive conditions to individual conditions (currently not used - commented in main AK\_rel\_eq\_selection function), it can be usefull in some optimization cases

1. For each delimited item (' AND ') insert item to the struct `list_node`
2. Remove unused pointers and return the conditions list

## Parameters

|               |                      |
|---------------|----------------------|
| * <i>cond</i> | condition expression |
|---------------|----------------------|

## Returns

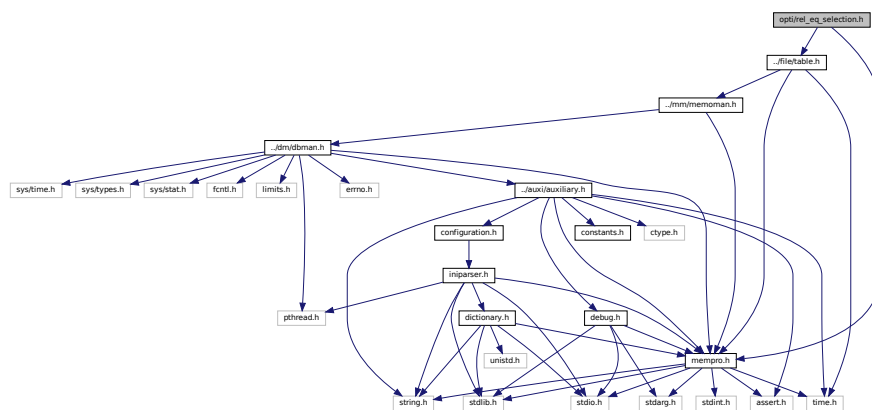
conditions list

## 5.52 opti/rel\_eq\_selection.h File Reference

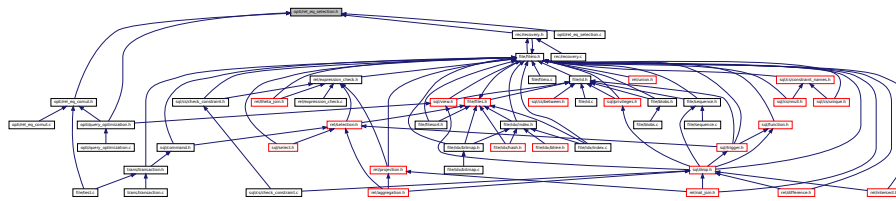
```
#include "../file/table.h"
```

```
#include "../auxi/mempro.h"
```

Include dependency graph for rel\_eq\_selection.h:



This graph shows which files directly or indirectly include this file:



## Functions

- int [AK\\_rel\\_eq\\_is\\_attr\\_subset](#) (char \*set, char \*subset)  
*Check if some set of attributes is subset of larger set.*
- char \* [AK\\_rel\\_eq\\_get\\_attributes\\_char](#) (char \*tblName)  
*Get attributes for a given table and store them to the char array.*
- char \* [AK\\_rel\\_eq\\_cond\\_attributes](#) (char \*cond)  
*Function for filtering and returning attributes from condition.*
- int [AK\\_rel\\_eq\\_share\\_attributes](#) (char \*set, char \*subset)  
*Check if two sets share one or more of it's attributes.*
- struct [list\\_node](#) \* [AK\\_rel\\_eq\\_split\\_condition](#) (char \*cond)  
*Check if selection can commute with theta-join or product (if working with conditions in infix format use this function instead - also remember to change code at the other places)*
- struct [list\\_node](#) \* [AK\\_rel\\_eq\\_selection](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Main function for generating RA expresion according to selection equivalence rules.*
- void [AK\\_print\\_rel\\_eq\\_selection](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Function for printing struct [list\\_node](#) to the screen.*
- void [AK\\_rel\\_eq\\_selection\\_test](#) ()  
*Function for testing [rel\\_eq\\_selection](#).*

### 5.52.1 Detailed Description

Header file that provides data structures for relational equivalences in selection

### 5.52.2 Function Documentation

#### 5.52.2.1 [AK\\_print\\_rel\\_eq\\_selection\(\)](#)

```
void AK_print_rel_eq_selection (
    struct list\_node * list_rel_eq )
```

Function for printing struct [list\\_node](#) to the screen.

#### Author

Dino Laktašić.

**Parameters**

|                     |  |
|---------------------|--|
| <i>*list_rel_eq</i> | RA expresion as the struct <a href="#">list_node</a> |
|---------------------|--|

**Returns**

void

**5.52.2.2 AK\_rel\_eq\_cond\_attributes()**

```
char* AK_rel_eq_cond_attributes (
    char * cond )
```

Function for filtering and returning attributes from condition.

**Author**

Dino Laktašić.

**Parameters**

|              |  |
|--------------|--|
| <i>*cond</i> | condition array that contains condition data |
|--------------|--|

**Returns**

pointer to array that contains attributes for a given condition

**5.52.2.3 AK\_rel\_eq\_get\_atrributes\_char()**

```
char* AK_rel_eq_get_atrributes_char (
    char * tblName )
```

Get attributes for a given table and store them to the char array.

**Author**

Dino Laktašić.

1. Get the number of attributes in a given table
2. If there is no attributes return NULL
3. Get the table header for a given table
4. Initialize struct [list\\_node](#)
5. For each attribute in table header, insert attribute in the array
6. Delimit each new attribute with ";" (ATTR\_DELIMITER)
7. return pointer to char array

**Parameters**

|                 |                   |
|-----------------|-------------------|
| <i>*tblName</i> | name of the table |
|-----------------|-------------------|

**Returns**

pointer to char array

**5.52.2.4 AK\_rel\_eq\_is\_attr\_subset()**

```
int AK_rel_eq_is_attr_subset (
    char * set,
    char * subset )
```

Check if some set of attributes is subset of larger set.

**Author**

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is reached
8. on loop exit return EXIT\_SUCCESS

**Parameters**

|                |              |
|----------------|--------------|
| <i>*set</i>    | set array    |
| <i>*subset</i> | subset array |

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**5.52.2.5 AK\_rel\_eq\_selection()**

```
struct list_node* AK_rel_eq_selection (
    struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to selection equivalence rules.



**Author**

Dino Laktašić.

**Parameters**

|                     |  |
|---------------------|--|
| <i>*list_rel_eq</i> | RA expresion as the struct <a href="#">list_node</a> |
|---------------------|--|

**Returns**

optimised RA expresion as the struct [list\\_node](#)

**5.52.2.6 AK\_rel\_eq\_selection\_test()**

```
void AK_rel_eq_selection_test ( )
```

Function for testing rel\_eq\_selection.

**Author**

Dino Laktašić.

**Returns**

No return value

**5.52.2.7 AK\_rel\_eq\_share\_attributes()**

```
int AK_rel_eq_share_attributes (
    char * set,
    char * subset )
```

Check if two sets share one or more of it's attributes.

**Author**

Dino Laktašić.

1. If is empty set or subset returns EXIT\_FAILURE
2. For each attribute in one set check if there is same attribute in the second set
3. If there is the same attribute return EXIT\_SUCCESS
4. else remove unused pointers and return EXIT\_FAILURE

**Parameters**

|                |  |
|----------------|--|
| <i>*set</i>    | first set of attributes delimited by ";" (ATTR_DELIMITER)  |
| <i>*subset</i> | second set of attributes delimited by ";" (ATTR_DELIMITER) |

**Returns**

EXIT\_SUCCESS if set and subset share at least one attribute, else returns EXIT\_FAILURE

**5.52.2.8 AK\_rel\_eq\_split\_condition()**

```
struct list_node* AK_rel_eq_split_condition (
    char * cond )
```

Check if selection can commute with theta-join or product (if working with conditions in infix format use this function instead - also remember to change code at the other places)

**Author**

Dino Laktašić.

1. For each token (delimited by " ") in selection condition first check if token represents attribute/s and is subset in the given table
2. If token is a subset set variable id to 1
3. else check if token differs from "OR", and if so, set id to 0, else make no changes to variable id
4. if token equals to "AND" and id equals to 1 append collected conds to result condition
5. else if token equals to "AND" and id equals to 0 discharge collected conds
6. else append token to collected data
7. When exits from loop if id greater then 0, append the last collected data to result
8. return pointer to char array that contains new condition for a given table

**Parameters**

|                 |  |
|-----------------|--|
| <i>*cond</i>    | condition array that contains condition data |
| <i>*tblName</i> | name of the table                            |

**Returns**

pointer to char array that contains new condition for a given table

**Author**

Dino Laktašić. Break conjunctive conditions to individual conditions (currently not used - commented in main AK\_rel\_eq\_selection function), it can be usefull in some optimization cases

1. For each delimited item (' AND ') insert item to the struct [list\\_node](#)
2. Remove unused pointers and return the conditions list

## Parameters

|              |                      |
|--------------|----------------------|
| <i>*cond</i> | condition expression |
|--------------|----------------------|

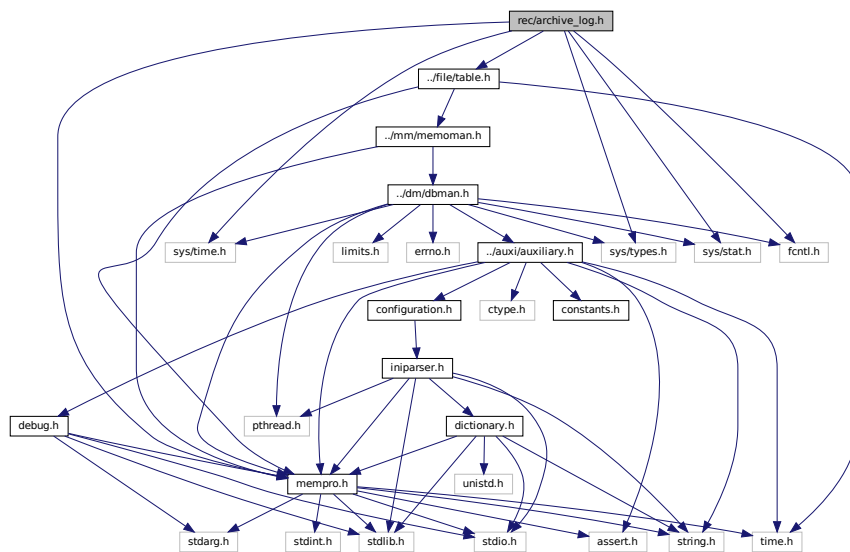
## Returns

conditions list

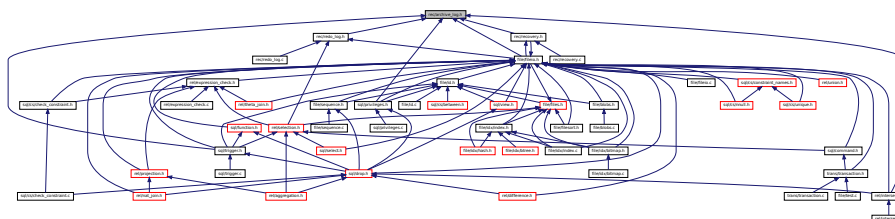
## 5.53 rec/archive\_log.h File Reference

```
#include "../file/table.h"
#include "sys/time.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "../aux/mempro.h"
```

Include dependency graph for archive\_log.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [AK\\_archive\\_log](#) (int sig)  
*Function for making archive log.*
- void [AK\\_empty\\_archive\\_log](#) ()
- char \* [AK\\_get\\_timestamp](#) ()  
*Function that returns the current timestamp.*

### 5.53.1 Detailed Description

Header file that provides data structures for archive logging

### 5.53.2 Function Documentation

#### 5.53.2.1 AK\_archive\_log()

```
void AK_archive_log (  
    int sig )
```

Function for making archive log.

Function creates a binary file that stores all commands that failed to execute with a number that shows the size of how many commands failed.

**Todo** this function takes static filename to store the failed commands, create certain logic that would make the function to use dynamic filename (this is partly implemented inside [AK\\_get\\_timestamp](#), but there is no logic that uses the last file when recovering - [recovery.c](#)) {link} [recovery.c](#) function test

#### Author

Dražen Bandić, update by Tomislav Turek

#### Returns

No return value

#### 5.53.2.2 AK\_empty\_archive\_log()

```
void AK_empty_archive_log ( )
```

Empties archive log

### 5.53.2.3 AK\_get\_timestamp()

```
char* AK_get_timestamp ( )
```

Function that returns the current timestamp.

This function returns the current timestamp that could be concatenated to a log file in future usages.

## Author

Dražen Bandić main logic, replaced by Tomislav Turek

**Todo** Think about this in the future when creating multiple binary recovery files. Implementation gives the timestamp, but is not used anywhere for now.

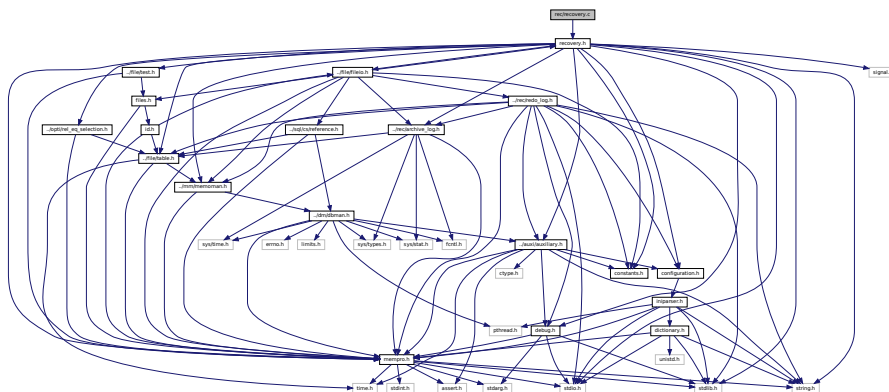
## Returns

char array in format day.month.year-hour:min:sec.usecu.log

## 5.54 rec/recovery.c File Reference

```
#include "recovery.h"
```

Include dependency graph for recovery.c:



## Functions

- void [AK\\_recover\\_archive\\_log](#) (char \*fileName)  
*Reads binary file where last commands were saved, and executes them.*
- void [AK\\_recovery\\_insert\\_row](#) (char \*table, char \*\*attributes)  
*Inserts a new row in table with attributes.*
- char \*\* [AK\\_recovery\\_tokenize](#) (char \*input, char \*delimiter, int valuesOrNot)  
*Tokenizes the input with the given delimiter and puts them in an double pointer structure (so we can execute an insert)*
- void [AK\\_recover\\_operation](#) (int sig)  
*Function that recovers and executes failed commands.*
- void [AK\\_recovery\\_test](#) ()  
*Function for recovery testing.*

## Variables

- short `grandfailure` = 0

### 5.54.1 Detailed Description

Provides recovery functions.

### 5.54.2 Function Documentation

#### 5.54.2.1 AK\_recover\_archive\_log()

```
void AK_recover_archive_log (
    char * fileName )
```

Reads binary file where last commands were saved, and executes them.

Function opens the recovery binary file and executes all commands that were saved inside the redo\_log structure

#### Author

Dražen Bandić, update by Tomislav Turek

#### Parameters

|                 |                           |
|-----------------|---------------------------|
| <i>fileName</i> | - name of the archive log |
|-----------------|---------------------------|

#### Returns

no value

#### 5.54.2.2 AK\_recover\_operation()

```
void AK_recover_operation (
    int sig )
```

Function that recovers and executes failed commands.

Function is called when SIGINT signal is sent to the system. All commands that are written to rec.bin file are recovered to the designated structure and then executed.

#### Author

Tomislav Turek

## Parameters

|            |   |
|------------|---|
| <i>sig</i> | required integer parameter for SIGINT handler functions |
|------------|---|

## 5.54.2.3 AK\_recovery\_insert\_row()

```
void AK_recovery_insert_row (
    char * table,
    char ** attributes )
```

Inserts a new row in table with attributes.

Function is given the table name with desired data that should be inserted inside. By using the table name, function retrieves table attributes names and their types which uses afterwards for `insert_data_test` function to insert data to designated table.

## Author

Dražen Bandić, updated by Tomislav Turek

## Parameters

|                   |                           |
|-------------------|---------------------------|
| <i>table</i>      | - table name to insert to |
| <i>attributes</i> | - attribute to insert     |

## Returns

no value

## 5.54.2.4 AK\_recovery\_test()

```
void AK_recovery_test ( )
```

Function for recovery testing.

Function does nothing while waiting a SIGINT signal (signal represents // doxygen @ for full description ??? system failure). Upon retrieving the signal it calls function `AK_recover_operation` which starts the recovery by building commands. To comply with the designated structure [AK\\_command\\_recovery\\_struct](#) // {link} to struct ??? it writes dummy commands to the file `log.log`

## Author

Tomislav Turek

#### 5.54.2.5 AK\_recovery\_tokenize()

```
char** AK_recovery_tokenize (
    char * input,
    char * delimiter,
    int valuesOrNot )
```

Tokenizes the input with the given delimiter and puts them in an double pointer structure (so we can execute an insert)

#### Author

Dražen Bandić

#### Parameters

|                    |  |
|--------------------|--|
| <i>input</i>       | - input to tokenize                      |
| <i>delimiter</i>   | - delimiter                              |
| <i>valuesOrNot</i> | - 1 if the input are values, 0 otherwise |

#### Returns

new double pointer structure with tokens

### 5.54.3 Variable Documentation

#### 5.54.3.1 grandfailure

```
short grandfailure = 0
```

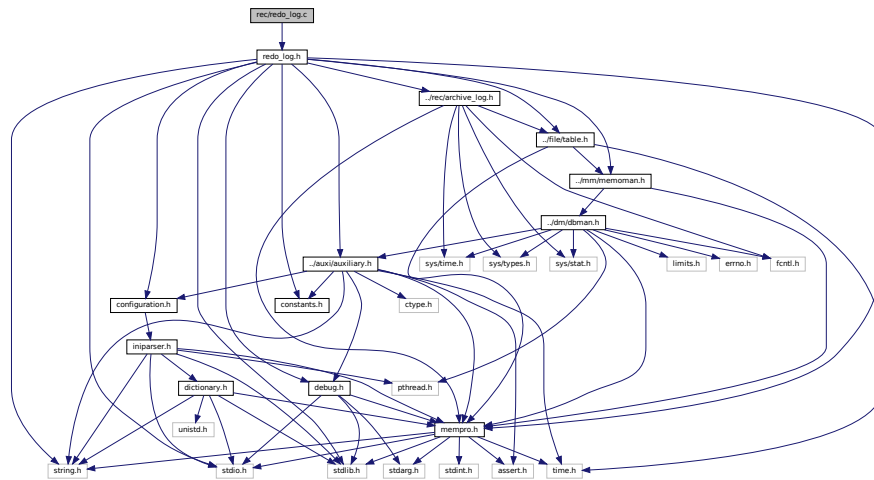
this variable flags if system failed

## 5.55 rec/redo\_log.c File Reference

```
#include "redo_log.h"
```



Include dependency graph for redo\_log.c:



## Functions

- int [AK\\_add\\_to\\_redolog](#) (int [command](#), struct [list\\_node](#) \*row\_root)  
Function adds new element to redolog.
- void [AK\\_redolog\\_commit](#) ()
- void [AK\\_printout\\_redolog](#) ()  
Function prints out the content of redolog memory.
- char \* [AK\\_check\\_attributes](#) (char \*attributes)  
Checks if the attribute contains '|', and if it does it replaces it with "|".

### 5.55.1 Detailed Description

Provides redolog functions.

### 5.55.2 Function Documentation

#### 5.55.2.1 AK\_add\_to\_redolog()

```
int AK_add_to_redolog (
    int command,
    struct list_node * row_root )
```

Function adds new element to redolog.

#### Author

Krunoslav Bilić updated by Dražen Bandić, second update by Tomislav Turek

#### Returns

EXIT\_FAILURE if not allocated memory for ispis, otherwise EXIT\_SUCCESS

### 5.55.2.2 AK\_check\_attributes()

```
char* AK_check_attributes (
    char * attributes )
```

Checks if the attribute contains '|', and if it does it replaces it with "\\|".

#### Author

Dražen Bandić

#### Returns

new attribute

### 5.55.2.3 AK\_printout\_redolog()

```
void AK_printout_redolog ( )
```

Function prints out the content of redolog memory.

#### Author

Krunoslav Bilić updated by Dražen Bandić, second update by Tomislav Turek

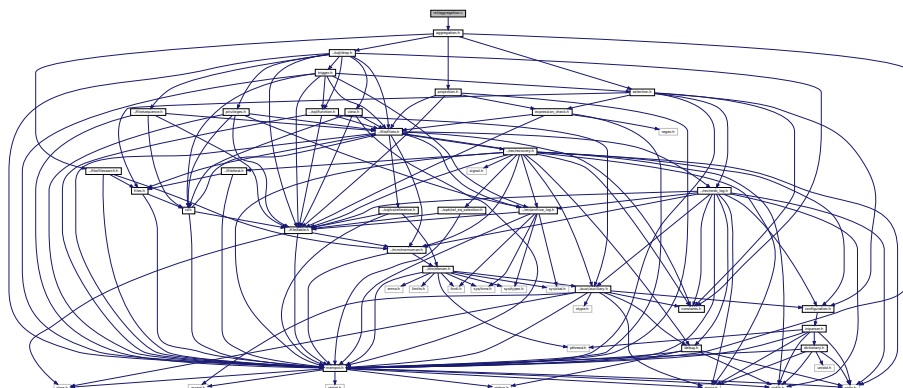
#### Returns

No return value.

## 5.56 rel/aggregation.c File Reference

```
#include "aggregation.h"
```

Include dependency graph for aggregation.c:



## Functions

- [search\\_result AK\\_search\\_unsorted](#) (char \*szRelation, [search\\_params](#) \*aspParams, int iNum\_search\_↵  
params)  
*Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH\_RANGE is inclusive. Only one value (or range) per attribute allowed - use [search\\_params.pData\\_lower](#) for SEARCH\_PARTICULAR. Supported types for SEARCH\_R↵  
ANGE: TYPE\_INT, TYPE\_FLOAT, TYPE\_NUMBER, TYPE\_DATE, TYPE\_DATETIME, TYPE\_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.*
- int [AK\\_header\\_size](#) ([AK\\_header](#) \*header)  
*Function calculates how many attributes there are in a header with while loop.*
- void [AK\\_agg\\_input\\_init](#) ([AK\\_agg\\_input](#) \*input)  
*Function initializes the input object for aggregation with init values.*
- int [AK\\_agg\\_input\\_add](#) ([AK\\_header](#) header, int agg\_task, [AK\\_agg\\_input](#) \*input)  
*Function adds a header with a task in input object for aggregation.*
- int [AK\\_agg\\_input\\_add\\_to\\_beginning](#) ([AK\\_header](#) header, int agg\_task, [AK\\_agg\\_input](#) \*input)  
*Function adds a header with a task on the beginning of the input object for aggregation so with for loop existing attributes and tasks are moved one place forward in input object.*
- void [AK\\_agg\\_input\\_fix](#) ([AK\\_agg\\_input](#) \*input)  
*This function is used to handle AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with value -1. While loop examines whether the task in array is equal to AGG\_TASK\_AVG. If so, AGG\_TA↵  
SK\_AVG\_COUNT is put on the beginning of input object. After that, AGG\_TASK\_AVG\_SUM is put on the beginning of input object.*
- int [AK\\_aggregation](#) ([AK\\_agg\\_input](#) \*input, char \*source\_table, char \*agg\_table)  
*Function aggregates a given table by given attributes. Firstly, AGG\_TASK\_AVG\_COUNT and AGG\_TASK\_AVG\_↵  
SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assigns the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed\_↵  
values array and results are put in new table.*
- void [Ak\\_aggregation\\_test](#) ()

### 5.56.1 Detailed Description

Provides functions for aggregation and grouping

### 5.56.2 Function Documentation

#### 5.56.2.1 AK\_agg\_input\_add()

```
int AK_agg_input_add (
    AK\_header header,
    int agg_task,
    AK\_agg\_input * input )
```

Function adds a header with a task in input object for aggregation.

#### Author

Dejan Frankovic

**Parameters**

|                 |  |
|-----------------|--|
| <i>header</i>   | a header that is being aggregated          |
| <i>agg_task</i> | the task which is to be done on the header |
| <i>input</i>    | the input object                           |

**Returns**

On success, returns EXIT\_SUCCESS, otherwise EXIT\_FAILURE

**5.56.2.2 AK\_agg\_input\_add\_to\_beginning()**

```
int AK_agg_input_add_to_beginning (
    AK_header header,
    int agg_task,
    AK_agg_input * input )
```

Function adds a header with a task on the beginning of the input object for aggregation so with for loop existing attributes and tasks are moved one place forward in input object.

**Author**

Dejan Frankovic

**Parameters**

|                 |  |
|-----------------|--|
| <i>header</i>   | a header that is being aggregated          |
| <i>agg_task</i> | the task which is to be done on the header |
| <i>input</i>    | the input object                           |

**Returns**

On success, returns EXIT\_SUCCESS, otherwise EXIT\_FAILURE

**5.56.2.3 AK\_agg\_input\_fix()**

```
void AK_agg_input_fix (
    AK_agg_input * input )
```

This function is used to handle AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with value -1. While loop examines whether the task in array is equal to AGG\_TASK\_AVG. If so, AGG\_TASK\_AVG\_COUNT is put on the beginning of input object. After that, AGG\_TASK\_AVG\_SUM is put on the begginig of input object.

**Author**

Dejan Frankovic

**Parameters**

|              |                  |
|--------------|------------------|
| <i>input</i> | the input object |
|--------------|------------------|

**Returns**

No return value

**5.56.2.4 AK\_agg\_input\_init()**

```
void AK_agg_input_init (
    AK_agg_input * input )
```

Function initializes the input object for aggregation whit init values.

**Author**

Dejan Frankovic

**Parameters**

|              |                  |
|--------------|------------------|
| <i>input</i> | the input object |
|--------------|------------------|

**Returns**

No return value

**5.56.2.5 AK\_aggregation()**

```
int AK_aggregation (
    AK_agg_input * input,
    char * source_table,
    char * agg_table )
```

Function aggregates a given table by given attributes. Firstly, AGG\_TASK\_AVG\_COUNT and AGG\_TASK\_AVG\_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assigns the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed\_values array and results are put in new table.

**Author**

Dejan Frankovic

**Parameters**

|                     |  |
|---------------------|--|
| <i>input</i>        | input object with list of attributes by which we aggregate and types of aggregations |
| <i>source_table</i> | - table name for the source table  |
| <i>agg_table</i>    | table name for aggregated table  |

**Returns**

EXIT\_SUCCESS if continues succesfully, when not EXIT\_ERROR

THIS SINGLE LINE BELOW (memcpy) is the purpose of ALL evil in the world! This line is the reason why test function prints one extra empty row with "nulls" at the end! Trust me! Comment it, and you will see - test function will not print extra row with nulls (but counts and averages in table will be all messed up!) After two days of hard research, I still have not found what is the reason behind printing extra row at the end! Fellow programmer, if you really really want to solve this issue, arm yourself with at least 2 liters of hot coffee!

What this line does? What is the purpose of this line in the universe? Well, fellow programmer, this line sets the initial count to 1. That means if name "Ivan" is found, it will have count of 1 because, well, that's the first Ivan that is found! If function finds another Ivan (which, actually, will happen), this part of code will not handle it (other part of code will).

That actually means that this little piece of code (this line below) only (and ONLY) sets count to 1! And besides that causes every other evil in the world. :O

P.S. The reason for that may be in linked list, or in AK\_insert\_row() You'll have to check every piece of AKDB code to find cause! I have found out that additional line is added when k == 25. There may be problem in linked lists or in AK\_insert\_row function or somewhere else. Who knows.

If I didn't handle that last row (which has one attribute of size 0), test would not pass!

Good luck, fellow programmer!

**5.56.2.6 Ak\_aggregation\_test()**

```
void Ak_aggregation_test ( )
```

checking results

This variable was added to handle bug described in this file.

**5.56.2.7 AK\_header\_size()**

```
int AK_header_size (
    AK_header * header )
```

Function calculates how many attributes there are in a header with while loop.

**Author**

Dejan Frankovic

## Parameters

|               |                |
|---------------|----------------|
| <i>header</i> | A header array |
|---------------|----------------|

## Returns

Number of attributes defined in header array

## 5.56.2.8 AK\_search\_unsorted()

```
search_result AK_search_unsorted (
    char * szRelation,
    search_params * aspParams,
    int iNum_search_params )
```

Searches through unsorted values of multiple attributes in a segment. Only tuples that are equal on all given attribute values are returned (A == 1 AND B == 7 AND ...). SEARCH\_RANGE is inclusive. Only one value (or range) per attribute allowed - use [search\\_params.pData\\_lower](#) for SEARCH\_PARTICULAR. Supported types for SEARCH\_RANGE: TYPE\_INT, TYPE\_FLOAT, TYPE\_NUMBER, TYPE\_DATE, TYPE\_DATETIME, TYPE\_TIME. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.

## Author

Miroslav Policki

## Parameters

|                           |                             |
|---------------------------|-----------------------------|
| <i>szRelation</i>         | relation name               |
| <i>aspParams</i>          | array of search parameters  |
| <i>iNum_search_params</i> | number of search parameters |

## Returns

[search\\_result](#) structure defined in [filesearch.h](#). Use [AK\\_deallocate\\_search\\_result](#) to deallocate.

iterate through all the blocks

count number of attributes in segment/relation

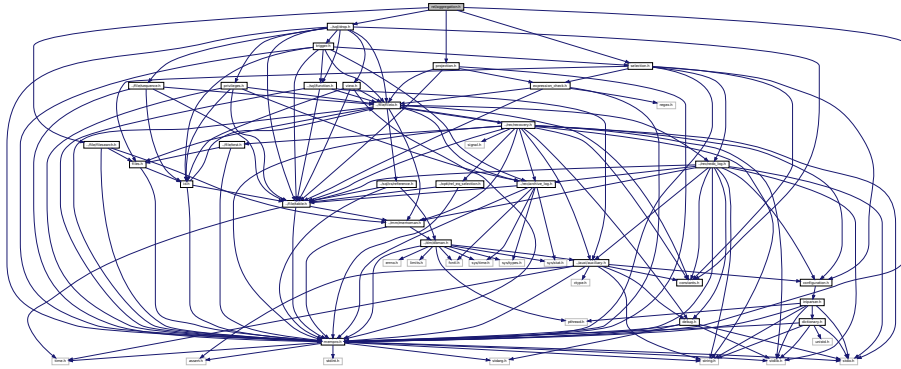
determine index of attributes on which search will be performed

if any of the provided attributes are not found in the relation, return empty result

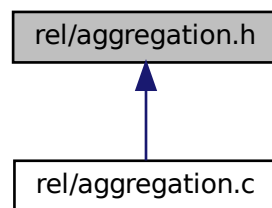
in every tuple, for all required attributes, compare attribute value with searched-for value and store matched tuple addresses

## 5.57 rel/aggregation.h File Reference

```
#include "selection.h"
#include "projection.h"
#include "../file/filesearch.h"
#include "../auxi/mempro.h"
#include "../sql/drop.h"
Include dependency graph for aggregation.h:
```



This graph shows which files directly or indirectly include this file:



### Classes

- struct [AK\\_agg\\_value](#)  
*Structure that contains attribute name, date and aggregation task associated.*
- struct [AK\\_agg\\_input](#)  
*Structure that contains attributes from table header, tasks for this table and counter value.*

### Macros

- `#define AGG_TASK_GROUP 1`
- `#define AGG_TASK_COUNT 2`
- `#define AGG_TASK_SUM 3`
- `#define AGG_TASK_MAX 4`
- `#define AGG_TASK_MIN 5`
- `#define AGG_TASK_AVG 6`
- `#define AGG_TASK_AVG_COUNT 10`
- `#define AGG_TASK_AVG_SUM 11`



## Functions

- int [AK\\_header\\_size](#) ([AK\\_header](#) \*)  
Function calculates how many attributes there are in a header with while loop.
- void [AK\\_agg\\_input\\_init](#) ([AK\\_agg\\_input](#) \*input)  
Function initializes the input object for aggregation with init values.
- int [AK\\_agg\\_input\\_add](#) ([AK\\_header](#) header, int agg\_task, [AK\\_agg\\_input](#) \*input)  
Function adds a header with a task in input object for aggregation.
- int [AK\\_agg\\_input\\_add\\_to\\_beginning](#) ([AK\\_header](#) header, int agg\_task, [AK\\_agg\\_input](#) \*input)  
Function adds a header with a task on the beginning of the input object for aggregation so with for loop existing attributes and tasks are moved one place forward in input object.
- void [AK\\_agg\\_input\\_fix](#) ([AK\\_agg\\_input](#) \*input)  
This function is used to handle AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with value -1. While loop examines whether the task in array is equal to AGG\_TASK\_AVG. If so, AGG\_TASK\_AVG\_COUNT is put on the beginning of input object. After that, AGG\_TASK\_AVG\_SUM is put on the beginning of input object.
- int [AK\\_aggregation](#) ([AK\\_agg\\_input](#) \*input, char \*source\_table, char \*agg\_table)  
Function aggregates a given table by given attributes. Firstly, AGG\_TASK\_AVG\_COUNT and AGG\_TASK\_AVG\_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assigns the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed\_values array and results are put in new table.
- void [Ak\\_aggregation\\_test](#) ()

### 5.57.1 Detailed Description

Header file that provides data structures for aggregation and grouping

### 5.57.2 Function Documentation

#### 5.57.2.1 AK\_agg\_input\_add()

```
int AK_agg_input_add (
    AK\_header header,
    int agg_task,
    AK\_agg\_input * input )
```

Function adds a header with a task in input object for aggregation.

#### Author

Dejan Frankovic

#### Parameters

|                 |  |
|-----------------|--|
| <i>header</i>   | a header that is being aggregated          |
| <i>agg_task</i> | the task which is to be done on the header |
| <i>input</i>    | the input object                           |

**Returns**

On success, returns EXIT\_SUCCESS, otherwise EXIT\_FAILURE

**5.57.2.2 AK\_agg\_input\_add\_to\_beginning()**

```
int AK_agg_input_add_to_beginning (
    AK_header header,
    int agg_task,
    AK_agg_input * input )
```

Function adds a header with a task on the beginning of the input object for aggregation so with for loop existing attributes and tasks are moved one place forward in input object.

**Author**

Dejan Frankovic

**Parameters**

|                 |  |
|-----------------|--|
| <i>header</i>   | a header that is being aggregated          |
| <i>agg_task</i> | the task which is to be done on the header |
| <i>input</i>    | the input object                           |

**Returns**

On success, returns EXIT\_SUCCESS, otherwise EXIT\_FAILURE

**5.57.2.3 AK\_agg\_input\_fix()**

```
void AK_agg_input_fix (
    AK_agg_input * input )
```

This function is used to handle AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with value -1. While loop examines whether the task in array is equal to AGG\_TASK\_AVG. If so, AGG\_TASK\_AVG\_COUNT is put on the beginning of input object. After that, AGG\_TASK\_AVG\_SUM is put on the beginning of input object.

**Author**

Dejan Frankovic

**Parameters**

|              |                  |
|--------------|------------------|
| <i>input</i> | the input object |
|--------------|------------------|

**Returns**

No return value

**5.57.2.4 AK\_agg\_input\_init()**

```
void AK_agg_input_init (
    AK_agg_input * input )
```

Function initializes the input object for aggregation with init values.

**Author**

Dejan Frankovic

**Parameters**

|              |                  |
|--------------|------------------|
| <i>input</i> | the input object |
|--------------|------------------|

**Returns**

No return value

**5.57.2.5 AK\_aggregation()**

```
int AK_aggregation (
    AK_agg_input * input,
    char * source_table,
    char * agg_table )
```

Function aggregates a given table by given attributes. Firstly, AGG\_TASK\_AVG\_COUNT and AGG\_TASK\_AVG\_SUM are put on the beginning of the input object. Then for loop iterates through input tasks and assigns the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed\_values array and results are put in new table.

**Author**

Dejan Frankovic

**Parameters**

|                     |  |
|---------------------|--|
| <i>input</i>        | input object with list of attributes by which we aggregate and types of aggregations |
| <i>source_table</i> | - table name for the source table  |
| <i>agg_table</i>    | table name for aggregated table  |

**Returns**

EXIT\_SUCCESS if continues succesfully, when not EXIT\_ERROR

THIS SINGLE LINE BELOW (memcpy) is the purpose of ALL evil in the world! This line is the reason why test function prints one extra empty row with "nulls" at the end! Trust me! Comment it, and you will see - test function will not print extra row with nulls (but counts and averages in table will be all messed up!) After two days of hard research, I still have not found what is the reason behind printing extra row at the end! Fellow programmer, if you really really want to solve this issue, arm yourself with at least 2 liters of hot coffee!

What this line does? What is the purpose of this line in the universe? Well, fellow programmer, this line sets the initial count to 1. That means if name "Ivan" is found, it will have count of 1 because, well, that's the first Ivan that is found! If function finds another Ivan (which, actually, will happen), this part of code will not handle it (other part of code will).

That actually means that this little piece of code (this line below) only (and ONLY) sets count to 1! And besides that causes every other evil in the world. :O

P.S. The reason for that may be in linked list, or in AK\_insert\_row() You'll have to check every piece of AKDB code to find cause! I have found out that additional line is added when k == 25. There may be problem in linked lists or in AK\_insert\_row function or somewhere else. Who knows.

If I didn't handle that last row (which has one attribute of size 0), test would not pass!

Good luck, fellow programmer!

**5.57.2.6 Ak\_aggregation\_test()**

```
void Ak_aggregation_test ( )
```

checking results

This variable was added to handle bug described in this file.

**5.57.2.7 AK\_header\_size()**

```
int AK_header_size (
    AK_header * header )
```

Function calculates how many attributes there are in a header with while loop.

**Author**

Dejan Frankovic

**Parameters**

|               |                |
|---------------|----------------|
| <i>header</i> | A header array |
|---------------|----------------|

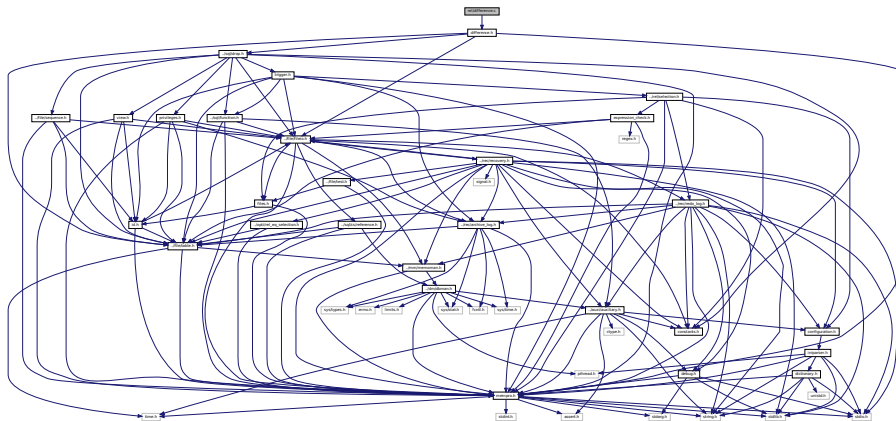
## Returns

Number of attributes defined in header array

## 5.58 rel/difference.c File Reference

```
#include "difference.h"
```

Include dependency graph for difference.c:



## Functions

- int [AK\\_difference](#) (char \*srcTable1, char \*srcTable2, char \*dstTable)

*Function to make difference of the two tables. Table addresses are get through names of tables. Specially start addresses are taken from them. They are used to allocate blocks for them. It is checked whether the tables have same table schemas. If not, it returns EXIT\_ERROR. New segment for result of difference operation is initialized. Function compares every block in extent of first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.*

- void [Ak\\_op\\_difference\\_test](#) ()

*Function for difference operator testing.*

### 5.58.1 Detailed Description

Provides functions for relational difference operation

### 5.58.2 Function Documentation

### 5.58.2.1 AK\_difference()

```
int AK_difference (
    char * srcTable1,
    char * srcTable2,
    char * dstTable )
```

Function to make difference of the two tables. Table addresses are get through names of tables. Specially start addresses are taken from them. They are used to allocate blocks for them. It is checked whether the tables have same table schemas. If not, it returns EXIT\_ERROR. New segment for result of difference operation is initialized. Function compares every block in extent of first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.

#### Author

Dino Laktašić

#### Parameters

|                  |                          |
|------------------|--------------------------|
| <i>srcTable1</i> | name of the first table  |
| <i>srcTable2</i> | name of the second table |
| <i>dstTable</i>  | name of the new table    |

#### Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

### 5.58.2.2 Ak\_op\_difference\_test()

```
void Ak_op_difference_test ( )
```

Function for difference operator testing.

#### Author

Dino Laktašić

## 5.59 rel/difference.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../aux/mempro.h"
```



### 5.59.2.1 AK\_difference()

```
int AK_difference (
    char * srcTable1,
    char * srcTable2,
    char * dstTable )
```

Function to make difference of the two tables. Table addresses are get through names of tables. Specially start addresses are taken from them. They are used to allocate blocks for them. It is checked whether the tables have same table schemas. If not, it returns EXIT\_ERROR. New segment for result of difference operation is initialized. Function compares every block in extent of first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.

#### Author

Dino Laktašić

#### Parameters

|                  |                          |
|------------------|--------------------------|
| <i>srcTable1</i> | name of the first table  |
| <i>srcTable2</i> | name of the second table |
| <i>dstTable</i>  | name of the new table    |

#### Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

### 5.59.2.2 Ak\_op\_difference\_test()

```
void Ak_op_difference_test ( )
```

Function for difference operator testing.

#### Author

Dino Laktašić

## 5.60 rel/expression\_check.c File Reference

```
#include "expression_check.h"
```



[illegible]

- int [AK\\_check\\_arithmetic\\_statement](#) (struct [list\\_node](#) \*el, const char \*op, const char \*a, const char \*b)  
*Function compares values according to their data type, checks arithmetic statement which is part of expression given in the function below. For every type of arithmetic operator, there is switch-case statement which examines type of el and casts void operands to this type.*
- char \* [AK\\_replace\\_wild\\_card](#) (const char \*s, char ch, const char \*repl)  
*Function that replaces character wildcard (%\_ \_) ch in string s with repl characters.*
- int [AK\\_check\\_regex\\_expression](#) (const char \*value, const char \*expression, int sensitive, int checkWildCard)  
*Function that evaluates regex expression on given string input.*
- int [AK\\_check\\_regex\\_operator\\_expression](#) (const char \*value, const char \*expression)  
*Function that evaluates regex expression on given string input.*
- int [AK\\_check\\_if\\_row\\_satisfies\\_expression](#) (struct [list\\_node](#) \*row\_root, struct [list\\_node](#) \*expr)  
*Function evaluates whether one record (row) satisfies logical expression. It goes through given row. If it comes to logical operator, it evaluates by itself. For arithmetic operators function [AK\\_check\\_arithmetic\\_statement\(\)](#) is called.*
- void [Ak\\_expression\\_check\\_test](#) ()

Provides functions for constraint checking used in selection and theta-join

Generated by Doxygen

### 5.60.2.1 AK\_check\_arithmetic\_statement()

```
int AK_check_arithmetic_statement (
    struct list\_node * el,
    const char * op,
    const char * a,
    const char * b )
```

Function compares values according to their data type, checks arithmetic statement which is part of expression given in the function below. For every type of arithmetic operator, there is switch-case statement which examines type of *el* and casts void operands to this type.

#### Author

Dino Laktašić, abstracted by Tomislav Mikulček, updated by Nikola Miljancic

#### Parameters

|            |   |
|------------|---|
| <i>el</i>  | list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation |
| <i>*op</i> | comparison operator   |
| <i>*a</i>  | left operand  |
| <i>*b</i>  | right operand   |

#### Returns

0 if arithmetic statement is false, 1 if arithmetic statement is true

### 5.60.2.2 AK\_check\_if\_row\_satisfies\_expression()

```
int AK_check_if_row_satisfies_expression (
    struct list\_node * row_root,
    struct list\_node * expr )
```

Function evaluates whether one record (row) satisfies logical expression. It goes through given row. If it comes to logical operator, it evaluates by itself. For arithmetic operators function [AK\\_check\\_arithmetic\\_statement\(\)](#) is called.

#### Author

Matija Šestak, updated by Dino Laktašić, Nikola Miljancic, abstracted by Tomislav Mikulček

#### Parameters

|                 |  |
|-----------------|--|
| <i>row_root</i> | beginning of the row that is to be evaluated         |
| <i>*expr</i>    | list with the logical expression in postfix notation |

**Returns**

0 if row does not satisfy, 1 if row satisfies expression

**5.60.2.3 Ak\_check\_regex\_expression()**

```
int Ak_check_regex_expression (
    const char * value,
    const char * expression,
    int sensitive,
    int checkWildCard )
```

Function that evaluates regex expression on given string input.

Leon Palać

**Parameters**

|                      |   |
|----------------------|---|
| <i>value</i>         | string value that must match regex expression                   |
| <i>expression</i>    | POSIX regex expression  |
| <i>checkWildCard</i> | replaces SQL wildcard to corresponding POSIX regex character    |
| <i>sensitive</i>     | case insensitive indicator 1-case sensitive,0- case insensitive |

**Returns**

0 if regex didnt match or syntax of regex is incorrect 1 if string matches corresponding regex expression

**5.60.2.4 Ak\_check\_regex\_operator\_expression()**

```
int Ak_check_regex_operator_expression (
    const char * value,
    const char * expression )
```

Function that evaluates regex expression on given string input.

Leon Palać

**Parameters**

|                   |   |
|-------------------|---|
| <i>value</i>      | string value that must match regex expression |
| <i>expression</i> | POSIX regex expression                        |

**Returns**

0 if regex didnt match or syntax of regex is incorrect 1 if string matches corresponding regex expression





**Parameters**

|            |   |
|------------|---|
| <i>el</i>  | list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation |
| <i>*op</i> | comparison operator   |
| <i>*a</i>  | left operand  |
| <i>*b</i>  | right operand   |

**Returns**

0 if arithmetic statement is false, 1 if arithmetic statement is true

**5.61.2.2 AK\_check\_if\_row\_satisfies\_expression()**

```
int AK_check_if_row_satisfies_expression (
    struct list_node * row_root,
    struct list_node * expr )
```

Function evaluates whether one record (row) satisfies logical expression. It goes through given row. If it comes to logical operator, it evaluates by itself. For arithmetic operators function [AK\\_check\\_arithmetic\\_statement\(\)](#) is called.

**Author**

Matija Šestak, updated by Dino Laktašić, Nikola Miljancic, abstracted by Tomislav Mikulček

**Parameters**

|                 |  |
|-----------------|--|
| <i>row_root</i> | beginning of the row that is to be evaluated         |
| <i>*expr</i>    | list with the logical expression in postfix notation |

**Returns**

0 if row does not satisfy, 1 if row satisfies expression

**5.61.2.3 Ak\_check\_regex\_expression()**

```
int Ak_check_regex_expression (
    const char * value,
    const char * expression,
    int sensitive,
    int checkWildcard )
```

Function that evaluates regex expression on given string input.

Leon Palaić

## Parameters

|                      |   |
|----------------------|---|
| <i>value</i>         | string value that must match regex expression                   |
| <i>expression</i>    | POSIX regex expression  |
| <i>checkWildCard</i> | replaces SQL wildcard to corresponding POSIX regex character    |
| <i>sensitive</i>     | case insensitive indicator 1-case sensitive,0- case insensitive |

## Returns

0 if regex didnt match or syntax of regex is incorrect 1 if string matches corresponding regex expression

## 5.61.2.4 Ak\_check\_regex\_operator\_expression()

```
int Ak_check_regex_operator_expression (
    const char * value,
    const char * expression )
```

Function that evaluates regex expression on given string input.

Leon Palać

## Parameters

|                   |   |
|-------------------|---|
| <i>value</i>      | string value that must match regex expression |
| <i>expression</i> | POSIX regex expression                        |

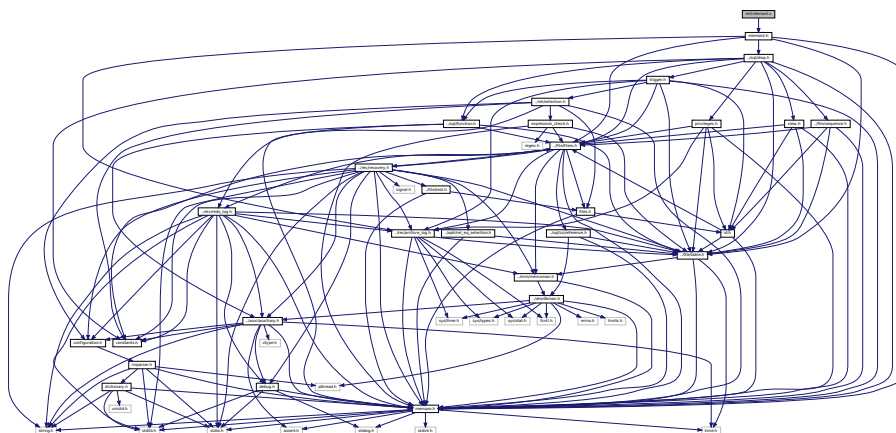
## Returns

0 if regex didnt match or syntax of regex is incorrect 1 if string matches corresponding regex expression

## 5.62 rel/intersect.c File Reference

```
#include "intersect.h"
```

Include dependency graph for intersect.c:



## Functions

- int [AK\\_intersect](#) (char \*srcTable1, char \*srcTable2, char \*dstTable)  
*Function to make intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)*
- void [Ak\\_op\\_intersect\\_test](#) ()  
*Function for intersect operator testing.*

### 5.62.1 Detailed Description

Provides functions for relational intersect operation

### 5.62.2 Function Documentation

#### 5.62.2.1 AK\_intersect()

```
int AK_intersect (
    char * srcTable1,
    char * srcTable2,
    char * dstTable )
```

Function to make intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)

#### Author

Dino Laktašić

#### Parameters

|                  |                          |
|------------------|--------------------------|
| <i>srcTable1</i> | name of the first table  |
| <i>srcTable2</i> | name of the second table |
| <i>dstTable</i>  | name of the new table    |

#### Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

#### 5.62.2.2 Ak\_op\_intersect\_test()

```
void Ak_op_intersect_test ( )
```

Function for intersect operator testing.



## Author

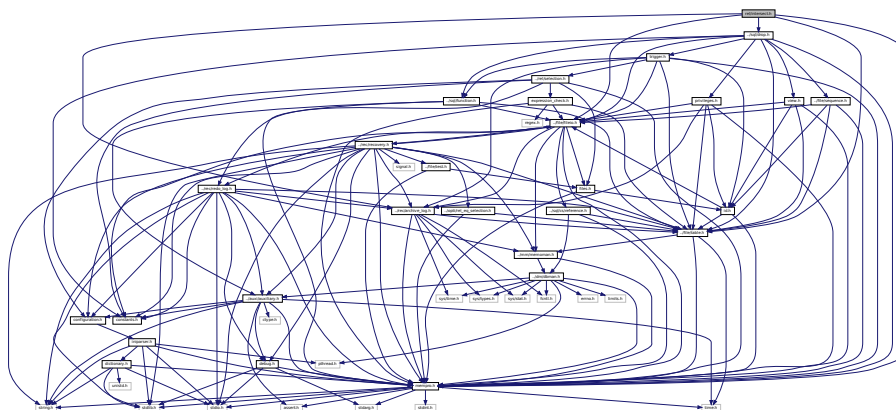
Dino Laktašić

## Returns

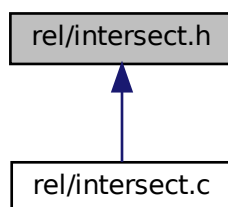
No return value

## 5.63 rel/intersect.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../rec/archive_log.h"
#include "../auxi/mempro.h"
#include "../sql/drop.h"
Include dependency graph for intersect.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [intersect\\_attr](#)

*Structure defines intersect attribute.*

## Functions

- int [AK\\_intersect](#) (char \*srcTable1, char \*srcTable2, char \*dstTable)  
*Function to make intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)*
- void [Ak\\_op\\_intersect\\_test](#) ()  
*Function for intersect operator testing.*

### 5.63.1 Detailed Description

Provides data structures for relational intersect operation

### 5.63.2 Function Documentation

#### 5.63.2.1 AK\_intersect()

```
int AK_intersect (
    char * srcTable1,
    char * srcTable2,
    char * dstTable )
```

Function to make intersect of the two tables. Intersect is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (intersect)

#### Author

Dino Laktašić

#### Parameters

|                  |                          |
|------------------|--------------------------|
| <i>srcTable1</i> | name of the first table  |
| <i>srcTable2</i> | name of the second table |
| <i>dstTable</i>  | name of the new table    |

#### Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

#### 5.63.2.2 Ak\_op\_intersect\_test()

```
void Ak_op_intersect_test ( )
```

Function for intersect operator testing.

**Author**

Dino Laktašić

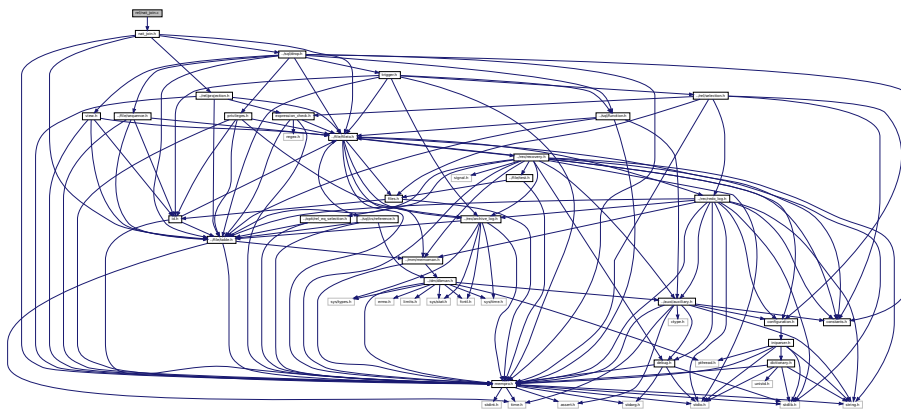
**Returns**

No return value

**5.64 rel/nat\_join.c File Reference**

```
#include "nat_join.h"
```

Include dependency graph for nat\_join.c:

**Functions**

- void [AK\\_create\\_join\\_block\\_header](#) (int table\_address1, int table\_address2, char \*new\_table, struct [list\\_node](#) \*att)  
*Function to make header for the new table and call the function to create the segment.*
- void [AK\\_merge\\_block\\_join](#) (struct [list\\_node](#) \*row\_root, struct [list\\_node](#) \*row\_root\_insert, [AK\\_block](#) \*temp\_block, char \*new\_table)  
*Function searches the second block and when found matches with the first one makes a join and write row to join table.*
- void [AK\\_copy\\_blocks\\_join](#) ([AK\\_block](#) \*tbl1\_temp\_block, [AK\\_block](#) \*tbl2\_temp\_block, struct [list\\_node](#) \*att, char \*new\_table)  
*Function iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.*
- int [AK\\_join](#) (char \*srcTable1, char \*srcTable2, char \*dstTable, struct [list\\_node](#) \*att)  
*Function to make nat\_join between two tables on some attributes.*
- void [AK\\_op\\_join\\_test](#) ()  
*Function for natural join testing.*

**5.64.1 Detailed Description**

Provides functions for relational natural join operation

## 5.64.2 Function Documentation

### 5.64.2.1 AK\_copy\_blocks\_join()

```
void AK_copy_blocks_join (
    AK_block * tbl1_temp_block,
    AK_block * tbl2_temp_block,
    struct list_node * att,
    char * new_table )
```

Function iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.

#### Author

Matija Novak, optimized, and updated to work with AK\_list by Dino Laktašić

#### Parameters

|                        |                                      |
|------------------------|--------------------------------------|
| <i>tbl1_temp_block</i> | block of the first table             |
| <i>tbl2_temp_block</i> | block of the second join table       |
| <i>att</i>             | attributes on which we make nat_join |
| <i>new_table</i>       | name of the nat_join table           |

#### Returns

No return value

### 5.64.2.2 AK\_create\_join\_block\_header()

```
void AK_create_join_block_header (
    int table_address1,
    int table_address2,
    char * new_table,
    struct list_node * att )
```

Function to make header for the new table and call the function to create the segment.

#### Author

Matija Novak, optimized, and updated to work with AK\_list by Dino Laktašić

#### Parameters

|                       |  |
|-----------------------|--|
| <i>table_address1</i> | address of the block of the first table  |
| <i>table_address2</i> | address of the block of the second table |
| <i>new_table</i>      | name of the join table                   |
| <i>att_root</i>       | ttributes on which we make nat_join      |

**Returns**

No return value

**5.64.2.3 AK\_join()**

```
int AK_join (
    char * srcTable1,
    char * srcTable2,
    char * dstTable,
    struct list_node * att )
```

Function to make nat\_join between two tables on some attributes.

**Author**

Matija Novak, updated to work with AK\_list and support cacheing by Dino Laktašić

**Parameters**

|                  |                                      |
|------------------|--------------------------------------|
| <i>srcTable1</i> | name of the first table to join      |
| <i>srcTable2</i> | name of the second table to join     |
| <i>att</i>       | attributes on which we make nat_join |
| <i>dstTable</i>  | name of the nat_join table           |

**Returns**

if success returns EXIT\_SUCCESS

**5.64.2.4 AK\_merge\_block\_join()**

```
void AK_merge_block_join (
    struct list_node * row_root,
    struct list_node * row_root_insert,
    AK_block * temp_block,
    char * new_table )
```

Function searches the second block and when found matches with the first one makes a join and write row to join table.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

|                        |  |
|------------------------|--|
| <i>row_root</i>        | - list of values from the first table to be marged with table2           |
| <i>row_root_insert</i> | - list of values from the first table to be inserted into nat_join table |
| <i>temp_block</i>      | - block from the second table to be merged                               |
| <i>new_table</i>       | - name of the nat_join table   |

**Returns**

No return value

**5.64.2.5 AK\_op\_join\_test()**

```
void AK_op_join_test ( )
```

Function for natural join testing.

**Author**

Matija Novak

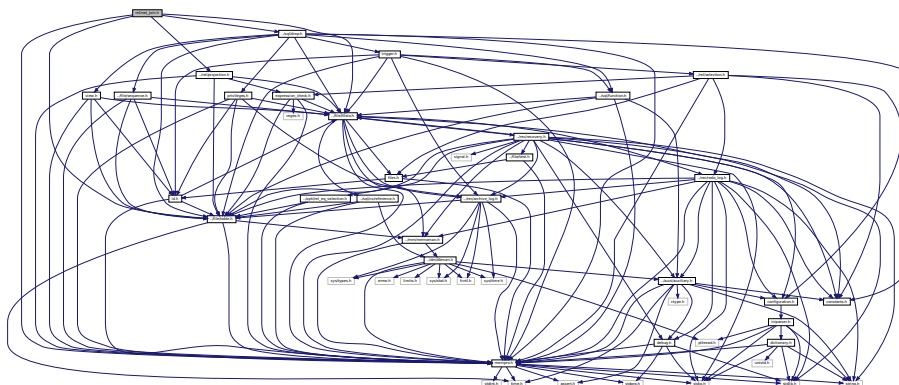
**Returns**

No return value

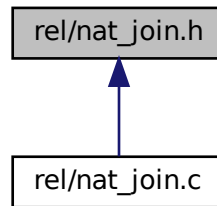
**5.65 rel/nat\_join.h File Reference**

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../rel/projection.h"
#include "../auxi/mempro.h"
#include "../sql/drop.h"
```

Include dependency graph for nat\_join.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [AK\\_create\\_join\\_block\\_header](#) (int table\_address1, int table\_address2, char \*new\_table, struct [list\\_node](#) \*att)  
*Function to make header for the new table and call the function to create the segment.*
- void [AK\\_merge\\_block\\_join](#) (struct [list\\_node](#) \*row\_root, struct [list\\_node](#) \*row\_root\_insert, [AK\\_block](#) \*temp\_block, char \*new\_table)  
*Function searches the second block and when found matches with the first one makes a join and write row to join table.*
- void [AK\\_copy\\_blocks\\_join](#) ([AK\\_block](#) \*tbl1\_temp\_block, [AK\\_block](#) \*tbl2\_temp\_block, struct [list\\_node](#) \*att, char \*new\_table)  
*Function iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.*
- int [AK\\_join](#) (char \*srcTable1, char \*srcTable2, char \*dstTable, struct [list\\_node](#) \*att)  
*Function to make nat\_join between two tables on some attributes.*
- void [AK\\_op\\_join\\_test](#) ()  
*Function for natural join testing.*

### 5.65.1 Detailed Description

Header file that provides data structures for relational natural join operation

### 5.65.2 Function Documentation

### 5.65.2.1 AK\_copy\_blocks\_join()

```
void AK_copy_blocks_join (
    AK_block * tbl1_temp_block,
    AK_block * tbl2_temp_block,
    struct list_node * att,
    char * new_table )
```

Function iterates through block of the first table and copies data that needs for join, then it calls a merge function to merge with the second table.

#### Author

Matija Novak, optimized, and updated to work with AK\_list by Dino Laktašić

#### Parameters

|                        |                                      |
|------------------------|--------------------------------------|
| <i>tbl1_temp_block</i> | block of the first table             |
| <i>tbl2_temp_block</i> | block of the second join table       |
| <i>att</i>             | attributes on which we make nat_join |
| <i>new_table</i>       | name of the nat_join table           |

#### Returns

No return value

### 5.65.2.2 AK\_create\_join\_block\_header()

```
void AK_create_join_block_header (
    int table_address1,
    int table_address2,
    char * new_table,
    struct list_node * att )
```

Function to make header for the new table and call the function to create the segment.

#### Author

Matija Novak, optimized, and updated to work with AK\_list by Dino Laktašić

#### Parameters

|                       |  |
|-----------------------|--|
| <i>table_address1</i> | address of the block of the first table  |
| <i>table_address2</i> | address of the block of the second table |
| <i>new_table</i>      | name of the join table                   |
| <i>att_root</i>       | tributes on which we make nat_join       |



## Returns

No return value

### 5.65.2.3 AK\_join()

```
int AK_join (
    char * srcTable1,
    char * srcTable2,
    char * dstTable,
    struct list_node * att )
```

Function to make nat\_join between two tables on some attributes.

## Author

Matija Novak, updated to work with AK\_list and support cacheing by Dino Laktašić

## Parameters

|                  |                                      |
|------------------|--------------------------------------|
| <i>srcTable1</i> | name of the first table to join      |
| <i>srcTable2</i> | name of the second table to join     |
| <i>att</i>       | attributes on which we make nat_join |
| <i>dstTable</i>  | name of the nat_join table           |

## Returns

if success returns EXIT\_SUCCESS

### 5.65.2.4 AK\_merge\_block\_join()

```
void AK_merge_block_join (
    struct list_node * row_root,
    struct list_node * row_root_insert,
    AK_block * temp_block,
    char * new_table )
```

Function searches the second block and when found matches with the first one makes a join and write row to join table.

## Author

Matija Novak, updated by Dino Laktašić

**Parameters**

|                        |  |
|------------------------|--|
| <i>row_root</i>        | - list of values from the first table to be marged with table2           |
| <i>row_root_insert</i> | - list of values from the first table to be inserted into nat_join table |
| <i>temp_block</i>      | - block from the second table to be merged                               |
| <i>new_table</i>       | - name of the nat_join table   |

**Returns**

No return value

**5.65.2.5 AK\_op\_join\_test()**

```
void AK_op_join_test ( )
```

Function for natural join testing.

**Author**

Matija Novak

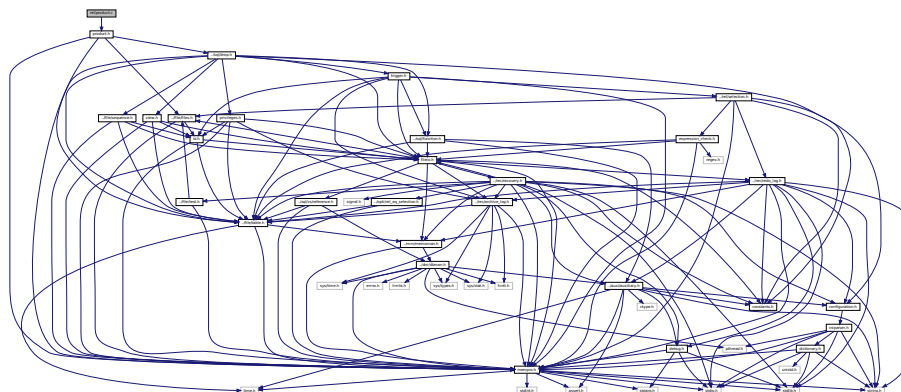
**Returns**

No return value

**5.66 rel/product.c File Reference**

```
#include "product.h"
```

Include dependency graph for product.c:



## Functions

- int [AK\\_product](#) (char \*srcTable1, char \*srcTable2, char \*dstTable)  
*Function to make product of two tables.*
- void [AK\\_op\\_product\\_test](#) ()  
*Function for product operator testing.*

### 5.66.1 Detailed Description

Provides functions for relational product operation

### 5.66.2 Function Documentation

#### 5.66.2.1 AK\_op\_product\_test()

```
void AK_op_product_test ( )
```

Function for product operator testing.

#### Author

Dino Laktašić

How does this test work? First, it reads all cells from both of the tables, employee and department. After that, it reads all cells from product\_test table and compares the data.

Reading data from first two tables (employee and department)

Now reading data from product table and comparing it to the data in first two tables

#### 5.66.2.2 AK\_product()

```
int AK_product (
    char * srcTable1,
    char * srcTable2,
    char * dstTable )
```

Function to make product of two tables.

#### Author

Dino Laktašić

#### Parameters

|                  |                           |
|------------------|---------------------------|
| <i>srcTable1</i> | name of the first table   |
| <i>srcTable2</i> | name of the second table  |
| <i>dstTable</i>  | name of the product table |

**Returns**

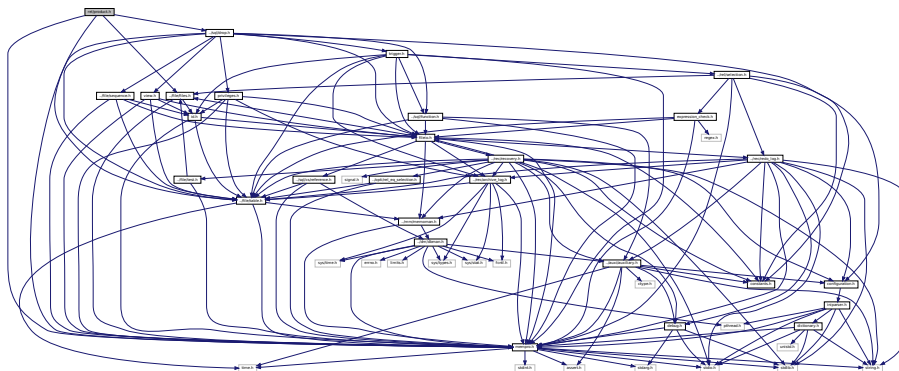
if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

Product procedure Going through one table, and for each row in it, going through another table, and joining rows that way!

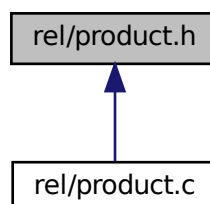
Sorry about indentations, but it is necessary to do it this way, until someone creates function to iterate through table rows (which would be pretty neat, btw.) At this level of code, we have access to rows from the first and from the second table. Then we do it this way: for each row in first table, read row from second table. And concatenate them! Since we have loop hierarchy here, each row from first table will be concatenated with each row from second table. End of story! Let's get back to work... BTW. Please comment your code in the future. It is a lot easier when someone explains to you what he or she was thinking that moment. Write comments in english. Write 'em in croatian. It does not matter! Just explain yourself! And share the idea about comments among others, please. Thank you!

## 5.67 rel/product.h File Reference

```
#include "../file/table.h"
#include "../file/files.h"
#include "../aux/mempro.h"
#include "../sql/drop.h"
Include dependency graph for product.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- int [AK\\_product](#) (char \*srcTable1, char \*srcTable2, char \*dstTable)  
*Function to make product of two tables.*
- void [AK\\_op\\_product\\_test](#) ()  
*Function for product operator testing.*

### 5.67.1 Detailed Description

Header file that provides data structures for relational product operation

### 5.67.2 Function Documentation

#### 5.67.2.1 AK\_op\_product\_test()

```
void AK_op_product_test ( )
```

Function for product operator testing.

#### Author

Dino Laktašić

How does this test work? First, it reads all cells from both of the tables, employee and department. After that, it reads all cells from product\_test table and compares the data.

Reading data from first two tables (employee and department)

Now reading data from product table and comparing it to the data in first two tables

#### 5.67.2.2 AK\_product()

```
int AK_product (
    char * srcTable1,
    char * srcTable2,
    char * dstTable )
```

Function to make product of two tables.

#### Author

Dino Laktašić

#### Parameters

|                  |                           |
|------------------|---------------------------|
| <i>srcTable1</i> | name of the first table   |
| <i>srcTable2</i> | name of the second table  |
| <i>dstTable</i>  | name of the product table |

## Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

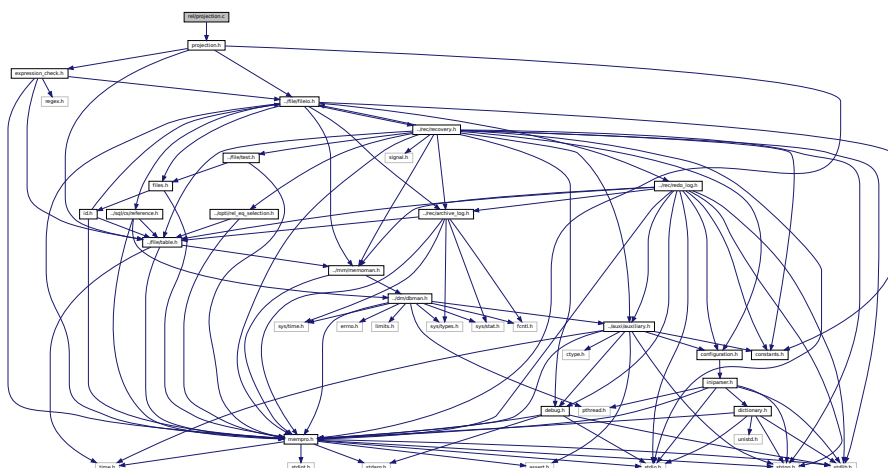
Product procedure Going through one table, and for each row in it, going through another table, and joining rows that way!

Sorry about indentations, but it is necessary to do it this way, until someone creates function to iterate through table rows (which would be pretty neat, btw.) At this level of code, we have access to rows from the first and from the second table. Then we do it this way: for each row in first table, read row from second table. And concatenate them! Since we have loop hierarchy here, each row from first table will be concatenated with each row from second table. End of story! Let's get back to work... BTW. Please comment your code in the future. It is a lot easier when someone explains to you what he or she was thinking that moment. Write comments in english. Write 'em in croatian. It does not matter! Just explain yourself! And share the idea about comments among others, please. Thank you!

## 5.68 rel/projection.c File Reference

```
#include "projection.h"
```

Include dependency graph for projection.c:



## Functions

- void [AK\\_temp\\_create\\_table](#) (char \*table, [AK\\_header](#) \*header, int type\_segment)  
*Temporaly function to create table, and insert entry to the system\_relation catalog.*
- void [AK\\_create\\_block\\_header](#) (int old\_block, char \*dstTable, struct [list\\_node](#) \*att)  
*Function to create a new header for the projection table.*
- char \* [AK\\_get\\_operator](#) (char \*exp)  
*Function that fetches arithmetics operator.*
- void [removeSubstring](#) (char \*s, const char \*toremove)  
*Function that removes specified part of string.*
- int [AK\\_determine\\_header\\_type](#) (int a, int b)  
*Determines new header type.*
- char \* [AK\\_create\\_header\\_name](#) (char \*first, char \*second, char \*operator)  
*Creates new header name from passed operand names and operator.*

- void [AK\\_copy\\_block\\_projection](#) ([AK\\_block](#) \*old\_block, struct [list\\_node](#) \*att, char \*dstTable, struct [list\\_node](#) \*expr)  
*Function for copying the data from old table block to the new projection table.*
- char \* [AK\\_perform\\_operatrion](#) (char \*op, struct [AK\\_operand](#) \*ab, struct [AK\\_operand](#) \*bb, int type)  
*Performes arithmetics operation on operand data.*
- int [AK\\_projection](#) (char \*srcTable, char \*dstTable, struct [list\\_node](#) \*att, struct [list\\_node](#) \*expr)  
*Function makes a projection of some table.*
- void [AK\\_op\\_projection\\_test](#) ()  
*Function for projection operator testing.*

### 5.68.1 Detailed Description

Provides functions for relational projection operation

### 5.68.2 Function Documentation

#### 5.68.2.1 AK\_copy\_block\_projection()

```
void AK_copy_block_projection (
    AK\_block * old_block,
    struct list\_node * att,
    char * dstTable,
    struct list\_node * expr )
```

Function for copying the data from old table block to the new projection table.

#### Author

Matija Novak, rewrited and optimized by Dino Laktašić to support AK\_list

#### Parameters

|                  |  |
|------------------|--|
| <i>old_block</i> | block from which we copy data  |
| <i>dstTable</i>  | name of the new table  |
| <i>att</i>       | list of the attributes which should the projeciton table contain No return value |

#### 5.68.2.2 AK\_create\_block\_header()

```
void AK_create_block_header (
    int old_block,
    char * dstTable,
    struct list\_node * att )
```

Function to create a new header for the projection table.

**Author**

Matija Novak, rewrited and optimized by Dino Laktašić to support AK\_list

**Parameters**

|                      |  |
|----------------------|--|
| <i>old_block_add</i> | address of the block from which we copy headers we need          |
| <i>dstTable</i>      | name of the new table  |
| <i>att</i>           | list of the attributes which should the projeciton table contain |

**Returns**

No return value

**5.68.2.3 AK\_create\_header\_name()**

```
char* AK_create_header_name (
    char * first,
    char * second,
    char * operator )
```

Creates new header name from passed operand names and operator.

Leon Palačić

**Parameters**

|                 |              |
|-----------------|--------------|
| <i>first</i>    | operand name |
| <i>second</i>   | operand name |
| <i>operator</i> |              |

**Returns**

new sequence of charachters

**5.68.2.4 AK\_determine\_header\_type()**

```
int AK_determine_header_type (
    int a,
    int b )
```

Determines new header type.

Leon Palačić



**Parameters**

|          |              |
|----------|--------------|
| <i>a</i> | operand type |
| <i>b</i> | operand type |

**Returns**

header type

**5.68.2.5 AK\_get\_operator()**

```
char* AK_get_operator (
    char * exp )
```

Function that fetches arithmetics operator.

Leon Palaić

**Parameters**

|            |                         |
|------------|-------------------------|
| <i>exp</i> | input expression string |
|------------|-------------------------|

**Returns**

operator

**5.68.2.6 AK\_op\_projection\_test()**

```
void AK_op_projection_test ( )
```

Function for projection operator testing.

**Author**

Dino Laktašić

**Returns**

No return value

#### 5.68.2.7 AK\_perform\_operatrion()

```
char* AK_perform_operatrion (
    char * op,
    struct AK_operand * ab,
    struct AK_operand * bb,
    int type )
```

Performes arithmetics operation on operand data.

Leon Palaic

**Parameters**

|           |                |
|-----------|----------------|
| <i>ab</i> | first operand  |
| <i>bb</i> | second operand |
| <i>op</i> | operator       |

**Returns**

result of arithmetics

**5.68.2.8 AK\_projection()**

```
int AK_projection (
    char * srcTable,
    char * dstTable,
    struct list_node * att,
    struct list_node * expr )
```

Function makes a projection of some table.

**Author**

Matija Novak, rewrited and optimized by Dino Laktašić, now support cacheing

**Parameters**

|                 |   |
|-----------------|---|
| <i>att</i>      | - list of atributes on which we make projection |
| <i>dstTable</i> | table name for projection table                 |

**Returns**

EXIT\_SUCCESS if continues succesfully, when not EXIT\_ERROR

**5.68.2.9 AK\_temp\_create\_table()**

```
void AK_temp_create_table (
    char * table,
    AK_header * header,
    int type_segment )
```

Temporaly function to create table, and insert entry to the system\_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

|                     |  |
|---------------------|--|
| <i>table</i>        | table name                                 |
| <i>header</i>       | <a href="#">AK_header</a> of the new table |
| <i>type_segment</i> | type of the new segment                    |

**Returns**

No return value

**5.68.2.10 removeSubstring()**

```
void removeSubstring (
    char * s,
    const char * toremove )
```

Function that removes specified part of string.

Leon Palać

**Parameters**

|                 |               |
|-----------------|---------------|
| <i>s</i>        | input string  |
| <i>toremove</i> | remove string |

**Returns**

new sequence of characters

**5.69 rel/projection.h File Reference**

```
#include "expression_check.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../aux/mempro.h"
```



- int [AK\\_determine\\_header\\_type](#) (int a, int b)  
*Determines new header type.*
- char \* [AK\\_create\\_header\\_name](#) (char \*first, char \*operator, char \*second)  
*Creates new header name from passed operand names and operator.*
- char \* [AK\\_perform\\_operatrion](#) (char \*op, struct [AK\\_operand](#) \*a, struct [AK\\_operand](#) \*b, int type)  
*Performs arithmetics operation on operand data.*
- void [removeSubstring](#) (char \*s, const char \*toremove)  
*Function that removes specified part of string.*
- void [AK\\_op\\_projection\\_test](#) ()  
*Function for projection operator testing.*
- char \* [AK\\_get\\_operator](#) (char \*exp)  
*Function that fetches arithmetics operator.*

### 5.69.1 Detailed Description

Header file that provides data structures for relational projection operation

### 5.69.2 Function Documentation

#### 5.69.2.1 [AK\\_copy\\_block\\_projection\(\)](#)

```
void AK_copy_block_projection (
    AK\_block * old_block,
    struct list\_node * att,
    char * dstTable,
    struct list\_node * expr )
```

Function for copying the data from old table block to the new projection table.

#### Author

Matija Novak, rewrited and optimized by Dino Laktašić to support [AK\\_list](#)

#### Parameters

|                  |  |
|------------------|--|
| <i>old_block</i> | block from which we copy data  |
| <i>dstTable</i>  | name of the new table  |
| <i>att</i>       | list of the attributes which should the projeciton table contain No return value |

#### 5.69.2.2 [AK\\_create\\_block\\_header\(\)](#)

```
void AK_create_block_header (
    int old_block,
```

```
char * dstTable,  
struct list_node * att )
```

Function to create a new header for the projection table.

#### Author

Matija Novak, rewrited and optimized by Dino Laktašić to support AK\_list

#### Parameters

|                      |  |
|----------------------|--|
| <i>old_block_add</i> | address of the block from which we copy headers we need          |
| <i>dstTable</i>      | name of the new table  |
| <i>att</i>           | list of the attributes which should the projeciton table contain |

#### Returns

No return value

#### 5.69.2.3 AK\_create\_header\_name()

```
char* AK_create_header_name (  
    char * first,  
    char * second,  
    char * operator )
```

Creates new header name from passed operand names and operator.

Leon Palačić

#### Parameters

|                 |              |
|-----------------|--------------|
| <i>first</i>    | operand name |
| <i>second</i>   | operand name |
| <i>operator</i> |              |

#### Returns

new sequence of charachters

#### 5.69.2.4 AK\_determine\_header\_type()

```
int AK_determine_header_type (  
    int a,  
    int b )
```

Determines new header type.

Leon Palaić



**Parameters**

|          |              |
|----------|--------------|
| <i>a</i> | operand type |
| <i>b</i> | operand type |

**Returns**

header type

**5.69.2.5 AK\_get\_operator()**

```
char* AK_get_operator (
    char * exp )
```

Function that fetches arithmetics operator.

Leon Palaić

**Parameters**

|            |                         |
|------------|-------------------------|
| <i>exp</i> | input expression string |
|------------|-------------------------|

**Returns**

operator

**5.69.2.6 AK\_op\_projection\_test()**

```
void AK_op_projection_test ( )
```

Function for projection operator testing.

**Author**

Dino Laktašić

**Returns**

No return value

#### 5.69.2.7 AK\_perform\_operatrion()

```
char* AK_perform_operatrion (
    char * op,
    struct AK_operand * ab,
    struct AK_operand * bb,
    int type )
```

Performes arithmetics operation on operand data.

Leon Palaic

**Parameters**

|           |                |
|-----------|----------------|
| <i>ab</i> | first operand  |
| <i>bb</i> | second operand |
| <i>op</i> | operator       |

**Returns**

result of arithmetics

**5.69.2.8 AK\_projection()**

```
int AK_projection (
    char * srcTable,
    char * dstTable,
    struct list_node * att,
    struct list_node * expr )
```

Function makes a projection of some table.

**Author**

Matija Novak, rewrited and optimized by Dino Laktašić, now support cacheing

**Parameters**

|                 |   |
|-----------------|---|
| <i>att</i>      | - list of atributes on which we make projection |
| <i>dstTable</i> | table name for projection table                 |

**Returns**

EXIT\_SUCCESS if continues succesfully, when not EXIT\_ERROR

**5.69.2.9 AK\_temp\_create\_table()**

```
void AK_temp_create_table (
    char * table,
    AK_header * header,
    int type_segment )
```

Temporaly function to create table, and insert entry to the system\_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

|                     |  |
|---------------------|--|
| <i>table</i>        | table name                                 |
| <i>header</i>       | <a href="#">AK_header</a> of the new table |
| <i>type_segment</i> | type of the new segment                    |

**Returns**

No return value

**5.69.2.10 removeSubstring()**

```
void removeSubstring (
    char * s,
    const char * toremove )
```

Function that removes specified part of string.

Leon Palać

**Parameters**

|                 |               |
|-----------------|---------------|
| <i>s</i>        | input string  |
| <i>toremove</i> | remove string |

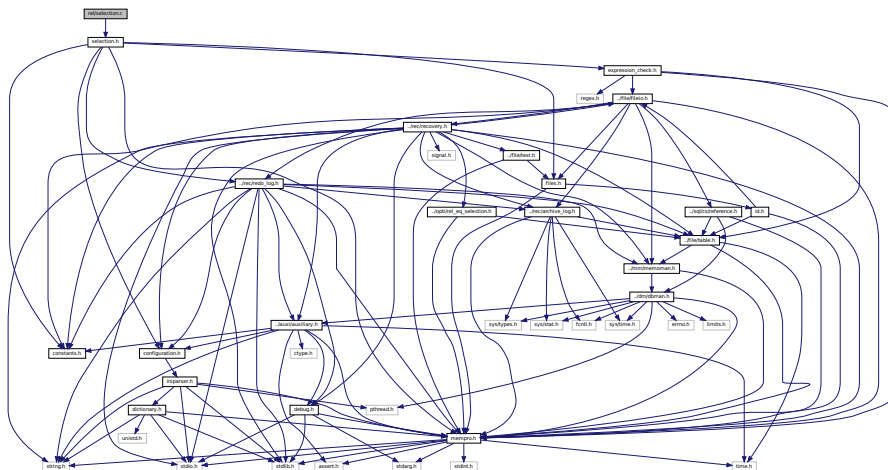
**Returns**

new sequence of characters

**5.70 rel/selection.c File Reference**

```
#include "selection.h"
```

Include dependency graph for selection.c:



## Functions

- int [AK\\_selection](#) (char \*srcTable, char \*dstTable, struct [list\\_node](#) \*expr)  
*Function which implements selection.*
- void [AK\\_op\\_selection\\_test](#) ()  
*Function for selection operator testing.*
- void [AK\\_op\\_selection\\_test\\_pattern](#) ()  
*Function for selection operator testing.*
- void [AK\\_op\\_selection\\_test\\_redolog](#) ()  
*Function for redolog testing.*

### 5.70.1 Detailed Description

Provides functions for relational selection operation

### 5.70.2 Function Documentation

#### 5.70.2.1 AK\_op\_selection\_test()

```
void AK_op_selection_test ( )
```

Function for selection operator testing.

#### Author

Matija Šestak, updated by Dino Laktašić, Nikola Miljancic

#### 5.70.2.2 AK\_op\_selection\_test\_pattern()

```
void AK_op_selection_test_pattern ( )
```

Function for selection operator testing.

##### Author

Krunoslav Bilić

#### 5.70.2.3 AK\_op\_selection\_test\_redolog()

```
void AK_op_selection_test_redolog ( )
```

Function for redolog testing.

##### Author

Krunoslav Bilić

#### 5.70.2.4 AK\_selection()

```
int AK_selection (
    char * srcTable,
    char * dstTable,
    struct list_node * expr )
```

Function which implements selection.

##### Author

Matija Šestak.

##### Parameters

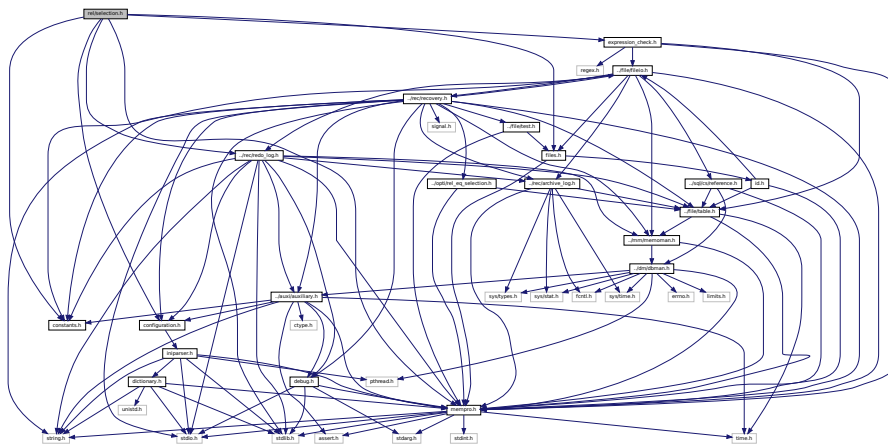
|                  |   |
|------------------|---|
| <i>*srcTable</i> | source table name                                   |
| <i>*dstTable</i> | destination table name                              |
| <i>*expr</i>     | list with posfix notation of the logical expression |

##### Returns

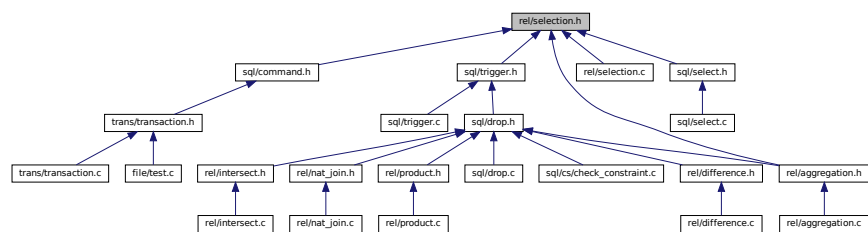
EXIT\_SUCCESS

## 5.71 rel/selection.h File Reference

```
#include "expression_check.h"
#include "../rec/redo_log.h"
#include "../auxi/constants.h"
#include "../auxi/configuration.h"
#include "../file/files.h"
#include "../auxi/mempro.h"
Include dependency graph for selection.h:
```



This graph shows which files directly or indirectly include this file:



### Functions

- int [AK\\_selection](#) (char \*srcTable, char \*dstTable, struct [list\\_node](#) \*expr)  
Function which implements selection.
- void [AK\\_op\\_selection\\_test](#) ()  
Function for selection operator testing.
- void [AK\\_op\\_selection\\_test\\_pattern](#) ()  
Function for selection operator testing.
- void [AK\\_op\\_selection\\_test\\_redolog](#) ()  
Function for redolog testing.

#### 5.71.1 Detailed Description

Header file that provides data structures for relational selection operation

## 5.71.2 Function Documentation

### 5.71.2.1 AK\_op\_selection\_test()

```
void AK_op_selection_test ( )
```

Function for selection operator testing.

**Author**

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic

### 5.71.2.2 AK\_op\_selection\_test\_pattern()

```
void AK_op_selection_test_pattern ( )
```

Function for selection operator testing.

**Author**

Krunoslav Bilić

### 5.71.2.3 AK\_op\_selection\_test\_redolog()

```
void AK_op_selection_test_redolog ( )
```

Function for redolog testing.

**Author**

Krunoslav Bilić

### 5.71.2.4 AK\_selection()

```
int AK_selection (
    char * srcTable,
    char * dstTable,
    struct list_node * expr )
```

Function which implements selection.

**Author**

Matija Šestak.



### Parameters

|                  |  |
|------------------|--|
| <i>*srcTable</i> | source table name                                    |
| <i>*dstTable</i> | destination table name                               |
| <i>*expr</i>     | list with postfix notation of the logical expression |

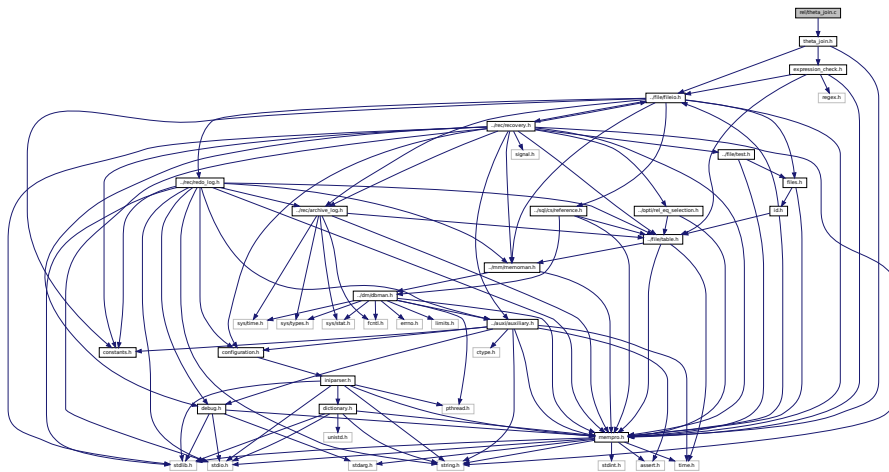
## Returns

EXIT\_SUCCESS

## 5.72 rel/theta\_join.c File Reference

```
#include "theta_join.h"
```

Include dependency graph for theta\_join.c:



## Functions

- int [AK\\_create\\_theta\\_join\\_header](#) (char \*srcTable1, char \*srcTable2, char \*new\_table)  
*Function for creating the header of the new table for theta join.*
- void [AK\\_check\\_constraints](#) ([AK\\_block](#) \*tbl1\_temp\_block, [AK\\_block](#) \*tbl2\_temp\_block, int tbl1\_num\_att, int tbl2\_num\_att, struct [list\\_node](#) \*constraints, char \*new\_table)  
*Function iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.*
- int [AK\\_theta\\_join](#) (char \*srcTable1, char \*srcTable2, char \*dstTable, struct [list\\_node](#) \*constraints)  
*Function for creating a theta join between two tables on specified conditions. Names of the attributes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.*
- void [AK\\_op\\_theta\\_join\\_test](#) ()  
*Function for testing the theta join.*

### 5.72.1 Detailed Description

Provides functions for relational theta join operation

## 5.72.2 Function Documentation

### 5.72.2.1 AK\_check\_constraints()

```
void AK_check_constraints (
    AK_block * tbl1_temp_block,
    AK_block * tbl2_temp_block,
    int tbl1_num_att,
    int tbl2_num_att,
    struct list_node * constraints,
    char * new_table )
```

Function iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.

#### Author

Tomislav Mikulček

#### Parameters

|                        |  |
|------------------------|--|
| <i>tbl1_temp_block</i> | block of the first table   |
| <i>tbl2_temp_block</i> | block of the second join table   |
| <i>tbl1_num_att</i>    | number of attributes in the first table  |
| <i>tbl2_num_att</i>    | number of attributes in the second table   |
| <i>constraints</i>     | list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation |
| <i>new_table</i>       | name of the theta_join table   |

#### Returns

No return value

### 5.72.2.2 AK\_create\_theta\_join\_header()

```
int AK_create_theta_join_header (
    char * srcTable1,
    char * srcTable2,
    char * new_table )
```

Function for creating the header of the new table for theta join.

#### Author

Tomislav Mikulček

## Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>srcTable1</i> | name of the first table       |
| <i>srcTable2</i> | name of the second table      |
| <i>new_table</i> | name of the destination table |

## Returns

EXIT\_SUCCESS if the header was successfully created and EXIT\_ERROR if the renamed headers are too long

## 5.72.2.3 AK\_op\_theta\_join\_test()

```
void AK_op_theta_join_test ( )
```

Function for testing the theta join.

## Author

Tomislav Mikulček

## Returns

No return value

## 5.72.2.4 AK\_theta\_join()

```
int AK_theta_join (
    char * srcTable1,
    char * srcTable2,
    char * dstTable,
    struct list_node * constraints )
```

Function for creating a theta join between two tables on specified conditions. Names of the attributes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.

## Author

Tomislav Mikulček, updated by Nikola Miljancic

## Parameters

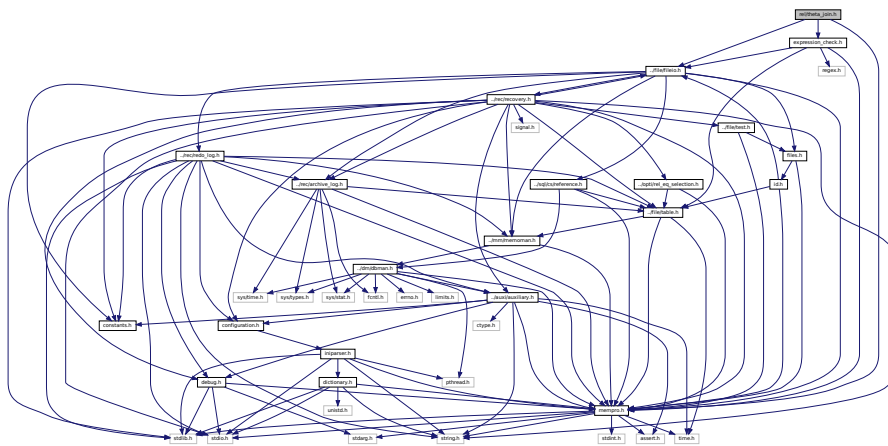
|                    |  |
|--------------------|--|
| <i>srcTable1</i>   | name of the first table to join  |
| <i>srcTable2</i>   | name of the second table to join   |
| <i>constraints</i> | list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation |
| <i>dstTable</i>    | name of the theta join table   |

## Returns

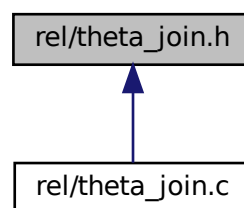
if successful returns EXIT\_SUCCESS and EXIT\_ERROR otherwise

## 5.73 rel/theta\_join.h File Reference

```
#include "expression_check.h"
#include "../file/fileio.h"
#include "../aux/mempro.h"
Include dependency graph for theta_join.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- int [AK\\_theta\\_join](#) (char \*srcTable1, char \*srcTable2, char \*dstTable, struct [list\\_node](#) \*constraints)

Function for creating a theta join between two tables on specified conditions. Names of the attributes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.

- int [AK\\_create\\_theta\\_join\\_header](#) (char \*srcTable1, char \*srcTable2, char \*new\_table)

Function for creating the header of the new table for theta join.

- void [AK\\_check\\_constraints](#) ([AK\\_block](#) \*tbl1\_temp\_block, [AK\\_block](#) \*tbl2\_temp\_block, int tbl1\_num\_att, int tbl2\_num\_att, struct [list\\_node](#) \*constraints, char \*new\_table)  
Function iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.
- void [AK\\_op\\_theta\\_join\\_test](#) ()  
Function for testing the theta join.

### 5.73.1 Detailed Description

Header file that provides data structures for theta-join

### 5.73.2 Function Documentation

#### 5.73.2.1 [AK\\_check\\_constraints\(\)](#)

```
void AK_check_constraints (
    AK\_block * tbl1_temp_block,
    AK\_block * tbl2_temp_block,
    int tbl1_num_att,
    int tbl2_num_att,
    struct list\_node * constraints,
    char * new_table )
```

Function iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.

#### Author

Tomislav Mikulček

#### Parameters

|                        |  |
|------------------------|--|
| <i>tbl1_temp_block</i> | block of the first table   |
| <i>tbl2_temp_block</i> | block of the second join table   |
| <i>tbl1_num_att</i>    | number of attributes in the first table  |
| <i>tbl2_num_att</i>    | number of attributes in the second table   |
| <i>constraints</i>     | list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation |
| <i>new_table</i>       | name of the theta_join table   |

#### Returns

No return value

#### 5.73.2.2 AK\_create\_theta\_join\_header()

```
int AK_create_theta_join_header (
    char * srcTable1,
    char * srcTable2,
    char * new_table )
```

Function for creating the header of the new table for theta join.

##### Author

Tomislav Mikulček

##### Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>srcTable1</i> | name of the first table       |
| <i>srcTable2</i> | name of the second table      |
| <i>new_table</i> | name of the destination table |

##### Returns

EXIT\_SUCCESS if the header was successfully created and EXIT\_ERROR if the renamed headers are too long

#### 5.73.2.3 AK\_op\_theta\_join\_test()

```
void AK_op_theta_join_test ( )
```

Function for testing the theta join.

##### Author

Tomislav Mikulček

##### Returns

No return value

#### 5.73.2.4 AK\_theta\_join()

```
int AK_theta_join (
    char * srcTable1,
    char * srcTable2,
    char * dstTable,
    struct list_node * constraints )
```

Function for creating a theta join between two tables on specified conditions. Names of the attributes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.

##### Author

Tomislav Mikulček, updated by Nikola Miljancic

## Parameters

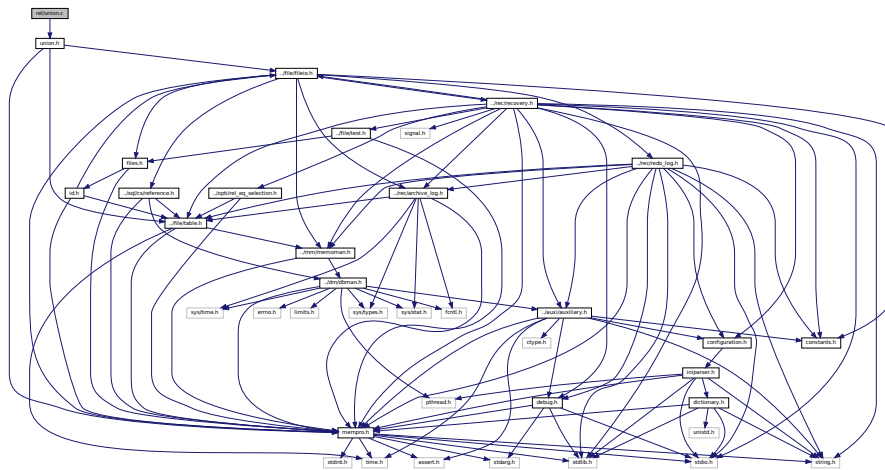
|                    |  |
|--------------------|--|
| <i>srcTable1</i>   | name of the first table to join  |
| <i>srcTable2</i>   | name of the second table to join   |
| <i>constraints</i> | list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation |
| <i>dstTable</i>    | name of the theta join table   |

## Returns

if successful returns EXIT\_SUCCESS and EXIT\_ERROR otherwise

## 5.74 rel/union.c File Reference

```
#include "union.h"
Include dependency graph for union.c:
```



## Functions

- int [AK\\_union](#) (char \*srcTable1, char \*srcTable2, char \*dstTable)  
Function to make union of the two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)
- void [AK\\_op\\_union\\_test](#) ()  
Function for union operator testing.

## 5.74.1 Detailed Description

Provides functions for relational union operation

## 5.74.2 Function Documentation

#### 5.74.2.1 AK\_op\_union\_test()

```
void AK_op_union_test ( )
```

Function for union operator testing.

##### Author

Dino Laktašić

##### Returns

No return value

#### 5.74.2.2 AK\_union()

```
int AK_union (
    char * srcTable1,
    char * srcTable2,
    char * dstTable )
```

Function to make union of the two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)

##### Author

Dino Laktašić

##### Parameters

|                  |                          |
|------------------|--------------------------|
| <i>srcTable1</i> | name of the first table  |
| <i>srcTable2</i> | name of the second table |
| <i>dstTable</i>  | name of the new table    |

##### Returns

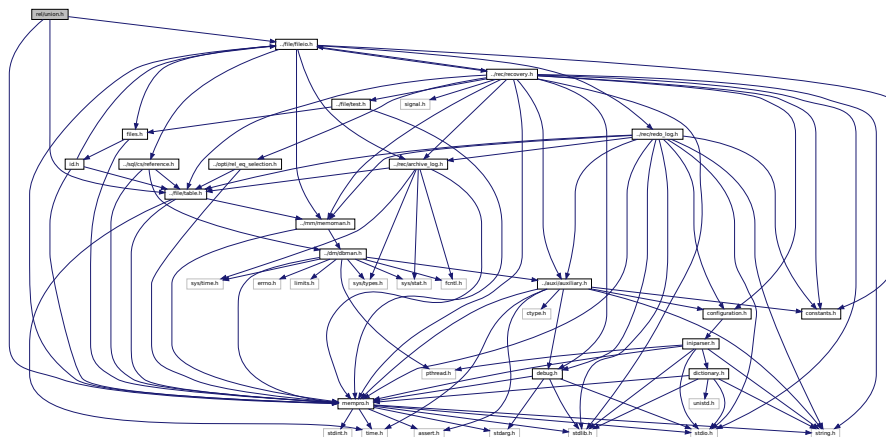
if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

## 5.75 rel/union.h File Reference

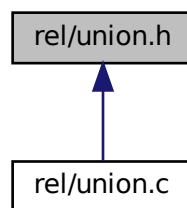
```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../aux/mempro.h"
```



Include dependency graph for union.h:



This graph shows which files directly or indirectly include this file:



## Functions

- int [AK\\_union](#) (char \*srcTable1, char \*srcTable2, char \*dstTable)  
Function to make union of the two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)
- void [AK\\_op\\_union\\_test](#) ()  
Function for union operator testing.

### 5.75.1 Detailed Description

Header file that provides data structures for relational union operation

### 5.75.2 Function Documentation

### 5.75.2.1 AK\_op\_union\_test()

```
void AK_op_union_test ( )
```

Function for union operator testing.

#### Author

Dino Laktašić

#### Returns

No return value

### 5.75.2.2 AK\_union()

```
int AK_union (
    char * srcTable1,
    char * srcTable2,
    char * dstTable )
```

Function to make union of the two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)

#### Author

Dino Laktašić

#### Parameters

|                  |                          |
|------------------|--------------------------|
| <i>srcTable1</i> | name of the first table  |
| <i>srcTable2</i> | name of the second table |
| <i>dstTable</i>  | name of the new table    |

#### Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

## 5.76 sql/cs/between.c File Reference

```
#include "between.h"
```



#### 5.76.2.1 Ak\_constraint\_between\_test()

```
void Ak_constraint_between_test ( )
```

Tests the functionality of implemented between constraint.

##### Author

Saša Vukšić, updated by Mislav Jurinić

##### Returns

No return value

#### 5.76.2.2 AK\_delete\_constraint\_between()

```
int AK_delete_constraint_between (
    char * tableName,
    char * constraintNamePar,
    char * attNamePar )
```

Function for deleting specific between constraint.

##### Author

Maja Vračan

##### Parameters

|                       |   |
|-----------------------|---|
| <i>tableName</i>      | name of table on which constraint refers          |
| <i>attName</i>        | name of attribute on which constraint is declared |
| <i>constraintName</i> | name of constraint                                |

##### Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

#### 5.76.2.3 AK\_find\_table\_address()

```
int AK_find_table_address (
    char * _systemTableName )
```

Returns system tables address by name.

##### Author

Mislav Jurinić

## Parameters

|                         |            |
|-------------------------|------------|
| <i>_systemTableName</i> | table name |
|-------------------------|------------|

## Returns

int

## 5.76.2.4 AK\_print\_constraints()

```
void AK_print_constraints (
    char * tableName )
```

Function for printing tables.

## Author

Maja Vračan

## Parameters

|                  |               |
|------------------|---------------|
| <i>tableName</i> | name of table |
|------------------|---------------|

## 5.76.2.5 AK\_read\_constraint\_between()

```
int AK_read_constraint_between (
    char * tableName,
    char * newValue,
    char * attNamePar )
```

Checks if the given value is between lower and upper bounds of the "between" constraint.

## Author

Saša Vukšić, updated by Mislav Jurinić

## Parameters

|                   |                         |
|-------------------|-------------------------|
| <i>tableName</i>  | table name              |
| <i>newValue</i>   | value we want to insert |
| <i>attNamePar</i> | attribute name          |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.76.2.6 AK\_set\_constraint\_between()**

```
void AK_set_constraint_between (
    char * tableName,
    char * constraintName,
    char * attName,
    char * startValue,
    char * endValue )
```

Function sets between constraints on particular attribute, string constraint should be written in lowercase. It searches for AK\_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.

**Author**

Saša Vukšić, updated by Mislav Jurinić

**Parameters**

|                       |                    |
|-----------------------|--------------------|
| <i>tableName</i>      | table name         |
| <i>constraintName</i> | name of constraint |
| <i>attName</i>        | name of attribute  |
| <i>startValue</i>     | initial constraint |
| <i>endValue</i>       | final constraint   |

**Returns**

No return value

**5.77 sql/cs/between.h File Reference**

```
#include "../mm/memoman.h"
#include "../file/id.h"
#include "../auxi/mempro.h"
```



## 5.77.2 Function Documentation

### 5.77.2.1 Ak\_constraint\_between\_test()

```
void Ak_constraint_between_test ( )
```

Tests the functionality of implemented between constraint.

#### Author

Saša Vukšić, updated by Mislav Jurinić

#### Returns

No return value

### 5.77.2.2 AK\_find\_table\_address()

```
int AK_find_table_address (
    char * _systemTableName )
```

Returns system tables address by name.

#### Author

Mislav Jurinić

#### Parameters

|                         |            |
|-------------------------|------------|
| <i>_systemTableName</i> | table name |
|-------------------------|------------|

#### Returns

int

### 5.77.2.3 AK\_read\_constraint\_between()

```
int AK_read_constraint_between (
    char * tableName,
    char * newValue,
    char * attNamePar )
```

Checks if the given value is between lower and upper bounds of the "between" constraint.



**Author**

Saša Vukšić, updated by Mislav Jurinić

**Parameters**

|                   |                         |
|-------------------|-------------------------|
| <i>tableName</i>  | table name              |
| <i>newValue</i>   | value we want to insert |
| <i>attNamePar</i> | attribute name          |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.77.2.4 AK\_set\_constraint\_between()**

```
void AK_set_constraint_between (
    char * tableName,
    char * constraintName,
    char * attName,
    char * startValue,
    char * endValue )
```

Function sets between constraints on particular attribute, string constraint should be written in lowercase. It searches for AK\_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.

**Author**

Saša Vukšić, updated by Mislav Jurinić

**Parameters**

|                       |                    |
|-----------------------|--------------------|
| <i>tableName</i>      | table name         |
| <i>constraintName</i> | name of constraint |
| <i>attName</i>        | name of attribute  |
| <i>startValue</i>     | initial constraint |
| <i>endValue</i>       | final constraint   |

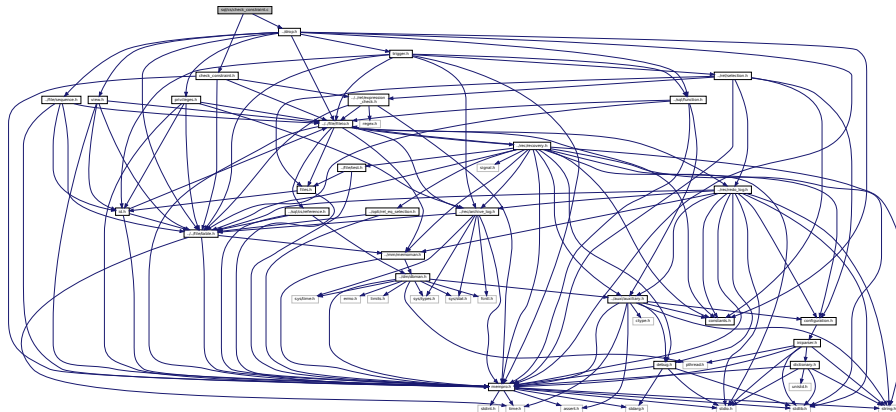
**Returns**

No return value

**5.78 sql/cs/check\_constraint.c File Reference**

```
#include "check_constraint.h"
```

```
#include "../drop.h"
Include dependency graph for check_constraint.c:
```



## Functions

- `int condition_passed (char *condition, int type, void *value, void *row_data)`  
*For given value, checks if it satisfies the "check" constraint.*
- `int AK_set_check_constraint (char *table_name, char *constraint_name, char *attribute_name, char *condition, int type, void *value)`  
*Adds a new "check" constraint into the system table.*
- `int AK_check_constraint (char *table, char *attribute, void *value)`  
*Verifies if the value we want to insert satisfies the "check" constraint.*
- `void AK_check_constraint_test ()`  
*Test function for "check" constraint.*

### 5.78.1 Detailed Description

Check constraint implementation file.

### 5.78.2 Function Documentation

#### 5.78.2.1 AK\_check\_constraint()

```
int AK_check_constraint (
    char * table,
    char * attribute,
    void * value )
```

Verifies if the value we want to insert satisfies the "check" constraint.

#### Author

Mislav Jurinić

## Parameters

|                  |                        |
|------------------|------------------------|
| <i>table</i>     | target table name      |
| <i>attribute</i> | target attribute name  |
| <i>value</i>     | data we want to insert |

## Returns

1 - success, 0 - failure

## 5.78.2.2 AK\_check\_constraint\_test()

```
void AK_check_constraint_test ( )
```

Test function for "check" constraint.

## Author

Mislav Jurinić

## Returns

void

## 5.78.2.3 AK\_set\_check\_constraint()

```
int AK_set_check_constraint (
    char * table_name,
    char * constraint_name,
    char * attribute_name,
    char * condition,
    int type,
    void * value )
```

Adds a new "check" constraint into the system table.

## Author

Mislav Jurinić

## Parameters

|                        |  |
|------------------------|--|
| <i>table_name</i>      | target table for "check" constraint evaluation                       |
| <i>constraint_name</i> | new "check" constraint name that will be visible in the system table |
| <i>attribute_name</i>  | target attribute for "check" constraint evaluation                   |
| <i>condition</i>       | logical operator ['<', '>', '!=', ...]                               |
| <i>type</i>            | data type [int, float, varchar, datetime, ...]                       |
| <i>value</i>           | condition to be set  |

**Returns**

1 - success, 0 - failure

**5.78.2.4 condition\_passed()**

```
int condition_passed (
    char * condition,
    int type,
    void * value,
    void * row_data )
```

For given value, checks if it satisfies the "check" constraint.

**Author**

Mislav Jurinić

**Parameters**

|                  |  |
|------------------|--|
| <i>condition</i> | logical operator ['<', '>', '!=', ...]         |
| <i>type</i>      | data type [int, float, varchar, datetime, ...] |
| <i>value</i>     | condition to be set                            |
| <i>row_data</i>  | data in table                                  |

**Returns**

1 - success, 0 - failure

**5.79 sql/cs/check\_constraint.h File Reference**

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../rel/expression_check.h"
#include "../auxi/mempro.h"
```

```
graph BT; A[sql/cs/check_constraint.c] --> B[sql/cs/check_constraint.h]
```

- int [condition\\_passed](#) (char \*condition, int type, void \*value, void \*row\_data)  
*For given value, checks if it satisfies the "check" constraint.*
- int [AK\\_set\\_check\\_constraint](#) (char \*table\_name, char \*constraint\_name, char \*attribute\_name, char \*condition, int type, void \*value)  
*Adds a new "check" constraint into the system table.*
- int [AK\\_check\\_constraint](#) (char \*table, char \*attribute, void \*value)  
*Verifies if the value we want to insert satisfies the "check" constraint.*
- void [AK\\_check\\_constraint\\_test](#) ()  
*Test function for "check" constraint.*

Header file that provides data structures for check constraint

## 5.79.2 Function Documentation

### 5.79.2.1 AK\_check\_constraint()

```
int AK_check_constraint (
    char * table,
    char * attribute,
    void * value )
```

Verifies if the value we want to insert satisfies the "check" constraint.

#### Author

Mislav Jurinić

#### Parameters

|                  |                        |
|------------------|------------------------|
| <i>table</i>     | target table name      |
| <i>attribute</i> | target attribute name  |
| <i>value</i>     | data we want to insert |

#### Returns

1 - success, 0 - failure

### 5.79.2.2 AK\_check\_constraint\_test()

```
void AK_check_constraint_test ( )
```

Test function for "check" constraint.

#### Author

Mislav Jurinić

#### Returns

void

### 5.79.2.3 AK\_set\_check\_constraint()

```
int AK_set_check_constraint (
    char * table_name,
    char * constraint_name,
    char * attribute_name,
    char * condition,
    int type,
    void * value )
```

Adds a new "check" constraint into the system table.

#### Author

Mislav Jurinić

#### Parameters

|                        |  |
|------------------------|--|
| <i>table_name</i>      | target table for "check" constraint evaluation                       |
| <i>constraint_name</i> | new "check" constraint name that will be visible in the system table |
| <i>attribute_name</i>  | target attribute for "check" constraint evaluation                   |
| <i>condition</i>       | logical operator ['<', '>', '!=', ...]                               |
| <i>type</i>            | data type [int, float, varchar, datetime, ...]                       |
| <i>value</i>           | condition to be set  |

#### Returns

1 - success, 0 - failure

### 5.79.2.4 condition\_passed()

```
int condition_passed (
    char * condition,
    int type,
    void * value,
    void * row_data )
```

For given value, checks if it satisfies the "check" constraint.

#### Author

Mislav Jurinić

#### Parameters

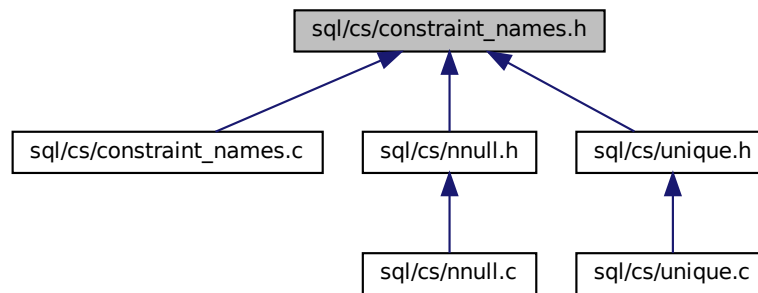
|                  |  |
|------------------|--|
| <i>condition</i> | logical operator ['<', '>', '!=', ...]         |
| <i>type</i>      | data type [int, float, varchar, datetime, ...] |
| <i>value</i>     | condition to be set                            |
| <i>row_data</i>  | data in table                                  |







This graph shows which files directly or indirectly include this file:



## Functions

- int [Ak\\_check\\_constraint\\_name](#) (char \*constraintName)  
Function checks if constraint name would be unique in database.
- void [AK\\_constraint\\_names\\_test](#) ()  
Function tests if constraint name would be unique in database.

### 5.81.1 Detailed Description

Header file that provides functions and data structures for checking if constraint name is unique in database

### 5.81.2 Function Documentation

#### 5.81.2.1 Ak\_check\_constraint\_name()

```
int Ak_check_constraint_name (
    char * constraintName )
```

Function checks if constraint name would be unique in database.

#### Author

Nenad Makar, updated by Mislav Jurinić

#### Parameters

|             |   |
|-------------|---|
| <i>char</i> | constraintName name which you want to give to constraint which you are trying to create |
|-------------|---|

## Returns

EXIT\_ERROR or EXIT\_SUCCESS

TODO add other constraint names from the catalog Also add them to "constants.h"

### 5.81.2.2 AK\_constraint\_names\_test()

```
void AK_constraint_names_test ( )
```

Function tests if constraint name would be unique in database.

**Author**

Nenad Makar

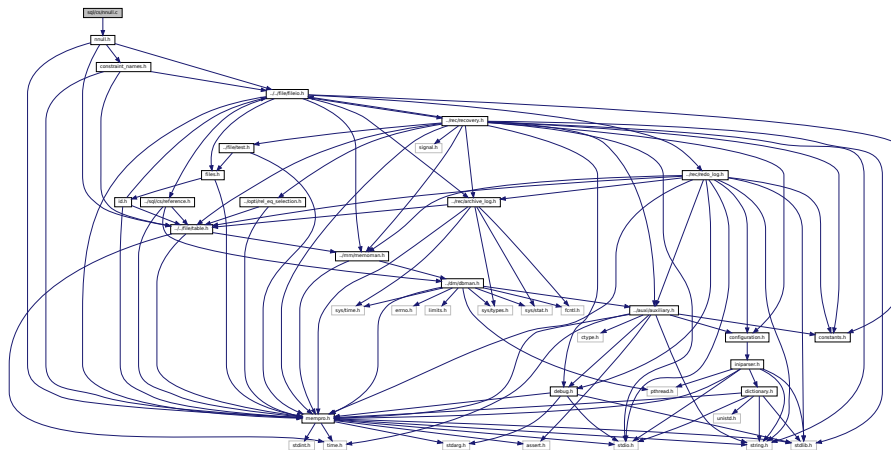
## Returns

No return value

## 5.82 sql/cs/nnull.c File Reference

```
#include "nnull.h"
```

Include dependency graph for nnull.c:



## Functions

- int [AK\\_set\\_constraint\\_not\\_null](#) (char \*tableName, char \*attName, char \*constraintName)  
*Function that sets NOT NULL constraint on attribute.*
- int [AK\\_read\\_constraint\\_not\\_null](#) (char \*tableName, char \*attName, char \*newValue)  
*Function checks if there's violation of NOT NULL constraint.*
- int [AK\\_delete\\_constraint\\_not\\_null](#) (char \*tableName, char attName[], char constraintName[])  
*Function for deleting specific not null constraint.*
- void [AK\\_null\\_test](#) ()  
*Function for testing testing NOT NULL constraint.*

## 5.82.1 Detailed Description

Provides functions for not null constraint

## 5.82.2 Function Documentation

### 5.82.2.1 AK\_delete\_constraint\_not\_null()

```
int AK_delete_constraint_not_null (
    char * tableName,
    char attName[],
    char constraintName[] )
```

Function for deleting specific not null constraint.

#### Author

Maja Vračan

#### Parameters

|                       |   |
|-----------------------|---|
| <i>tableName</i>      | name of table on which constraint refers          |
| <i>attName</i>        | name of attribute on which constraint is declared |
| <i>constraintName</i> | name of constraint                                |

#### Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

### 5.82.2.2 AK\_null\_test()

```
void AK_null_test ( )
```

Function for testing testing NOT NULL constraint.

#### Author

Saša Vukšić, updated by Nenad Makar

#### Returns

No return value

### 5.82.2.3 AK\_read\_constraint\_not\_null()

```
int AK_read_constraint_not_null (
    char * tableName,
    char * attName,
    char * newValue )
```

Function checks if there's violation of NOT NULL constraint.

#### Author

Saša Vukšić, updated by Nenad Makar

#### Parameters

|              |                           |
|--------------|---------------------------|
| <i>char*</i> | tableName name of table   |
| <i>char*</i> | attName name of attribute |
| <i>char*</i> | newValue new value        |

#### Returns

EXIT\_ERROR or EXIT\_SUCCESS

### 5.82.2.4 AK\_set\_constraint\_not\_null()

```
int AK_set_constraint_not_null (
    char * tableName,
    char * attName,
    char * constraintName )
```

Function that sets NOT NULL constraint on attribute.

#### Author

Saša Vukšić, updated by Nenad Makar

#### Parameters

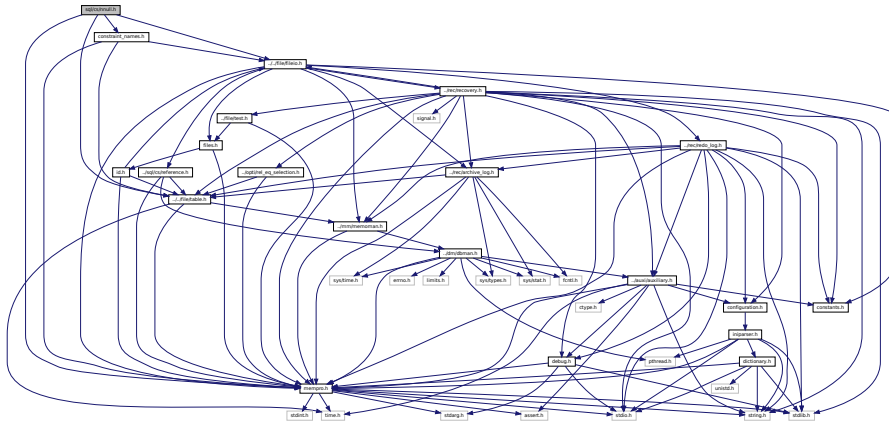
|              |                                   |
|--------------|-----------------------------------|
| <i>char*</i> | tableName name of table           |
| <i>char*</i> | attName name of attribute         |
| <i>char*</i> | constraintName name of constraint |

#### Returns

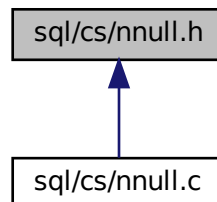
EXIT\_ERROR or EXIT\_SUCCESS

## 5.83 sql/cs/nnull.h File Reference

```
#include "../..//file/table.h"
#include "../..//file/fileio.h"
#include "../..//aux/mempro.h"
#include "constraint_names.h"
Include dependency graph for nnull.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- int [AK\\_set\\_constraint\\_not\\_null](#) (char \*tableName, char \*attName, char \*constraintName)  
*Function that sets NOT NULL constraint on attribute.*
- int [AK\\_read\\_constraint\\_not\\_null](#) (char \*tableName, char \*attName, char \*newValue)  
*Function checks if there's violation of NOT NULL constraint.*
- int [AK\\_delete\\_constraint\\_not\\_null](#) (char \*tableName, char attName[], char constraintName[])  
*Function for deleting specific not null constraint.*
- void [AK\\_null\\_test](#) ()  
*Function for testing testing NOT NULL constraint.*

### 5.83.1 Detailed Description

Header file that provides data structures for not null constraint

### 5.83.2 Function Documentation

#### 5.83.2.1 AK\_delete\_constraint\_not\_null()

```
int AK_delete_constraint_not_null (
    char * tableName,
    char attName[],
    char constraintName[] )
```

Function for deleting specific not null constraint.

#### Author

Maja Vračan

#### Parameters

|                       |   |
|-----------------------|---|
| <i>tableName</i>      | name of table on which constraint refers          |
| <i>attName</i>        | name of attribute on which constraint is declared |
| <i>constraintName</i> | name of constraint                                |

#### Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

#### 5.83.2.2 AK\_null\_test()

```
void AK_null_test ( )
```

Function for testing testing NOT NULL constraint.

#### Author

Saša Vukšić, updated by Nenad Makar

#### Returns

No return value

### 5.83.2.3 AK\_read\_constraint\_not\_null()

```
int AK_read_constraint_not_null (
    char * tableName,
    char * attName,
    char * newValue )
```

Function checks if there's violation of NOT NULL constraint.

#### Author

Saša Vukšić, updated by Nenad Makar

#### Parameters

|              |                                  |
|--------------|----------------------------------|
| <i>char*</i> | <i>tableName</i> name of table   |
| <i>char*</i> | <i>attName</i> name of attribute |
| <i>char*</i> | <i>newValue</i> new value        |

#### Returns

EXIT\_ERROR or EXIT\_SUCCESS

### 5.83.2.4 AK\_set\_constraint\_not\_null()

```
int AK_set_constraint_not_null (
    char * tableName,
    char * attName,
    char * constraintName )
```

Function that sets NOT NULL constraint on attribute.

#### Author

Saša Vukšić, updated by Nenad Makar

#### Parameters

|              |  |
|--------------|--|
| <i>char*</i> | <i>tableName</i> name of table           |
| <i>char*</i> | <i>attName</i> name of attribute         |
| <i>char*</i> | <i>constraintName</i> name of constraint |

#### Returns

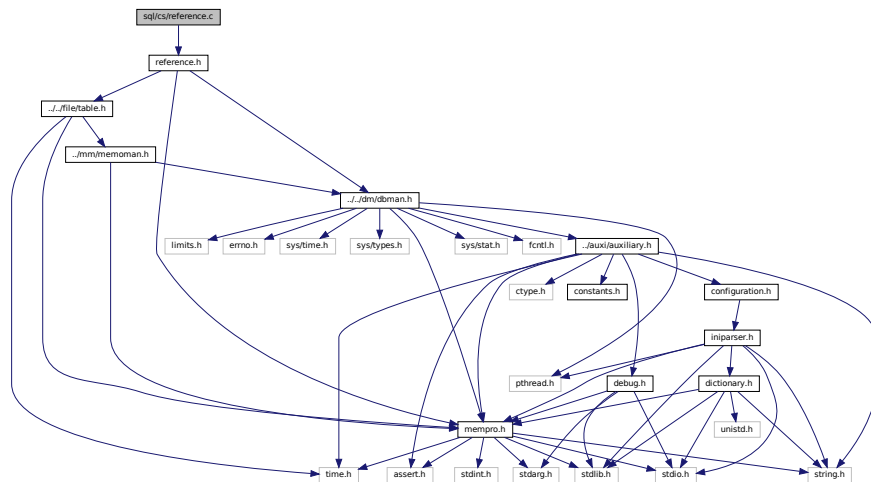
EXIT\_ERROR or EXIT\_SUCCESS



## 5.84 sql/cs/reference.c File Reference

```
#include "reference.h"
```

Include dependency graph for reference.c:



### Functions

- `int AK_add_reference (char *childTable, char *childAttNames[], char *parentTable, char *parentAttNames[], int attNum, char *constraintName, int type)`  
*Function adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name..*
- `AK_ref_item AK_get_reference (char *tableName, char *constraintName)`  
*Function reads a reference entry from system table.*
- `int AK_reference_check_attribute (char *tableName, char *attribute, char *value)`  
*Function checks referential integrity for one attribute.*
- `int AK_reference_check_if_update_needed (struct list_node *lista, int action)`  
*Funvction that quickly checks if there are any referential constraints that should be applied on a given list of changes.*
- `int AK_reference_check_restricion (struct list_node *lista, int action)`  
*Function checks for REF\_TYPE\_RESTRICT references applicable to the operation of updating or deleting a row in a table.*
- `int AK_reference_update (struct list_node *lista, int action)`  
*Function updates child table entries according to ongoing update of parent table entries.*
- `int AK_reference_check_entry (struct list_node *lista)`  
*Function checks new entry for referential integrity.*
- `void AK_reference_test ()`  
*Function for testing referential integrity.*

#### 5.84.1 Detailed Description

Provides functions for referential integrity

## 5.84.2 Function Documentation

### 5.84.2.1 AK\_add\_reference()

```
int AK_add_reference (
    char * childTable,
    char * childAttNames[],
    char * parentTable,
    char * parentAttNames[],
    int attNum,
    char * constraintName,
    int type )
```

Function adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name..

#### Author

Dejan Frankovic

#### Parameters

|               |   |
|---------------|---|
| <i>name</i>   | of the child table  |
| <i>array</i>  | of child table attribute names (foreign key attributes)                 |
| <i>name</i>   | of the parent table   |
| <i>array</i>  | of parent table attribute names (primary key attributes)                |
| <i>number</i> | of attributes in foreign key  |
| <i>name</i>   | of the constraint   |
| <i>type</i>   | of the constraint, constants defined in ' <a href="#">reference.h</a> ' |

#### Returns

EXIT\_SUCCESS

### 5.84.2.2 AK\_get\_reference()

```
AK_ref_item AK_get_reference (
    char * tableName,
    char * constraintName )
```

Function reads a reference entry from system table.

#### Author

Dejan Frankovic

## Parameters

|             |  |
|-------------|--|
| <i>name</i> | of the table with reference (with foreign key) |
| <i>name</i> | of the reference constraint                    |

## Returns

[AK\\_ref\\_item](#) object with all necessary information about the reference

## 5.84.2.3 AK\_reference\_check\_attribute()

```
int AK_reference_check_attribute (
    char * tableName,
    char * attribute,
    char * value )
```

Function checks referential integrity for one attribute.

## Author

Dejan Frankovic

## Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>child</i>     | table name                      |
| <i>attribute</i> | name (foreign key attribute)    |
| <i>value</i>     | of the attribute we're checking |

## Returns

EXIT\_ERROR if check failed, EXIT\_SUCCESS if referential integrity is ok

## 5.84.2.4 AK\_reference\_check\_entry()

```
int AK_reference_check_entry (
    struct list_node * lista )
```

Function checks new entry for referential integrity.

## Author

Dejan Franković

**Parameters**

|             |                            |
|-------------|----------------------------|
| <i>list</i> | of elements for insert row |
|-------------|----------------------------|

**Returns**

EXIT\_SUCCESS if referential integrity is ok, EXIT\_ERROR if it is compromised

**5.84.2.5 AK\_reference\_check\_if\_update\_needed()**

```
int AK_reference_check_if_update_needed (
    struct list_node * lista,
    int action )
```

Funvction that quickly checks if there are any referential constraints that should be applied on a given list of changes.

**Author**

Dejan Frankovic

**Parameters**

|             |                           |
|-------------|---------------------------|
| <i>list</i> | of elements for update    |
| <i>is</i>   | action UPDATE or DELETE ? |

**Returns**

EXIT\_SUCCESS if update is needed, EXIT\_ERROR if not

**5.84.2.6 AK\_reference\_check\_restricion()**

```
int AK_reference_check_restricion (
    struct list_node * lista,
    int action )
```

Function checks for REF\_TYPE\_RESTRICT references appliable to the operation of updating or deleting a row in a table.

**Author**

Dejan Franković

**Parameters**

|             |                          |
|-------------|--------------------------|
| <i>list</i> | of elements for update   |
| <i>is</i>   | action UPDATE or DELETE? |

**Returns**

EXIT\_SUCCESS if there is no restriction on this action, EXIT\_ERROR if there is

**5.84.2.7 AK\_reference\_test()**

```
void AK_reference_test ( )
```

Function for testing referential integrity.

**Author**

Dejan Franković

**Returns**

No return value

**5.84.2.8 AK\_reference\_update()**

```
int AK_reference_update (
    struct list_node * lista,
    int action )
```

Function updates child table entries according to ongoing update of parent table entries.

**Author**

Dejan Franković

**Parameters**

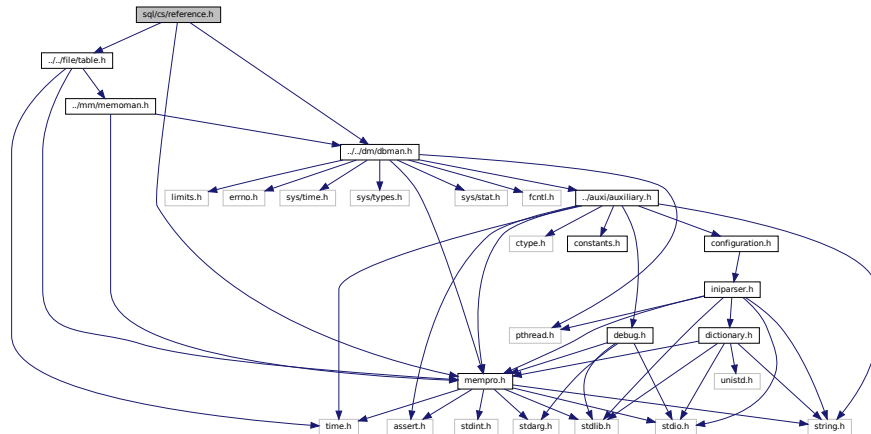
|             |                           |
|-------------|---------------------------|
| <i>list</i> | of elements for update    |
| <i>is</i>   | action UPDATE or DELETE ? |

**Returns**

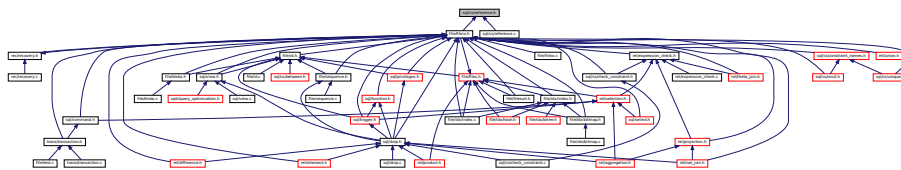
EXIT\_SUCCESS

## 5.85 sql/cs/reference.h File Reference

```
#include "../dm/dbman.h"
#include "../file/table.h"
#include "../auxi/mempro.h"
Include dependency graph for reference.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [AK\\_ref\\_item](#)

Structure that represents reference item. It contains of table, attributes, parent table and it's attributes, number of attributes, constraint and type of reference.

## Macros

- `#define` [REF\\_TYPE\\_NONE](#) -1  
Constant declaring none reference type.
- `#define` [REF\\_TYPE\\_SET\\_NULL](#) 1  
Constant declaring set null reference type.
- `#define` [REF\\_TYPE\\_NO\\_ACTION](#) 2  
Constant declaring no action reference type.
- `#define` [REF\\_TYPE\\_CASCADE](#) 3
- `#define` [REF\\_TYPE\\_RESTRICT](#) 4  
Constant declaring restrict reference type.
- `#define` [REF\\_TYPE\\_SET\\_DEFAULT](#) 5

- Constant declaring set default reference type.
  - `#define MAX_REFERENCE_ATTRIBUTES 10`  
Constant declaring maximum number of reference attributes.
  - `#define MAX_CHILD_CONSTRAINTS 20`  
Constant declaring maximum number of child constraints.

## Functions

- int `AK_add_reference` (char \*childTable, char \*childAttNames[], char \*parentTable, char \*parentAttNames[], int attNum, char \*constraintName, int type)  
 Function adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name..
- AK\_ref\_item `AK_get_reference` (char \*tableName, char \*constraintName)  
 Function reads a reference entry from system table.
- int `AK_reference_check_attribute` (char \*tableName, char \*attribute, char \*value)  
 Function checks referential integrity for one attribute.
- int `AK_reference_check_if_update_needed` (struct list\_node \*lista, int action)  
 Function that quickly checks if there are any referential constraints that should be applied on a given list of changes.
- int `AK_reference_check_restricion` (struct list\_node \*lista, int action)  
 Function checks for REF\_TYPE\_RESTRICT references applicable to the operation of updating or deleting a row in a table.
- int `AK_reference_update` (struct list\_node \*lista, int action)  
 Function updates child table entries according to ongoing update of parent table entries.
- int `AK_reference_check_entry` (struct list\_node \*lista)  
 Function checks new entry for referential integrity.
- void `AK_reference_test` ()  
 Function for testing referential integrity.
- void `Ak_Insert_New_Element` (int newtype, void \*data, char \*table, char \*attribute\_name, struct list\_node \*ElementBefore)  
 Function inserts new element after some element, to insert on first place give list as before element. It calls function `Ak_Insert_New_Element_For_Update`.
- int `Ak_insert_row` (struct list\_node \*row\_root)  
 Function inserts a one row into table. Firstly it is checked whether inserted row would violate reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_DIRTY.
- int `AK_selection` (char \*srcTable, char \*dstTable, struct list\_node \*expr)  
 Function which implements selection.
- void `Ak_Insert_New_Element_For_Update` (int newtype, void \*data, char \*table, char \*attribute\_name, struct list\_node \*ElementBefore, int newconstraint)  
 Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elements are set according to function arguments. Pointers are changed so that before element points to new element.
- int `Ak_delete_row` (struct list\_node \*row\_root)  
 Function deletes rows.
- int `Ak_update_row` (struct list\_node \*row\_root)  
 Function updates rows of some table.
- int `AK_initialize_new_segment` (char \*name, int type, AK\_header \*header)  
 Function initializes new segment and writes its start and finish address in system catalog table. For creating new table, index, temporary table, etc. call this function.

### 5.85.1 Detailed Description

d Provides data structures for referential integrity

### 5.85.2 Macro Definition Documentation

#### 5.85.2.1 REF\_TYPE\_NO\_ACTION

```
#define REF_TYPE_NO_ACTION 2
```

Constant declaring no action reference type.

Constant declaring cascade reference type.

### 5.85.3 Function Documentation

#### 5.85.3.1 AK\_add\_reference()

```
int AK_add_reference (
    char * childTable,
    char * childAttNames[],
    char * parentTable,
    char * parentAttNames[],
    int attNum,
    char * constraintName,
    int type )
```

Function adds a reference for a group of attributes over a given table to a group of attributes over another table with a given constraint name..

#### Author

Dejan Frankovic

#### Parameters

|               |   |
|---------------|---|
| <i>name</i>   | of the child table  |
| <i>array</i>  | of child table attribute names (foreign key attributes)                 |
| <i>name</i>   | of the parent table   |
| <i>array</i>  | of parent table attribute names (primary key attributes)                |
| <i>number</i> | of attributes in foreign key  |
| <i>name</i>   | of the constraint   |
| <i>type</i>   | of the constraint, constants defined in ' <a href="#">reference.h</a> ' |



**Returns**

EXIT\_SUCCESS

**5.85.3.2 Ak\_delete\_row()**

```
int Ak_delete_row (
    struct list_node * row_root )
```

Function deletes rows.

**Author**

Matija Novak, Dejan Frankovic (added referential integrity)

**Parameters**

|                 |   |
|-----------------|---|
| <i>row_root</i> | elements of one row EXIT_SUCCESS if success |
|-----------------|---|

**5.85.3.3 AK\_get\_reference()**

```
AK_ref_item AK_get_reference (
    char * tableName,
    char * constraintName )
```

Function reads a reference entry from system table.

**Author**

Dejan Frankovic

**Parameters**

|             |  |
|-------------|--|
| <i>name</i> | of the table with reference (with foreign key) |
| <i>name</i> | of the reference constraint                    |

**Returns**

[AK\\_ref\\_item](#) object with all necessary information about the reference

**5.85.3.4 AK\_initialize\_new\_segment()**

```
int AK_initialize_new_segment (
    char * name,
```

```
int type,
AK_header * header )
```

Function initializes new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.

#### Author

Tomislav Fotak, updated by Matija Šestak (function now uses caching)

#### Parameters

|               |   |
|---------------|---|
| <i>name</i>   | segment name  |
| <i>type</i>   | segment type  |
| <i>header</i> | pointer to header that should be written to the new extent (all blocks) |

#### Returns

start address of new segment

#### 5.85.3.5 Ak\_Insert\_New\_Element()

```
void Ak_Insert_New_Element (
    int newtype,
    void * data,
    char * table,
    char * attribute_name,
    struct list_node * ElementBefore )
```

Function inserts new element after some element, to insert on first place give list as before element. It calls function Ak\_Insert\_New\_Element\_For\_Update.

#### Author

Matija Novak, changed by Dino Laktašić

#### Parameters

|                       |   |
|-----------------------|---|
| <i>newtype</i>        | type of the data                              |
| <i>data</i>           | the data                                      |
| <i>table</i>          | table name                                    |
| <i>attribute_name</i> | attribute name                                |
| <i>element</i>        | element after we which insert the new element |
| <i>constraint</i>     | is NEW_VALUE                                  |

#### Returns

No return value

### 5.85.3.6 Ak\_Insert\_New\_Element\_For\_Update()

```
void Ak_Insert_New_Element_For_Update (
    int newtype,
    void * data,
    char * table,
    char * attribute_name,
    struct list_node * ElementBefore,
    int newconstraint )
```

Function inserts new element after some element, to insert on first place give list as before element. New element is allocated. Type, data, attribute name and constraint of new elements are set according to function arguments. Pointers are changed so that before element points to new element.

#### Author

Matija Novak

#### Parameters

|                       |   |
|-----------------------|---|
| <i>newtype</i>        | type of the data  |
| <i>data</i>           | the data  |
| <i>table</i>          | table name  |
| <i>attribute_name</i> | attribute name  |
| <i>element</i>        | element after we which insert the new element   |
| <i>constraint</i>     | NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for |

#### Returns

No return value

### 5.85.3.7 Ak\_insert\_row()

```
int Ak_insert_row (
    struct list_node * row_root )
```

Function inserts a one row into table. Firstly it is checked whether inserted row would violate reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_DIRTY.

#### Author

Matija Novak, updated by Matija Šestak (function now uses caching), updated by Dejan Frankovic (added reference check), updated by Dino Laktašić (removed variable AK\_free, variable table initialized using memset)

**Parameters**

|                 |  |
|-----------------|--|
| <i>row_root</i> | list of elements which contain data of one row |
|-----------------|--|

**Returns**

EXIT\_SUCCESS if success else EXIT\_ERROR

**5.85.3.8 AK\_reference\_check\_attribute()**

```
int AK_reference_check_attribute (
    char * tableName,
    char * attribute,
    char * value )
```

Function checks referential integrity for one attribute.

**Author**

Dejan Frankovic

**Parameters**

|                  |                                 |
|------------------|---------------------------------|
| <i>child</i>     | table name                      |
| <i>attribute</i> | name (foreign key attribute)    |
| <i>value</i>     | of the attribute we're checking |

**Returns**

EXIT\_ERROR if check failed, EXIT\_SUCCESS if referential integrity is ok

**5.85.3.9 AK\_reference\_check\_entry()**

```
int AK_reference_check_entry (
    struct list_node * lista )
```

Function checks new entry for referential integrity.

**Author**

Dejan Franković

### Parameters

|             |                            |
|-------------|----------------------------|
| <i>list</i> | of elements for insert row |
|-------------|----------------------------|

### Returns

EXIT\_SUCCESS if referential integrity is ok, EXIT\_ERROR if it is compromised

#### 5.85.3.10 AK\_reference\_check\_if\_update\_needed()

```
int AK_reference_check_if_update_needed (
    struct list_node * lista,
    int action )
```

Funvction that quickly checks if there are any referential constraints that should be applied on a given list of changes.

### Author

Dejan Frankovic

### Parameters

|             |                           |
|-------------|---------------------------|
| <i>list</i> | of elements for update    |
| <i>is</i>   | action UPDATE or DELETE ? |

### Returns

EXIT\_SUCCESS if update is needed, EXIT\_ERROR if not

#### 5.85.3.11 AK\_reference\_check\_restricion()

```
int AK_reference_check_restricion (
    struct list_node * lista,
    int action )
```

Function checks for REF\_TYPE\_RESTRICT references appliable to the operation of updating or deleting a row in a table.

### Author

Dejan Franković

**Parameters**

|             |                          |
|-------------|--------------------------|
| <i>list</i> | of elements for update   |
| <i>is</i>   | action UPDATE or DELETE? |

**Returns**

EXIT\_SUCCESS if there is no restriction on this action, EXIT\_ERROR if there is

**5.85.3.12 AK\_reference\_test()**

```
void AK_reference_test ( )
```

Function for testing referential integrity.

**Author**

Dejan Franković

**Returns**

No return value

**5.85.3.13 AK\_reference\_update()**

```
int AK_reference_update (
    struct list_node * lista,
    int action )
```

Function updates child table entries according to ongoing update of parent table entries.

**Author**

Dejan Franković

**Parameters**

|             |                           |
|-------------|---------------------------|
| <i>list</i> | of elements for update    |
| <i>is</i>   | action UPDATE or DELETE ? |

**Returns**

EXIT\_SUCCESS

#### 5.85.3.14 AK\_selection()

```
int AK_selection (
    char * srcTable,
    char * dstTable,
    struct list_node * expr )
```

Function which implements selection.

##### Author

Matija Šestak.

##### Parameters

|                  |   |
|------------------|---|
| <i>*srcTable</i> | source table name                                   |
| <i>*dstTable</i> | destination table name                              |
| <i>*expr</i>     | list with posfix notation of the logical expression |

##### Returns

EXIT\_SUCCESS

#### 5.85.3.15 Ak\_update\_row()

```
int Ak_update_row (
    struct list_node * row_root )
```

Function updates rows of some table.

##### Author

Matija Novak, Dejan Frankovic (added referential integrity)

##### Parameters

|                 |                     |
|-----------------|---------------------|
| <i>row_root</i> | elements of one row |
|-----------------|---------------------|

##### Returns

EXIT\_SUCCESS if success

## 5.86 sql/cs/unique.c File Reference

```
#include "unique.h"
```





## Parameters

|                       |   |
|-----------------------|---|
| <i>tableName</i>      | name of table on which constraint refers          |
| <i>attName</i>        | name of attribute on which constraint is declared |
| <i>constraintName</i> | name of constraint                                |

## Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

## 5.86.2.2 AK\_read\_constraint\_unique()

```
int AK_read_constraint_unique (
    char * tableName,
    char attName[],
    char newValue[] )
```

Function checks if insertion of some value(s) would violate UNIQUE constraint.

## Author

Domagoj Tuličić, updated by Nenad Makar

## Parameters

|              |  |
|--------------|--|
| <i>char*</i> | tableName name of table  |
| <i>char</i>  | attName[] name(s) of attribute(s), if you want to check combination of values of more attributes separate names of attributes with constant SEPARATOR (see test)   |
| <i>char</i>  | newValue[] new value(s), if you want to check combination of values of more attributes separate their values with constant SEPARATOR (see test), if some value(s) which you want to check isn't stored as char (string) convert it to char (string) using <a href="#">AK_tuple_to_string(struct list_node *tuple)</a> or with <code>sprintf</code> in a similar way it's used in that function (if value isn't part of a *tuple), to concatenate more values in newValue[] use <code>strcat(destination, source)</code> and put constant SEPARATOR between them (see test) if newValue[] should contain NULL sign pass it as " " (space) |

## Returns

EXIT\_ERROR or EXIT\_SUCCESS

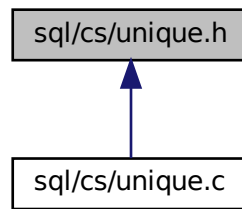
## 5.86.2.3 Ak\_set\_constraint\_unique()

```
int Ak_set_constraint_unique (
    char * tableName,
    char attName[],
    char constraintName[] )
```

Function sets unique constraint on attribute(s)



This graph shows which files directly or indirectly include this file:



## Functions

- int [Ak\\_set\\_constraint\\_unique](#) (char \*tableName, char attName[], char constraintName[])  
*Function sets unique constraint on attribute(s)*
- int [AK\\_read\\_constraint\\_unique](#) (char \*tableName, char attName[], char newValue[])  
*Function checks if insertion of some value(s) would violate UNIQUE constraint.*
- int [AK\\_delete\\_constraint\\_unique](#) (char \*tableName, char attName[], char constraintName[])  
*Function for deleting specific unique constraint.*
- void [AK\\_unique\\_test](#) ()  
*Function for testing UNIQUE constraint.*

### 5.87.1 Detailed Description

Header file that provides functions and data structures for unique constraint

### 5.87.2 Function Documentation

#### 5.87.2.1 AK\_delete\_constraint\_unique()

```
int AK_delete_constraint_unique (  
    char * tableName,  
    char attName[],  
    char constraintName[] )
```

Function for deleting specific unique constraint.

#### Author

Maja Vračan

**Parameters**

|                       |   |
|-----------------------|---|
| <i>tableName</i>      | name of table on which constraint refers          |
| <i>attName</i>        | name of attribute on which constraint is declared |
| <i>constraintName</i> | name of constraint                                |

**Returns**

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

**5.87.2.2 AK\_read\_constraint\_unique()**

```
int AK_read_constraint_unique (
    char * tableName,
    char attName[],
    char newValue[] )
```

Function checks if insertion of some value(s) would violate UNIQUE constraint.

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Parameters**

|              |  |
|--------------|--|
| <i>char*</i> | tableName name of table  |
| <i>char</i>  | attName[] name(s) of attribute(s), if you want to check combination of values of more attributes separate names of attributes with constant SEPARATOR (see test)   |
| <i>char</i>  | newValue[] new value(s), if you want to check combination of values of more attributes separate their values with constant SEPARATOR (see test), if some value(s) which you want to check isn't stored as char (string) convert it to char (string) using <a href="#">AK_tuple_to_string(struct list_node *tuple)</a> or with sprintf in a similar way it's used in that function (if value isn't part of a *tuple), to concatenate more values in newValue[] use strcat(destination, source) and put constant SEPARATOR between them (see test) if newValue[] should contain NULL sign pass it as " " (space) |

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**5.87.2.3 Ak\_set\_constraint\_unique()**

```
int Ak_set_constraint_unique (
    char * tableName,
    char attName[],
    char constraintName[] )
```

Function sets unique constraint on attribute(s)

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Parameters**

|              |  |
|--------------|--|
| <i>char*</i> | tableName name of table  |
| <i>char</i>  | attName[] name(s) of attribute(s), if you want to set UNIQUE constraint on combination of attributes seperate their names with constant SEPARATOR (see test) |
| <i>char</i>  | constraintName[] name of constraint  |

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**5.87.2.4 AK\_unique\_test()**

```
void AK_unique_test ( )
```

Function for testing UNIQUE constraint.

**Author**

Domagoj Tuličić, updated by Nenad Makar

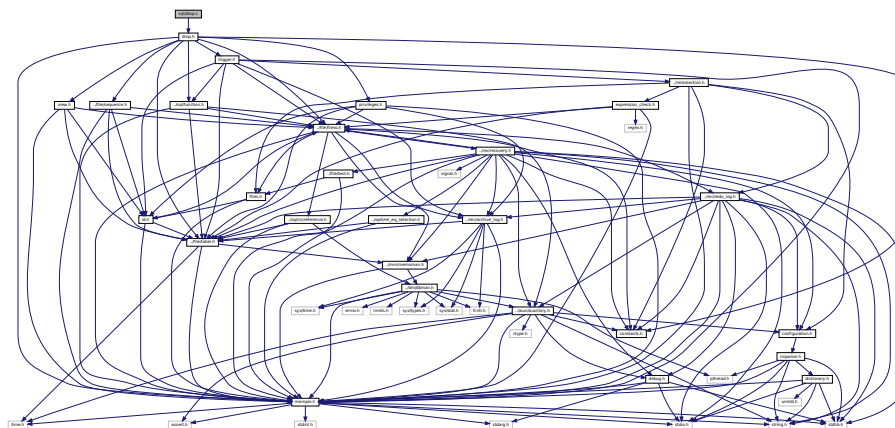
**Returns**

No return value

**5.88 sql/drop.c File Reference**

```
#include "drop.h"
```

Include dependency graph for drop.c:



## Functions

- int [AK\\_drop](#) (int type, [AK\\_drop\\_arguments](#) \*drop\_arguments)  
*Function for DROP table, index, view, sequence, trigger, function, user, group and constraint.*
- void [AK\\_drop\\_help\\_function](#) (char \*tblName, char \*sys\_table)  
*Help function for drop command. Delete memory blocks and addresses of table and removes table or index from system table.*
- int [AK\\_if\\_exist](#) (char \*tblName, char \*sys\_table)  
*Help function for check if element(view, function, sequence, user ...) exist in system catalog table.*
- void [AK\\_drop\\_test](#) ()  
*Function for testing all DROP functions.*

## Variables

- char \* **system\_catalog** [[NUM\\_SYS\\_TABLES](#)]

### 5.88.1 Detailed Description

#### Author

Unknown, Jurica Hlevnjak - drop table bugs fixed, reorganized code structure, system catalog tables drop disabled, drop index added, drop view added, drop sequence added, drop trigger added, drop\_function added, drop user added, drop group added, AK\_drop\_test updated

Provides DROP functions

### 5.88.2 Function Documentation

#### 5.88.2.1 AK\_drop()

```
int AK_drop (
    int type,
    AK\_drop\_arguments * drop_arguments )
```

Function for DROP table, index, view, sequence, trigger, function, user, group and constraint.

#### Author

Unknown, Jurica Hlevnjak, updated by Tomislav Ilisevic, Maja Vračan

#### Parameters

|                                |                           |
|--------------------------------|---------------------------|
| <i>type</i>                    | drop type                 |
| <a href="#">drop_arguments</a> | arguments of DROP command |

### 5.88.2.2 AK\_drop\_help\_function()

```
void AK_drop_help_function (
    char * tblName,
    char * sys_table )
```

Help function for drop command. Delete memory blocks and addresses of table and removes table or index from system table.

#### Author

unknown, Jurica Hlevnjak - fix bugs and reorganize code in this function

#### Parameters

|                  |                              |
|------------------|------------------------------|
| <i>tblName</i>   | name of table or index       |
| <i>sys_table</i> | name of system catalog table |

### 5.88.2.3 AK\_drop\_test()

```
void AK_drop_test ( )
```

Function for testing all DROP functions.

#### Author

unknown, Jurica Hlevnjak - added all tests except drop table test, updated by Tomislav Ilisevic, Maja Vračan

### 5.88.2.4 AK\_if\_exist()

```
int AK_if_exist (
    char * tblName,
    char * sys_table )
```

Help function for check if element(view, function, sequence, user ...) exist in system catalog table.

#### Author

Jurica Hlevnjak, updated by Tomislav Ilisevic

**Parameters**

|                  |   |
|------------------|---|
| <i>tblName</i>   | name of table, index view, function, trigger, sequence, user, group or constraint |
| <i>sys_table</i> | name of system catalog table  |

**Returns**

if element exist in system catalog returns 1, if not returns 0

**5.88.3 Variable Documentation****5.88.3.1 system\_catalog**

```
char* system_catalog[NUM\_SYS\_TABLES]
```

**Initial value:**

```
= {
    "AK_relation",
    "AK_attribute",
    "AK_index",
    "AK_view",
    "AK_sequence",
    "AK_function",
    "AK_function_arguments",
    "AK_trigger",
    "AK_trigger_conditions",
    "AK_db",
    "AK_db_obj",
    "AK_user",
    "AK_group",
    "AK_user_group",
    "AK_user_right",
    "AK_group_right",
    "AK_constraints_between",
    "AK_constraints_not_null",
    AK\_CONSTRAINTS\_CHECK\_CONSTRAINT,
    "AK_constraints_unique",
    "AK_reference"
}
```

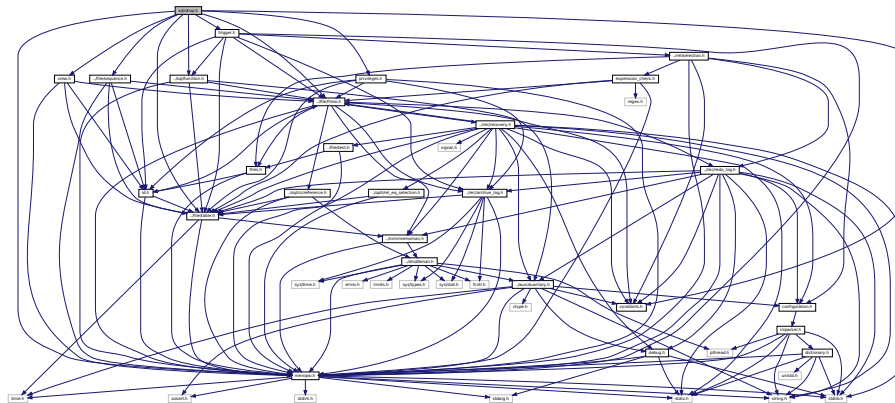
**5.89 sql/drop.h File Reference**

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../file/sequence.h"
#include "view.h"
#include "trigger.h"
#include "function.h"
#include "privileges.h"
#include "../aux/mempro.h"
```

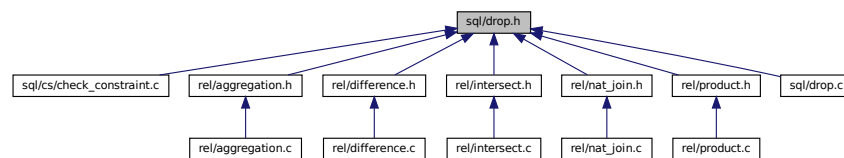


```
#include "../auxi/constants.h"
```

Include dependency graph for drop.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [drop\\_arguments](#)

## Typedefs

- typedef struct [drop\\_arguments](#) **AK\_drop\_arguments**

## Functions

- int [AK\\_drop](#) (int type, [AK\\_drop\\_arguments](#) \*drop\_arguments)  
*Function for DROP table, index, view, sequence, trigger, function, user, group and constraint.*
- void [AK\\_drop\\_test](#) ()  
*Function for testing all DROP functions.*
- int [AK\\_if\\_exist](#) (char \*tblName, char \*sys\_table)  
*Help function for check if element(view, function, sequence, user ...) exist in system catalog table.*

### 5.89.1 Function Documentation

### 5.89.1.1 AK\_drop()

```
int AK_drop (
    int type,
    AK_drop_arguments * drop_arguments )
```

Function for DROP table, index, view, sequence, trigger, function, user, group and constraint.

#### Author

Unknown, Jurica Hlevnjak, updated by Tomislav Ilisevic, Maja Vračan

#### Parameters

|                       |                           |
|-----------------------|---------------------------|
| <i>type</i>           | drop type                 |
| <i>drop_arguments</i> | arguments of DROP command |

### 5.89.1.2 AK\_drop\_test()

```
void AK_drop_test ( )
```

Function for testing all DROP functions.

#### Author

unknown, Jurica Hlevnjak - added all tests except drop table test, updated by Tomislav Ilisevic, Maja Vračan

### 5.89.1.3 AK\_if\_exist()

```
int AK_if_exist (
    char * tblName,
    char * sys_table )
```

Help function for check if element(view, function, sequence, user ...) exist in system catalog table.

#### Author

Jurica Hlevnjak, updated by Tomislav Ilisevic

#### Parameters

|                  |   |
|------------------|---|
| <i>tblName</i>   | name of table, index view, function, trigger, sequence, user, group or constraint |
| <i>sys_table</i> | name of system catalog table  |

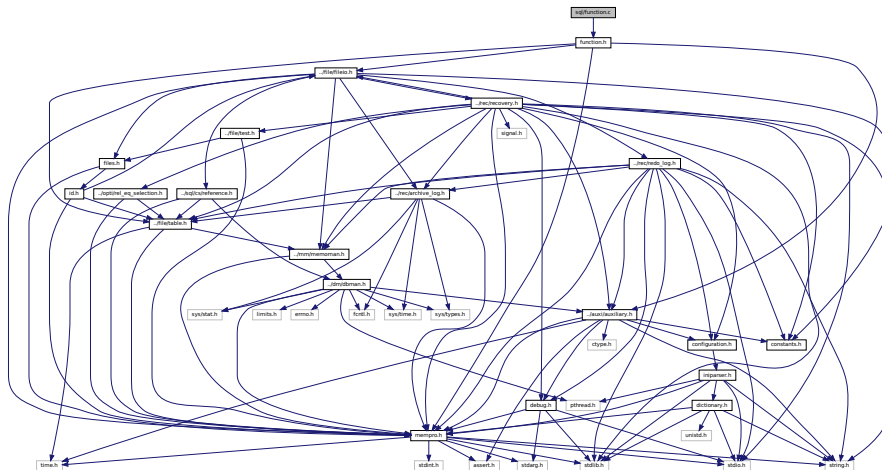
## Returns

if element exist in system catalog returns 1, if not returns 0

## 5.90 sql/function.c File Reference

```
#include "function.h"
```

Include dependency graph for function.c:



## Functions

- `int AK_get_function_obj_id (char *function, struct list_node *arguments_list)`  
Function that gets obj\_id of a function by name and arguments list (transferred from [trigger.c/drop.c](#)).
- `int AK_check_function_arguments (int function_id, struct list_node *arguments_list)`  
Function that checks whether arguments belong to function.
- `int AK_check_function_arguments_type (int function_id, struct list_node *args)`  
Function that checks whether arguments belong to function but only checks argument type (not name). Used for drop function.
- `int AK_function_add (char *name, int return_type, struct list_node *arguments_list)`  
Function that adds a function to system table.
- `int AK_function_arguments_add (int function_id, int arg_number, int arg_type, char *argname)`  
Function that adds a function argument to system table.
- `int AK_function_remove_by_obj_id (int obj_id)`  
Function removes a function by its obj\_id.
- `int AK_function_arguments_remove_by_obj_id (int *obj_id)`  
Function removes function arguments by function id.
- `int AK_function_remove_by_name (char *name, struct list_node *arguments_list)`  
Function that removes a function from system table by name and arguments.
- `int AK_function_rename (char *name, struct list_node *arguments_list, char *new_name)`  
Function that changes the function name.
- `int AK_function_change_return_type (char *name, struct list_node *arguments_list, int new_return_type)`  
Function that changes the return type.
- `void AK_function_test ()`  
Function for functions testing.

### 5.90.1 Detailed Description

Provides functions for functions

### 5.90.2 Function Documentation

#### 5.90.2.1 AK\_check\_function\_arguments()

```
int AK_check_function_arguments (
    int function_id,
    struct list_node * arguments_list )
```

Function that checks whether arguments belong to function.

#### Author

Boris Kišić

#### Parameters

|                        |                    |
|------------------------|--------------------|
| <i>*function_id</i>    | id of the function |
| <i>*arguments_list</i> | list of arguments  |

#### Returns

EXIT\_SUCCESS of the function or EXIT\_ERROR

#### 5.90.2.2 AK\_check\_function\_arguments\_type()

```
int AK_check_function_arguments_type (
    int function_id,
    struct list_node * args )
```

Function that checks whether arguments belong to function but only checks argument type (not name). Used for drop function.

#### Author

Jurica Hlevnjak

#### Parameters

|                    |                    |
|--------------------|--------------------|
| <i>function_id</i> | id of the function |
| <i>args</i>        | function arguments |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.90.2.3 AK\_function\_add()**

```
int AK_function_add (
    char * name,
    int return_type,
    struct list_node * arguments_list )
```

Function that adds a function to system table.

**Author**

Boris Kišić, updated by Tomislav Ilisevic

**Parameters**

|                        |   |
|------------------------|---|
| <i>*name</i>           | name of the function  |
| <i>*return_type</i>    | data type returned from a function - values from 0 to 13 - defined in <a href="#">constants.h</a> |
| <i>*arguments_list</i> | list of function arguments  |

**Returns**

function id or EXIT\_ERROR

**5.90.2.4 AK\_function\_arguments\_add()**

```
int AK_function_arguments_add (
    int function_id,
    int arg_number,
    int arg_type,
    char * argname )
```

Function that adds a function argument to system table.

**Author**

Boris Kišić

**Parameters**

|                     |  |
|---------------------|--|
| <i>*function_id</i> | id of the function to which the argument belongs |
| <i>*arg_number</i>  | number of the argument                           |
| <i>*arg_type</i>    | data type of the argument                        |
| <i>*argname</i>     | name of the argument                             |

**Returns**

function argument id or EXIT\_ERROR

**5.90.2.5 AK\_function\_arguments\_remove\_by\_obj\_id()**

```
int AK_function_arguments_remove_by_obj_id (
    int * obj_id )
```

Function removes function arguments by function id.

**Author**

Boris Kišić

**Parameters**

|               |                        |
|---------------|------------------------|
| <i>obj_id</i> | obj_id of the function |
|---------------|------------------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.90.2.6 AK\_function\_change\_return\_type()**

```
int AK_function_change_return_type (
    char * name,
    struct list_node * arguments_list,
    int new_return_type )
```

Function that changes the return type.

**Author**

Boris Kišić

**Parameters**

|                         |                                     |
|-------------------------|-------------------------------------|
| <i>*name</i>            | name of the function to be modified |
| <i>*arguments_list</i>  | list of function arguments          |
| <i>*new_return_type</i> | new return type                     |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.90.2.7 AK\_function\_remove\_by\_name()**

```
int AK_function_remove_by_name (
    char * name,
    struct list_node * arguments_list )
```

Function that removes a function from system table by name and arguments.

**Author**

Boris Kišić

**Parameters**

|                        |                      |
|------------------------|----------------------|
| <i>*name</i>           | name of the function |
| <i>*arguments_list</i> | list of arguments    |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.90.2.8 AK\_function\_remove\_by\_obj\_id()**

```
int AK_function_remove_by_obj_id (
    int obj_id )
```

Function removes a function by its obj\_id.

**Author**

Boris Kišić

**Parameters**

|               |                        |
|---------------|------------------------|
| <i>obj_id</i> | obj_id of the function |
|---------------|------------------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

#### 5.90.2.9 AK\_function\_rename()

```
int AK_function_rename (
    char * name,
    struct list_node * arguments_list,
    char * new_name )
```

Function that changes the function name.

##### Author

Boris Kišić

##### Parameters

|                        |                                     |
|------------------------|-------------------------------------|
| <i>*name</i>           | name of the function to be modified |
| <i>*arguments_list</i> | list of arguments to be modified    |
| <i>*new_name</i>       | new name of the function            |

##### Returns

EXIT\_SUCCESS or EXIT\_ERROR

#### 5.90.2.10 AK\_function\_test()

```
void AK_function_test ( )
```

Function for functions testing.

##### Author

Boris Kišić, updated by Tomislav Ilisevic

##### Returns

No return value

#### 5.90.2.11 AK\_get\_function\_obj\_id()

```
int AK_get_function_obj_id (
    char * function,
    struct list_node * arguments_list )
```

Function that gets obj\_id of a function by name and arguments list (transferred from [trigger.c/drop.c](https://github.com/trigger-c/drop.c)).

##### Author

Unknown, updated by Jurica Hlevnjak - check function arguments included for drop purpose, updated by Tomislav Ilisevic



## Parameters

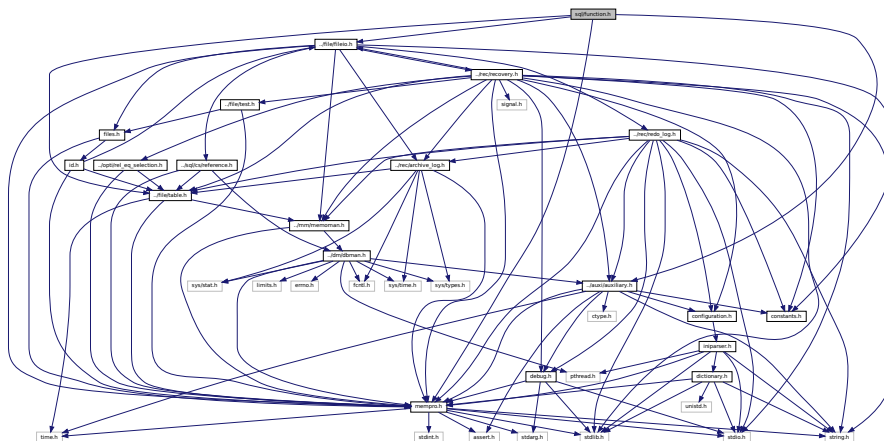
|                        |                      |
|------------------------|----------------------|
| <i>*function</i>       | name of the function |
| <i>*arguments_list</i> | list of arguments    |

## Returns

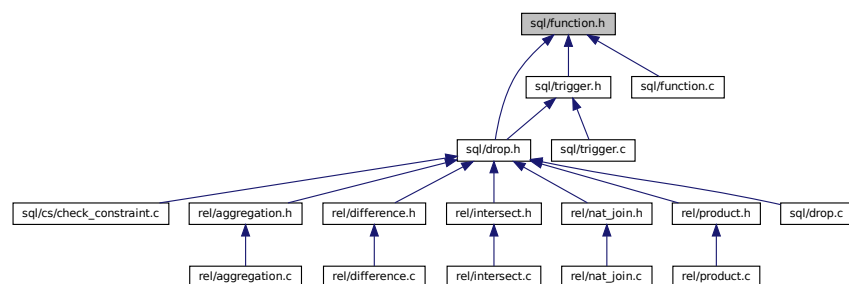
obj\_id of the function or EXIT\_ERROR

## 5.91 sql/function.h File Reference

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../aux/mempro.h"
#include "../aux/auxiliary.h"
Include dependency graph for function.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- int [AK\\_get\\_function\\_obj\\_id](#) (char \*function, struct [list\\_node](#) \*arguments\_list)  
*Function that gets obj\_id of a function by name and arguments list (transferred from [trigger.c/drop.c](#)).*
- int [AK\\_check\\_function\\_arguments](#) (int function\_id, struct [list\\_node](#) \*arguments\_list)  
*Function that checks whether arguments belong to function.*
- int [AK\\_check\\_function\\_arguments\\_type](#) (int function\_id, struct [list\\_node](#) \*args)  
*Function that checks whether arguments belong to function but only checks argument type (not name). Used for drop function.*
- int [AK\\_function\\_add](#) (char \*name, int return\_type, struct [list\\_node](#) \*arguments\_list)  
*Function that adds a function to system table.*
- int [AK\\_function\\_arguments\\_add](#) (int function\_id, int arg\_number, int arg\_type, char \*argname)  
*Function that adds a function argument to system table.*
- int [AK\\_function\\_remove\\_by\\_obj\\_id](#) (int obj\_id)  
*Function removes a function by its obj\_id.*
- int [AK\\_function\\_arguments\\_remove\\_by\\_obj\\_id](#) (int \*obj\_id)  
*Function removes function arguments by function id.*
- int [AK\\_function\\_remove\\_by\\_name](#) (char \*name, struct [list\\_node](#) \*arguments\_list)  
*Function that removes a function from system table by name and arguments.*
- int [AK\\_function\\_rename](#) (char \*name, struct [list\\_node](#) \*arguments\_list, char \*new\_name)  
*Function that changes the function name.*
- int [AK\\_function\\_change\\_return\\_type](#) (char \*name, struct [list\\_node](#) \*arguments\_list, int new\_return\_type)  
*Function that changes the return type.*
- void [AK\\_function\\_test](#) ()  
*Function for functions testing.*

### 5.91.1 Detailed Description

Header file that provides data structures for functions

Header file that provides data structures functions

### 5.91.2 Function Documentation

#### 5.91.2.1 [AK\\_check\\_function\\_arguments\(\)](#)

```
int AK_check_function_arguments (
    int function_id,
    struct list\_node * arguments_list )
```

Function that checks whether arguments belong to function.

#### Author

Boris Kišić

## Parameters

|                        |                    |
|------------------------|--------------------|
| <i>*function_id</i>    | id of the function |
| <i>*arguments_list</i> | list of arguments  |

## Returns

EXIT\_SUCCESS of the function or EXIT\_ERROR

## 5.91.2.2 AK\_check\_function\_arguments\_type()

```
int AK_check_function_arguments_type (
    int function_id,
    struct list_node * args )
```

Function that checks whether arguments belong to function but only checks argument type (not name). Used for drop function.

## Author

Jurica Hlevnjak

## Parameters

|                          |                    |
|--------------------------|--------------------|
| <i>function↔<br/>_id</i> | id of the function |
| <i>args</i>              | function arguments |

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

## 5.91.2.3 AK\_function\_add()

```
int AK_function_add (
    char * name,
    int return_type,
    struct list_node * arguments_list )
```

Function that adds a function to system table.

## Author

Boris Kišić, updated by Tomislav Ilisevic

**Parameters**

|                        |   |
|------------------------|---|
| <i>*name</i>           | name of the function  |
| <i>*return_type</i>    | data type returned from a function - values from 0 to 13 - defined in <a href="#">constants.h</a> |
| <i>*arguments_list</i> | list of function arguments  |

**Returns**

function id or EXIT\_ERROR

**5.91.2.4 AK\_function\_arguments\_add()**

```
int AK_function_arguments_add (
    int function_id,
    int arg_number,
    int arg_type,
    char * argname )
```

Function that adds a function argument to system table.

**Author**

Boris Kišić

**Parameters**

|                     |  |
|---------------------|--|
| <i>*function_id</i> | id of the function to which the argument belongs |
| <i>*arg_number</i>  | number of the argument                           |
| <i>*arg_type</i>    | data type of the argument                        |
| <i>*argname</i>     | name of the argument                             |

**Returns**

function argument id or EXIT\_ERROR

**5.91.2.5 AK\_function\_arguments\_remove\_by\_obj\_id()**

```
int AK_function_arguments_remove_by_obj_id (
    int * obj_id )
```

Function removes function arguments by function id.

**Author**

Boris Kišić

## Parameters

|               |                        |
|---------------|------------------------|
| <i>obj_id</i> | obj_id of the function |
|---------------|------------------------|

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

## 5.91.2.6 AK\_function\_change\_return\_type()

```
int AK_function_change_return_type (
    char * name,
    struct list_node * arguments_list,
    int new_return_type )
```

Function that changes the return type.

## Author

Boris Kišić

## Parameters

|                         |                                     |
|-------------------------|-------------------------------------|
| <i>*name</i>            | name of the function to be modified |
| <i>*arguments_list</i>  | list of function arguments          |
| <i>*new_return_type</i> | new return type                     |

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

## 5.91.2.7 AK\_function\_remove\_by\_name()

```
int AK_function_remove_by_name (
    char * name,
    struct list_node * arguments_list )
```

Function that removes a function from system table by name and arguments.

## Author

Boris Kišić

**Parameters**

|                        |                      |
|------------------------|----------------------|
| <i>*name</i>           | name of the function |
| <i>*arguments_list</i> | list of arguments    |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.91.2.8 AK\_function\_remove\_by\_obj\_id()**

```
int AK_function_remove_by_obj_id (
    int obj_id )
```

Function removes a function by its obj\_id.

**Author**

Boris Kišić

**Parameters**

|               |                        |
|---------------|------------------------|
| <i>obj_id</i> | obj_id of the function |
|---------------|------------------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.91.2.9 AK\_function\_rename()**

```
int AK_function_rename (
    char * name,
    struct list_node * arguments_list,
    char * new_name )
```

Function that changes the function name.

**Author**

Boris Kišić

**Parameters**

|                        |                                     |
|------------------------|-------------------------------------|
| <i>*name</i>           | name of the function to be modified |
| <i>*arguments_list</i> | list of arguments to be modified    |
| <i>*new_name</i>       | new name of the function            |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.91.2.10 AK\_function\_test()**

```
void AK_function_test ( )
```

Function for functions testing.

**Author**

Boris Kišić, updated by Tomislav Ilisevic

**Returns**

No return value

**5.91.2.11 AK\_get\_function\_obj\_id()**

```
int AK_get_function_obj_id (
    char * function,
    struct list_node * arguments_list )
```

Function that gets obj\_id of a function by name and arguments list (transferred from [trigger.c/drop.c](#)).

**Author**

Unknown, updated by Jurica Hlevnjak - check function arguments included for drop purpose, updated by Tomislav Ilisevic

**Parameters**

|                        |                      |
|------------------------|----------------------|
| <i>*function</i>       | name of the function |
| <i>*arguments_list</i> | list of arguments    |

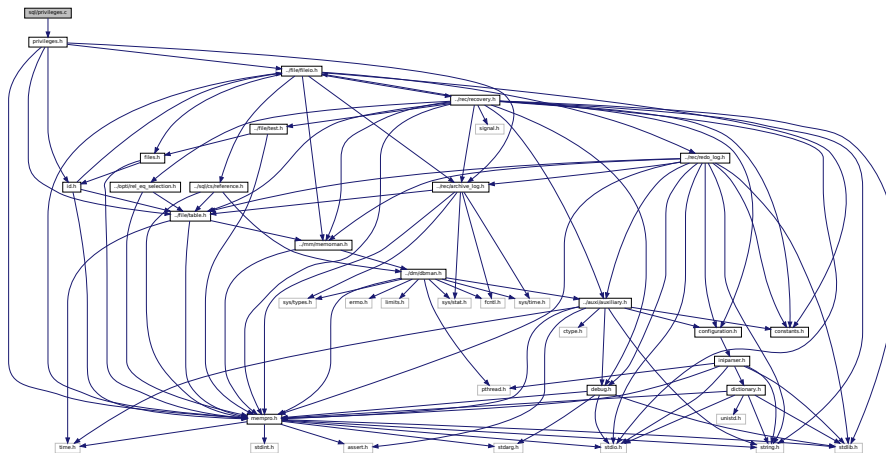
**Returns**

obj\_id of the function or EXIT\_ERROR

**5.92 sql/privileges.c File Reference**

```
#include "privileges.h"
```

Include dependency graph for privileges.c:



## Functions

- `int AK_user_add (char *username, int *password, int set_id)`  
*Inserts a new user in the AK\_user table.*
- `int AK_user_get_id (char *username)`  
*Returns an ID of the given user.*
- `int AK_user_remove_by_name (char *name)`  
*Removes the given user.*
- `int AK_user_rename (char *old_name, char *new_name, int *password)`  
*Renames the given user.*
- `int AK_group_add (char *name, int set_id)`  
*Adds a new group.*
- `int AK_group_get_id (char *name)`  
*Returns an ID of from the given group name.*
- `int AK_group_remove_by_name (char *name)`  
*Removes the given group.*
- `int AK_group_rename (char *old_name, char *new_name)`  
*Renames the given group.*
- `int AK_grant_privilege_user (char *username, char *table, char *right)`  
*Grants specific privilege to the desired user on a given table.*
- `int AK_revoke_privilege_user (char *username, char *table, char *right)`  
*Revokes user's privilege on the given table.*
- `int AK_revoke_all_privileges_user (char *username)`  
*Revokes ALL user's privileges on ALL tables (for DROP user)*
- `int AK_grant_privilege_group (char *groupname, char *table, char *right)`  
*Grants privilege to given group on given table.*
- `int AK_revoke_privilege_group (char *groupname, char *table, char *right)`  
*Revokes group's privilege on the given table.*
- `int AK_revoke_all_privileges_group (char *groupname)`  
*Revokes ALL privileges from the desired group on ALL tables (needed for DROP group)*
- `int AK_add_user_to_group (char *user, char *group)`  
*Puts the desired user in the given group.*
- `int AK_remove_user_from_all_groups (char *user)`



- Removes user from all groups. Used for DROP user.*
  - int [AK\\_remove\\_all\\_users\\_from\\_group](#) (char \*group)  
*Function removes all users from group. Used for DROP group.*
- int [AK\\_check\\_privilege](#) (char \*username, char \*table, char \*privilege)  
*Checks whether the given user has a right for the given operation on the given table.*
- int [AK\\_check\\_user\\_privilege](#) (char \*user)  
*Checks if the user has any privileges or belongs to any group. Used in drop user for restriction.*
- int [AK\\_check\\_group\\_privilege](#) (char \*group)  
*Checks if the group has any privileges. Used in drop group for restriction.*
- void [AK\\_privileges\\_test](#) ()  
*Function that tests all the previous functions.*

### 5.92.1 Detailed Description

Provides functions for privileges

### 5.92.2 Function Documentation

#### 5.92.2.1 AK\_add\_user\_to\_group()

```
int AK_add_user_to_group (  
    char * user,  
    char * group )
```

Puts the desired user in the given group.

#### Author

Kristina Takač, updated by Mario Peroković, added verifying the existence of user in the group, updated by Maja Vračan

#### Parameters

|               |   |
|---------------|---|
| <i>*user</i>  | username of user which will be put in group |
| <i>*group</i> | name of group in which user will be put     |

#### Returns

EXIT\_SUCCESS or EXIT\_ERROR if the user is already in the group

#### 5.92.2.2 AK\_check\_group\_privilege()

```
int AK_check_group_privilege (  
    char * group )
```

Checks if the group has any privileges. Used in drop group for restriction.

#### Author

Jurica Hlevnjak, updated by Lidija Lastavec, updated by Marko Flajšek

#### Parameters

|              |               |
|--------------|---------------|
| <i>group</i> | name of group |
|--------------|---------------|

#### Returns

EXIT\_ERROR or EXIT\_SUCCESS

#### 5.92.2.3 AK\_check\_privilege()

```
int AK_check_privilege (
    char * username,
    char * table,
    char * privilege )
```

Checks whether the given user has a right for the given operation on the given table.

#### Author

Kristina Takač, updated by Marko Flajšek

#### Parameters

|                   |  |
|-------------------|--|
| <i>*user</i>      | username for which we want check privileges                        |
| <i>*table</i>     | name of table for which we want to check whether user has right on |
| <i>*privilege</i> | privilege for which we want to check whether user has right for    |

#### Returns

EXIT\_SUCCESS if user has right, EXIT\_ERROR if user has no right

#### 5.92.2.4 AK\_check\_user\_privilege()

```
int AK_check_user_privilege (
    char * user )
```

Checks if the user has any privileges or belongs to any group. Used in drop user for restriction.

#### Author

Jurica Hlevnjak, updated by Lidija Lastavec

**Parameters**

|             |              |
|-------------|--------------|
| <i>user</i> | name of user |
|-------------|--------------|

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**5.92.2.5 AK\_grant\_privilege\_group()**

```
int AK_grant_privilege_group (
    char * groupname,
    char * table,
    char * right )
```

Grants privilege to given group on given table.

**Author**

Kristina Takač.

**Parameters**

|                   |  |
|-------------------|--|
| <i>*groupname</i> | name of group to which we want to grant privilege              |
| <i>*table</i>     | name of table on which privilege will be granted to user       |
| <i>*right</i>     | type of privilege which will be granted to user on given table |

**Returns**

privilege\_id or EXIT\_ERROR if table or user aren't correct

**5.92.2.6 AK\_grant\_privilege\_user()**

```
int AK_grant_privilege_user (
    char * username,
    char * table,
    char * right )
```

Grants specific privilege to the desired user on a given table.

**Author**

Kristina Takač, updated by Mario Peroković, inserting user id instead of username in AK\_user\_right, updated by Marko Flajšek

**Parameters**

|                  |  |
|------------------|--|
| <i>*username</i> | username of user to whom we want to grant privilege            |
| <i>*table</i>    | name of table on which privilege will be granted to user       |
| <i>*right</i>    | type of privilege which will be granted to user on given table |

**Returns**

privilege\_id or EXIT\_ERROR if table or user aren't correct

**5.92.2.7 AK\_group\_add()**

```
int AK_group_add (
    char * name,
    int set_id )
```

Adds a new group.

**Author**

Kristina Takač, edited by Ljubo Barać

**Parameters**

|               |                             |
|---------------|-----------------------------|
| <i>*name</i>  | name of group to be added   |
| <i>set_id</i> | non default id to be passed |

**Returns**

id of group

**5.92.2.8 AK\_group\_get\_id()**

```
int AK_group_get_id (
    char * name )
```

Returns an ID of from the given group name.

**Author**

Kristina Takač.

**Parameters**

|              |   |
|--------------|---|
| <i>*name</i> | name of group whose id we are looking for |
|--------------|---|

**Returns**

id of group, otherwise EXIT\_ERROR

**5.92.2.9 AK\_group\_remove\_by\_name()**

```
int AK_group_remove_by_name (  
    char * name )
```

Removes the given group.

**Author**

Ljubo Barać

**Parameters**

|             |                                 |
|-------------|---------------------------------|
| <i>name</i> | Name of the group to be removed |
|-------------|---------------------------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.92.2.10 AK\_group\_rename()**

```
int AK_group_rename (  
    char * old_name,  
    char * new_name )
```

Renames the given group.

**Author**

Ljubo Barać, update by Lidija Lastavec

**Parameters**

|                 |                                 |
|-----------------|---------------------------------|
| <i>old_name</i> | Name of the group to be renamed |
| <i>new_name</i> | New name of the group           |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.92.2.11 AK\_privileges\_test()**

```
void AK_privileges_test ( )
```

Function that tests all the previous functions.

**Author**

Kristina Takač, updated by Tomislav Ilisevic, updated by Lidija Lastavec, updated by Marko Flajšek

**Returns**

no return value

**5.92.2.12 AK\_remove\_all\_users\_from\_group()**

```
int AK_remove_all_users_from_group (
    char * group )
```

Function removes all users from group. Used for DROP group.

**Author**

Jurica Hlevnjak, update by Lidija Lastavec

**Parameters**

|              |               |
|--------------|---------------|
| <i>group</i> | name of group |
|--------------|---------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.92.2.13 AK\_remove\_user\_from\_all\_groups()**

```
int AK_remove_user_from_all_groups (
    char * user )
```

Removes user from all groups. Used for DROP user.

**Author**

Jurica Hlevnjak, update by Lidija Lastavec

**Parameters**

|             |              |
|-------------|--------------|
| <i>user</i> | name of user |
|-------------|--------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.92.2.14 AK\_revoke\_all\_privileges\_group()**

```
int AK_revoke_all_privileges_group (  
    char * groupname )
```

Revokes ALL privileges from the desired group on ALL tables (needed for DROP group)

**Author**

Jurica Hlevnjak

**Parameters**

|                  |   |
|------------------|---|
| <i>groupname</i> | name of group from which we want to revoke all privileges |
|------------------|---|

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**5.92.2.15 AK\_revoke\_all\_privileges\_user()**

```
int AK_revoke_all_privileges_user (  
    char * username )
```

Revokes ALL user's privileges on ALL tables (for DROP user)

**Author**

Jurica Hlevnjak, updated by Marko Flajšek

**Parameters**

|                 |   |
|-----------------|---|
| <i>username</i> | name of user from whom we want to revoke all privileges |
|-----------------|---|

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**5.92.2.16 AK\_revoke\_privilege\_group()**

```
int AK_revoke_privilege_group (
    char * groupname,
    char * table,
    char * right )
```

Revokes group's privilege on the given table.

NOTICE: Test 9 isn't currently revoking a privilege since the `obj_id` in the `AK_group_right` table is passing the value of 127. Once the issue #87 on GitHub concerning the data type is solved, the test should be working as expected.

**Author**

Kristina Takač, updated by Mario Peroković - added comparing by table id

**Parameters**

|                   |   |
|-------------------|---|
| <i>*grounamep</i> | name of group which user belongs to                             |
| <i>*table</i>     | name of table on which privilege will be granted to group       |
| <i>*right</i>     | type of privilege which will be granted to group on given table |

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**5.92.2.17 AK\_revoke\_privilege\_user()**

```
int AK_revoke_privilege_user (
    char * username,
    char * table,
    char * right )
```

Revokes user's privilege on the given table.

NOTICE: Test 12 isn't currently revoking a privilege since the `obj_id` in the `AK_group_right` table is passing the value of 127. Once the issue #87 on GitHub concerning the data type is solved, the test should be working as expected.

**Author**

Kristina Takač, updated by Mario Peroković - added comparing by table id, and use of `user_id` in `AK_user_right`



**Parameters**

|                  |  |
|------------------|--|
| <i>*username</i> | username of user to whom we want to grant privilege              |
| <i>*table</i>    | name of table on which privilege will be revoked from user       |
| <i>*right</i>    | type of privilege which will be revoked from user on given table |

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**5.92.2.18 AK\_user\_add()**

```
int AK_user_add (
    char * username,
    int * password,
    int set_id )
```

Inserts a new user in the AK\_user table.

**Author**

Kristina Takač.

**Parameters**

|                  |                              |
|------------------|------------------------------|
| <i>*username</i> | username of user to be added |
| <i>*password</i> | password of user to be added |
| <i>set_id</i>    | obj_id of the new user       |

**Returns**

user\_id

**5.92.2.19 AK\_user\_get\_id()**

```
int AK_user_get_id (
    char * username )
```

Returns an ID of the given user.

**Author**

Kristina Takač.

**Parameters**

|                  |  |
|------------------|--|
| <i>*username</i> | username of user whose id we are looking for |
|------------------|--|

**Returns**

user\_id, otherwise EXIT\_ERROR

**5.92.2.20 AK\_user\_remove\_by\_name()**

```
int AK_user_remove_by_name (
    char * name )
```

Removes the given user.

**Author**

Ljubo Barać

**Parameters**

|             |                                |
|-------------|--------------------------------|
| <i>name</i> | Name of the user to be removed |
|-------------|--------------------------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.92.2.21 AK\_user\_rename()**

```
int AK_user_rename (
    char * old_name,
    char * new_name,
    int * password )
```

Renames the given user.

**Author**

Ljubo Barać, update by Lidija Lastavec, update by Marko Flajšek

**Parameters**

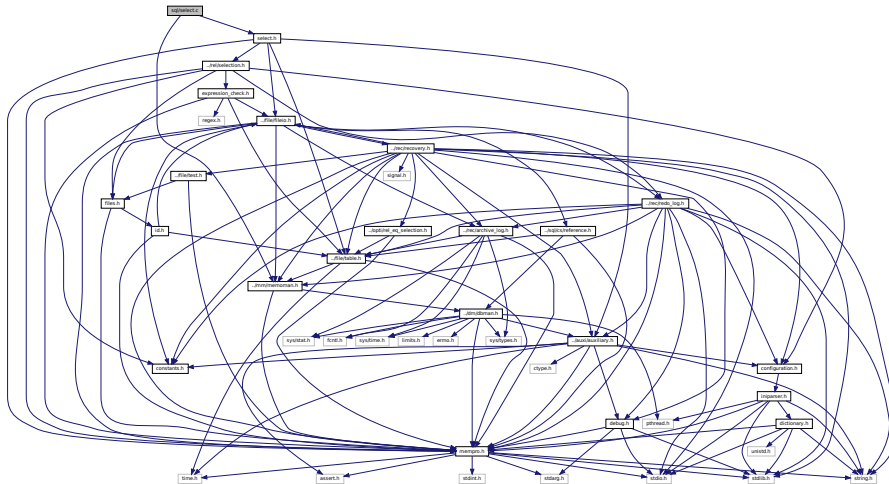
|                 |   |
|-----------------|---|
| <i>old_name</i> | Name of the user to be renamed                          |
| <i>new_name</i> | New name of the user                                    |
| <i>password</i> | Password of the user to be renamed (should be provided) |

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

## 5.93 sql/select.c File Reference

```
#include "select.h"
#include "../mm/memoman.h"
Include dependency graph for select.c:
```



## Functions

- int **AK\_select** (char \*srcTable, char \*destTable, struct **list\_node** \*attributes, struct **list\_node** \*condition)  
*Function that implements SELECT relational operator.*
- void **AK\_select\_test** ()  
*Function for testing the implementation.*

### 5.93.1 Detailed Description

Provides functions for SELECT relational operator

### 5.93.2 Function Documentation

### 5.93.2.1 AK\_select()

```
int AK_select (
    char * srcTable,
    char * destTable,
    struct list_node * attributes,
    struct list_node * condition )
```

Function that implements SELECT relational operator.

**Author**

Renata Mesaros



## Functions

- int [AK\\_trigger\\_save\\_conditions](#) (int trigger, struct [list\\_node](#) \*condition)  
*Saves conditions for a trigger.*
- int [AK\\_trigger\\_add](#) (char \*name, char \*event, struct [list\\_node](#) \*condition, char \*table, char \*function)  
*Function that adds a trigger to system table.*
- int [AK\\_trigger\\_get\\_id](#) (char \*name, char \*table)  
*Function that gets obj\_id of a trigger defined by name and table.*
- int [AK\\_trigger\\_remove\\_by\\_name](#) (char \*name, char \*table)  
*Function that removes a trigger from system table by name.*
- int [AK\\_trigger\\_remove\\_by\\_obj\\_id](#) (int obj\_id)  
*Function removes a trigger by its obj\_id.*
- int [AK\\_trigger\\_edit](#) (char \*name, char \*event, struct [list\\_node](#) \*condition, char \*table, char \*function)  
*Function edits information about the trigger in system table. In order to identify the trigger, either obj\_id or table and name parameters should be defined. The other options should be set to NULL. Values of parameters that aren't changing can be left NULL. If conditions are to be removed, condition parameter should hold an empty list.*
- struct [list\\_node](#) \* [AK\\_trigger\\_get\\_conditions](#) (int trigger)  
*Function gets postfix list of conditions for the trigger (compatible with selection)*
- int [AK\\_trigger\\_rename](#) (char \*old\_name, char \*new\_name, char \*table)  
*Function renames the trigger.*
- void [AK\\_trigger\\_test](#) ()  
*Function for trigger testing.*

### 5.94.1 Detailed Description

Provides functions for triggers

### 5.94.2 Function Documentation

#### 5.94.2.1 AK\_trigger\_add()

```
int AK_trigger_add (
    char * name,
    char * event,
    struct list\_node * condition,
    char * table,
    char * function )
```

Function that adds a trigger to system table.

#### Author

Unknown

#### Parameters

|                   |  |
|-------------------|--|
| <i>*name</i>      | name of the trigger  |
| <i>*event</i>     | event that calls the trigger - this should perhaps be an integer with defined constants... |
| <i>*condition</i> | AK_list list of conditions in postfix  |
| <i>*table</i>     | name of the table trigger is hooked on   |
| <i>*function</i>  | function that is being called by the trigger   |

**Returns**

trigger id or EXIT\_ERROR

**5.94.2.2 AK\_trigger\_edit()**

```
int AK_trigger_edit (
    char * name,
    char * event,
    struct list_node * condition,
    char * table,
    char * function )
```

Function edits information about the trigger in system table. In order to identify the trigger, either obj\_id or table and name parameters should be defined. The other options should be set to NULL. Values of parameters that aren't changing can be left NULL. If conditions are to be removed, condition parameter should hold an empty list.

**Author**

Unknown

**Parameters**

|                   |   |
|-------------------|---|
| <i>*name</i>      | name of the trigger (or NULL if using obj_id)   |
| <i>*event</i>     | event of the trigger (or NULL if it isn't changing)   |
| <i>*condition</i> | list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed) |
| <i>*table</i>     | name of the connected table (or NULL id using obj_id)   |
| <i>*function</i>  | name of the connected function (or NULL if it isn't changing)   |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.94.2.3 AK\_trigger\_get\_conditions()**

```
struct list_node* AK_trigger_get_conditions (
    int trigger )
```

Function gets postfix list of conditions for the trigger (compatible with selection)

**Author**

Unknown, updated by Mario Peroković

**Parameters**

|                |                       |
|----------------|-----------------------|
| <i>trigger</i> | obj_id of the trigger |
|----------------|-----------------------|

**Returns**

list of conditions for the trigger

**5.94.2.4 AK\_trigger\_get\_id()**

```
int AK_trigger_get_id (
    char * name,
    char * table )
```

Function that gets obj\_id of a trigger defined by name and table.

**Author****Parameters**

|               |  |
|---------------|--|
| <i>*name</i>  | name of the trigger                              |
| <i>*table</i> | name of the table on which the trigger is hooked |

**Returns**

obj\_id of the trigger or EXIT\_ERROR

**5.94.2.5 AK\_trigger\_remove\_by\_name()**

```
int AK_trigger_remove_by_name (
    char * name,
    char * table )
```

Function that removes a trigger from system table by name.

**Author**

Unknown

**Parameters**

|               |                     |
|---------------|---------------------|
| <i>*name</i>  | name of the trigger |
| <i>*table</i> | name of the table   |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.94.2.6 AK\_trigger\_remove\_by\_obj\_id()**

```
int AK_trigger_remove_by_obj_id (  
    int obj_id )
```

Function removes a trigger by its obj\_id.

**Author**

Unknown

**Parameters**

|               |                       |
|---------------|-----------------------|
| <i>obj_id</i> | obj_id of the trigger |
|---------------|-----------------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.94.2.7 AK\_trigger\_rename()**

```
int AK_trigger_rename (  
    char * old_name,  
    char * new_name,  
    char * table )
```

Function renames the trigger.

**Author**

Ljubo Barać

**Parameters**

|                 |                                   |
|-----------------|-----------------------------------|
| <i>old_name</i> | Name of the trigger to be renamed |
| <i>new_name</i> | New name of the trigger           |



**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.94.2.8 AK\_trigger\_save\_conditions()**

```
int AK_trigger_save_conditions (
    int trigger,
    struct list_node * condition )
```

Saves conditions for a trigger.

**Author**

Unknown, updated by Mario Peroković, check if data is TYPE\_INT

**Parameters**

|                   |                                   |
|-------------------|-----------------------------------|
| <i>trigger</i>    | obj_id of the trigger in question |
| <i>*condition</i> | AK_list list of conditions        |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.94.2.9 AK\_trigger\_test()**

```
void AK_trigger_test ( )
```

Function for trigger testing.

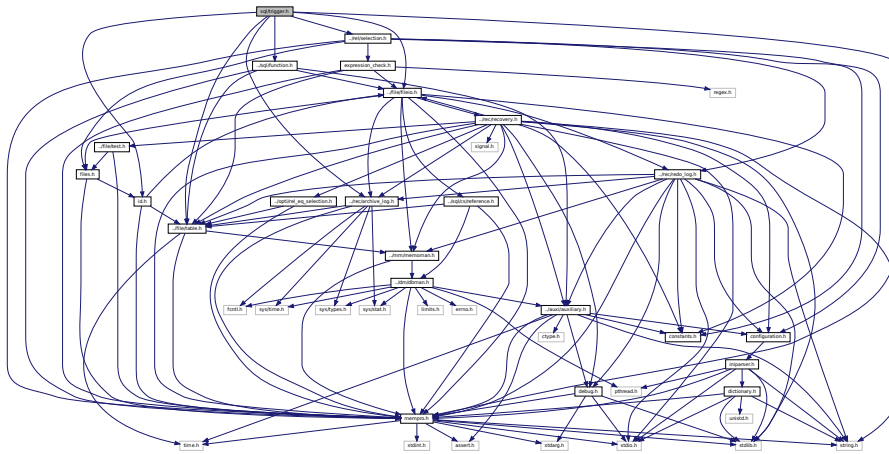
**Author**

Unknown

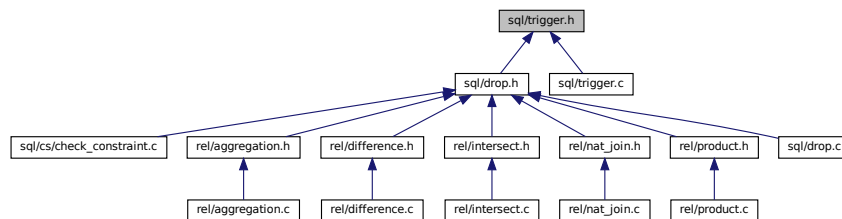
**5.95 sql/trigger.h File Reference**

```
#include "../rec/archive_log.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../file/id.h"
#include "../sql/function.h"
#include "../rel/selection.h"
```

```
#include "../auxi/mempro.h"
Include dependency graph for trigger.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- `int AK_trigger_save_conditions (int trigger, struct list_node *condition)`  
Saves conditions for a trigger.
- `int AK_trigger_add (char *name, char *event, struct list_node *condition, char *table, char *function)`  
Function that adds a trigger to system table.
- `int AK_trigger_get_id (char *name, char *table)`  
Function that gets `obj_id` of a trigger defined by name and table.
- `int AK_trigger_remove_by_name (char *name, char *table)`  
Function that removes a trigger from system table by name.
- `int AK_trigger_remove_by_obj_id (int obj_id)`  
Function removes a trigger by its `obj_id`.
- `int AK_trigger_edit (char *name, char *event, struct list_node *condition, char *table, char *function)`  
Function edits information about the trigger in system table. In order to identify the trigger, either `obj_id` or table and name parameters should be defined. The other options should be set to NULL. Values of parameters that aren't changing can be left NULL. If conditions are to be removed, condition parameter should hold an empty list.
- `struct list_node * AK_trigger_get_conditions (int trigger)`  
Function gets postfix list of conditions for the trigger (compatible with selection)
- `int AK_trigger_rename (char *old_name, char *new_name, char *table)`  
Function renames the trigger.
- `void AK_trigger_test ()`  
Function for trigger testing.

## 5.95.1 Detailed Description

Header file that provides data structures triggers

## 5.95.2 Function Documentation

### 5.95.2.1 AK\_trigger\_add()

```
int AK_trigger_add (
    char * name,
    char * event,
    struct list_node * condition,
    char * table,
    char * function )
```

Function that adds a trigger to system table.

#### Author

Unknown

#### Parameters

|                   |  |
|-------------------|--|
| <i>*name</i>      | name of the trigger  |
| <i>*event</i>     | event that calls the trigger - this should perhaps be an integer with defined constants... |
| <i>*condition</i> | AK_list list of conditions in postfix  |
| <i>*table</i>     | name of the table trigger is hooked on   |
| <i>*function</i>  | function that is being called by the trigger   |

#### Returns

trigger id or EXIT\_ERROR

### 5.95.2.2 AK\_trigger\_edit()

```
int AK_trigger_edit (
    char * name,
    char * event,
    struct list_node * condition,
    char * table,
    char * function )
```

Function edits information about the trigger in system table. In order to identify the trigger, either obj\_id or table and name parameters should be defined. The other options should be set to NULL. Values of parameters that aren't changing can be left NULL. If conditions are to be removed, condition parameter should hold an empty list.

**Author**

Unknown

**Parameters**

|                   |   |
|-------------------|---|
| <i>*name</i>      | name of the trigger (or NULL if using obj_id)   |
| <i>*event</i>     | event of the trigger (or NULL if it isn't changing)   |
| <i>*condition</i> | list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed) |
| <i>*table</i>     | name of the connected table (or NULL id using obj_id)   |
| <i>*function</i>  | name of the connected function (or NULL if it isn't changing)   |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.95.2.3 AK\_trigger\_get\_conditions()**

```
struct list_node* AK_trigger_get_conditions (
    int trigger )
```

Function gets postfix list of conditions for the trigger (compatible with selection)

**Author**

Unknown, updated by Mario Peroković

**Parameters**

|                |                       |
|----------------|-----------------------|
| <i>trigger</i> | obj_id of the trigger |
|----------------|-----------------------|

**Returns**

list of conditions for the trigger

**5.95.2.4 AK\_trigger\_get\_id()**

```
int AK_trigger_get_id (
    char * name,
    char * table )
```

Function that gets obj\_id of a trigger defined by name and table.

**Author**

## Parameters

|               |  |
|---------------|--|
| <i>*name</i>  | name of the trigger                              |
| <i>*table</i> | name of the table on which the trigger is hooked |

## Returns

obj\_id of the trigger or EXIT\_ERROR

## 5.95.2.5 AK\_trigger\_remove\_by\_name()

```
int AK_trigger_remove_by_name (
    char * name,
    char * table )
```

Function that removes a trigger from system table by name.

## Author

Unknown

## Parameters

|               |                     |
|---------------|---------------------|
| <i>*name</i>  | name of the trigger |
| <i>*table</i> | name of the table   |

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

## 5.95.2.6 AK\_trigger\_remove\_by\_obj\_id()

```
int AK_trigger_remove_by_obj_id (
    int obj_id )
```

Function removes a trigger by its obj\_id.

## Author

Unknown

**Parameters**

|               |                       |
|---------------|-----------------------|
| <i>obj_id</i> | obj_id of the trigger |
|---------------|-----------------------|

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.95.2.7 AK\_trigger\_rename()**

```
int AK_trigger_rename (
    char * old_name,
    char * new_name,
    char * table )
```

Function renames the trigger.

**Author**

Ljubo Barać

**Parameters**

|                 |                                   |
|-----------------|-----------------------------------|
| <i>old_name</i> | Name of the trigger to be renamed |
| <i>new_name</i> | New name of the trigger           |

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**5.95.2.8 AK\_trigger\_save\_conditions()**

```
int AK_trigger_save_conditions (
    int trigger,
    struct list_node * condition )
```

Saves conditions for a trigger.

**Author**

Unknown, updated by Mario Peroković, check if data is TYPE\_INT



## Functions

- int [AK\\_get\\_view\\_obj\\_id](#) (char \*name)  
*Finds an object's id by its name.*
- char \* [AK\\_get\\_view\\_query](#) (char \*name)  
*Returns a query by its name.*
- char \* [AK\\_get\\_rel\\_exp](#) (char \*name)  
*Returns a relation expression by its name param name name of the view.*
- int [AK\\_view\\_add](#) (char \*name, char \*query, char \*rel\_exp, int set\_id)  
*Adds a new view to the view table with the corresponding name and value (view query); set\_id is optional, if it's not set, the system will determine the new id automatically.*
- int [AK\\_view\\_remove\\_by\\_obj\\_id](#) (int obj\_id)  
*Removes the view by its object id.*
- int [AK\\_view\\_remove\\_by\\_name](#) (char \*name)  
*Removes the view by its name by identifying the view's id and passing id to AK\_view\_remove\_by\_obj\_id.*
- int [AK\\_view\\_rename](#) (char \*name, char \*new\_name)  
*Renames a view (based on it's name) from "name" to "new\_name".*
- int [AK\\_view\\_change\\_query](#) (char \*name, char \*query, char \*rel\_exp)  
*Changes the query for a view (determined by it's name) to "query".*
- void [AK\\_view\\_test](#) ()  
*A testing function for [view.c](#) functions.*

### 5.96.1 Detailed Description

Provides functions for views

### 5.96.2 Function Documentation

#### 5.96.2.1 [AK\\_get\\_rel\\_exp\(\)](#)

```
char* AK_get_rel_exp (
    char * name )
```

Returns a relation expression by its name param name name of the view.

#### Author

Danko Sačer

#### Returns

rel\_exp string or EXIT\_ERROR

#### 5.96.2.2 [AK\\_get\\_view\\_obj\\_id\(\)](#)

```
int AK_get_view_obj_id (
    char * name )
```

Finds an object's id by its name.

#### Author

Kresimir Ivkovic



## Parameters

|             |                  |
|-------------|------------------|
| <i>name</i> | name of the view |
|-------------|------------------|

## Returns

View's id or EXIT\_ERROR

## 5.96.2.3 AK\_get\_view\_query()

```
char* AK_get_view_query (  
    char * name )
```

Returns a query by its name.

## Author

Danko Sačer

## Parameters

|             |                  |
|-------------|------------------|
| <i>name</i> | name of the view |
|-------------|------------------|

## Returns

query string or EXIT\_ERROR

## 5.96.2.4 AK\_view\_add()

```
int AK_view_add (  
    char * name,  
    char * query,  
    char * rel_exp,  
    int set_id )
```

Adds a new view to the view table with the corresponding name and value (view query); *set\_id* is optional, if it's not set, the system will determine the new id automatically.

## Author

Kresimir Ivkovic

## Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>name</i>    | name og the view                |
| <i>query</i>   | query of the view               |
| <i>rel_exp</i> | relation expression of the view |
| <i>set_id</i>  | id of view                      |

**Returns**

Id of the newly inserted view

**5.96.2.5 AK\_view\_change\_query()**

```
int AK_view_change_query (
    char * name,
    char * query,
    char * rel_exp )
```

Changes the query for a view (determined by it's name) to "query".

**Author**

Kresimir Ivkovic

**Parameters**

|                |                                 |
|----------------|---------------------------------|
| <i>name</i>    | of the query                    |
| <i>query</i>   | new query of the view           |
| <i>rel_exp</i> | relation expression of the view |

**Returns**

error or success

**5.96.2.6 AK\_view\_remove\_by\_name()**

```
int AK_view_remove_by_name (
    char * name )
```

Removes the view by its name by identifying the view's id and passing id to AK\_view\_remove\_by\_obj\_id.

**Author**

Kresimir Ivkovic

**Parameters**

|             |                  |
|-------------|------------------|
| <i>name</i> | name of the view |
|-------------|------------------|

**Returns**

Result of AK\_view\_remove\_by\_obj\_id or EXIT\_ERROR if no id is found

**5.96.2.7 AK\_view\_remove\_by\_obj\_id()**

```
int AK_view_remove_by_obj_id (
    int obj_id )
```

Removes the view by its object id.

**Author**

Kresimir Ivkovic

**Parameters**

|               |                       |
|---------------|-----------------------|
| <i>obj_id</i> | object id of the view |
|---------------|-----------------------|

**Returns**

Result of AK\_delete\_row for the view (success or error)

**5.96.2.8 AK\_view\_rename()**

```
int AK_view_rename (
    char * name,
    char * new_name )
```

Renames a view (based on it's name) from "name" to "new\_name".

**Author**

Kresimir Ivkovic

**Parameters**

|                 |                      |
|-----------------|----------------------|
| <i>name</i>     | name of the view     |
| <i>new_name</i> | new name of the view |

**Returns**

error or success

### 5.96.2.9 AK\_view\_test()

```
void AK_view_test ( )
```

A testing function for [view.c](#) functions.

#### Author

Kresimir Ivkovic, updated by Lidija Lastavec

## 5.97 tools/comments.py File Reference

```
//!
```

### Functions

- def [comments.getcommentsFiles](#) ()  
*This function is searching for file that ends with either .py extension or .c extension and appending the same in constant cFiles/pyFiles.*
- def [comments.detectLanguage](#) ()  
*Function is detecting language (is it croatian or alike) of a newly created commentsFile.*
- def [comments.makeCommentsFile](#) ()  
*Function is parsing comments from file with .c extension and .py extension.*

### Variables

- string **comments.commentsFile** = "all\_comments.tmp"
- list **comments.cFiles** = []
- list **comments.pyFiles** = []

### 5.97.1 Detailed Description

```
//!
```

## 5.98 tools/getFiles.sh File Reference

Finding all files that ends with extension .py or .c and storing them into file.txt.

### 5.98.1 Detailed Description

Finding all files that ends with extension .py or .c and storing them into file.txt.



## Functions

- int [AK\\_memory\\_block\\_hash](#) (int blockMemoryAddress)  
*Calculates hash value for a given memory address. Hash values are used to identify location of locked resources.*
- [AK\\_transaction\\_elem\\_P AK\\_search\\_existing\\_link\\_for\\_hook](#) (int blockAddress)  
*Searches for a existing entry in hash list of active blocks.*
- [AK\\_transaction\\_elem\\_P AK\\_search\\_empty\\_link\\_for\\_hook](#) (int blockAddress)  
*Searches for a empty link for new active block, helper method in case of address collision.*
- [AK\\_transaction\\_elem\\_P AK\\_add\\_hash\\_entry\\_list](#) (int blockAddress, int type)  
*Adds an element to the doubly linked list.*
- int [AK\\_delete\\_hash\\_entry\\_list](#) (int blockAddress)  
*Deletes a specific element in the lockTable doubly linked list.*
- [AK\\_transaction\\_lock\\_elem\\_P AK\\_search\\_lock\\_entry\\_list\\_by\\_key](#) ([AK\\_transaction\\_elem\\_P](#) Lockslist, int memoryAddress, pthread\_t id)  
*Searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.*
- int [AK\\_delete\\_lock\\_entry\\_list](#) (int blockAddress, pthread\_t id)  
*Deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.*
- int [AK\\_isLock\\_waiting](#) ([AK\\_transaction\\_elem\\_P](#) lockHolder, int type, pthread\_t transactionId, [AK\\_transaction\\_lock\\_elem\\_P](#) lock)  
*Based on the parameters puts an transaction action in waiting phase or let's the transaction do it's actions.*
- [AK\\_transaction\\_lock\\_elem\\_P AK\\_add\\_lock](#) ([AK\\_transaction\\_elem\\_P](#) HashList, int type, pthread\_t transactionId)  
*Adds an element to the locks doubly linked list.*
- [AK\\_transaction\\_lock\\_elem\\_P AK\\_create\\_lock](#) (int blockAddress, int type, pthread\_t transactionId)  
*Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.*
- int [AK\\_acquire\\_lock](#) (int memoryAddress, int type, pthread\_t transactionId)  
*Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.*
- void [AK\\_release\\_locks](#) ([AK\\_memoryAddresses\\_link](#) addressesTmp, pthread\_t transactionId)  
*Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .*
- int [AK\\_get\\_memory\\_blocks](#) (char \*tblName, [AK\\_memoryAddresses\\_link](#) addressList)  
*Method that appends all addresses affected by the transaction.*
- int [AK\\_execute\\_commands](#) (command \*commandArray, int lengthOfArray)  
*Method that is called in a separate thread that is responsible for acquiring locks releasing them and finding the associated block addresses.*
- void \* [AK\\_execute\\_transaction](#) (void \*params)  
*thread start point all relevant functions are called from this function. It acts as an intermediary between the main thread and other threads*
- int [AK\\_remove\\_transaction\\_thread](#) (pthread\_t transaction\_thread)  
*Function for deleting one of active threads from array of all active transactions threads.*
- int [AK\\_create\\_new\\_transaction\\_thread](#) ([AK\\_transaction\\_data](#) \*transaction\_data)  
*Function for creating new thread. Function also adds thread ID to pthread\_t array.*
- void [AK\\_transaction\\_manager](#) (command \*commandArray, int lengthOfArray)  
*method that receives all the data and gives an id to that data and starts a thread that executes the transaction*
- int [AK\\_transaction\\_register\\_observer](#) ([AK\\_observable\\_transaction](#) \*observable\_transaction, [AK\\_observer](#) \*observer)  
*Function for registering new observer of AK\_observable\_transaction type.*
- int [AK\\_transaction\\_unregister\\_observer](#) ([AK\\_observable\\_transaction](#) \*observable\_transaction, [AK\\_observer](#) \*observer)  
*Function for unregistering observer from AK\_observable\_transaction type.*
- void [handle\\_transaction\\_notify](#) ([AK\\_observer\\_lock](#) \*observer\_lock)

- Function for handling AK\_observable\_transaction notify. Function is associated to some observer instance.*

  - void [AK\\_on\\_observable\\_notify](#) (void \*observer, void \*observable, AK\_ObservableType\_Enum type)

*Function for handling notify from some observable type.*

  - void [AK\\_on\\_transaction\\_end](#) (pthread\_t transaction\_thread)

*Function for handling event when some transaction is finished.*

  - void [AK\\_on\\_all\\_transactions\\_end](#) ()

*Function for handling event when all transactions are finished.*

  - void [AK\\_on\\_lock\\_release](#) ()

*Function for handling event when one of lock is released.*

  - void [AK\\_handle\\_observable\\_transaction\\_action](#) (NoticeType \*noticeType)

*Function for handling action which is called from observable\_transaction type.*

  - void [AK\\_lock\\_released](#) ()

*Function which is called when lock is released.*

  - void [AK\\_transaction\\_finished](#) ()

*Function which is called when some transaction is finished.*

  - void [AK\\_all\\_transactions\\_finished](#) ()

*Function which is called when all transactions are finished.*

  - [AK\\_observable\\_transaction](#) \* [AK\\_init\\_observable\\_transaction](#) ()

*Function for initialization of AK\_observable\_transaction type.*

  - [AK\\_observer\\_lock](#) \* [AK\\_init\\_observer\\_lock](#) ()

*Function for initialization of AK\_observer\_lock type.*

  - void [AK\\_test\\_Transaction](#) ()

## Variables

- [AK\\_transaction\\_list](#) **LockTable** [[NUMBER\\_OF\\_KEYS](#)]
- pthread\_mutex\_t **accessLockMutex** = PTHREAD\_MUTEX\_INITIALIZER
- pthread\_mutex\_t **acquireLockMutex** = PTHREAD\_MUTEX\_INITIALIZER
- pthread\_mutex\_t **newTransactionLockMutex** = PTHREAD\_MUTEX\_INITIALIZER
- pthread\_mutex\_t **endTransationTestLockMutex** = PTHREAD\_MUTEX\_INITIALIZER
- pthread\_cond\_t **cond\_lock** = PTHREAD\_COND\_INITIALIZER
- [AK\\_observable\\_transaction](#) \* **observable\_transaction**
- pthread\_t **activeThreads** [[MAX\\_ACTIVE\\_TRANSACTIONS\\_COUNT](#)]
- int **activeTransactionsCount** = 0
- int **transactionsCount** = 0

### 5.102.1 Detailed Description

Defines functions for transaction execution

### 5.102.2 Function Documentation

### 5.102.2.1 AK\_acquire\_lock()

```
int AK_acquire_lock (
    int memoryAddress,
    int type,
    pthread_t transactionId )
```

Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.

#### Author

Frane Jakelić updated by Ivan Pusic

**Todo** Implement a better deadlock detection. This method uses a very simple approach. It waits for 60sec before it restarts a transaction.

#### Parameters

|                      |  |
|----------------------|--|
| <i>memoryAddress</i> | integer representation of memory address.      |
| <i>type</i>          | of lock issued to the provided memory address. |
| <i>transactionId</i> | integer representation of transaction id.      |

#### Returns

OK or NOT\_OK based on the success of the function.

```
##### Lock Granted after wait##-----## Lock ID:lu TYPE:i ##-----## LockedAddress:i #####",
(unsigned long)lock->TransactionId, lock->lock_type, memoryAddress); */
```

```
##### Lock Granted ##-----## Lock ID:lu TYPE:i ##-----## LockedAddress↔
:i #####", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress); */
```

### 5.102.2.2 AK\_add\_hash\_entry\_list()

```
AK_transaction_elem_P AK_add_hash_entry_list (
    int blockAddress,
    int type )
```

Adds an element to the doubly linked list.

#### Author

Frane Jakelić

#### Parameters

|                     |  |
|---------------------|--|
| <i>blockAddress</i> | integer representation of memory address.      |
| <i>type</i>         | of lock issued to the provided memory address. |



**Returns**

pointer to the newly created doubly linked element.

**5.102.2.3 AK\_add\_lock()**

```
AK_transaction_lock_elem_P AK_add_lock (
    AK_transaction_elem_P HashList,
    int type,
    pthread_t transactionId )
```

Adds an element to the locks doubly linked list.

**Author**

Frane Jakelić

**Parameters**

|                      |  |
|----------------------|--|
| <i>memoryAddress</i> | integer representation of memory address.      |
| <i>type</i>          | of lock issued to the provided memory address. |
| <i>transactionId</i> | integer representation of transaction id.      |

**Returns**

pointer to the newly created Locks doubly linked element.

**5.102.2.4 AK\_all\_transactions\_finished()**

```
void AK_all_transactions_finished ( )
```

Function which is called when all transactions are finished.

**Author**

Ivan Pusic

**5.102.2.5 AK\_create\_lock()**

```
AK_transaction_lock_elem_P AK_create_lock (
    int blockAddress,
    int type,
    pthread_t transactionId )
```

Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.

**Author**

Frane Jakelić

**Parameters**

|                      |  |
|----------------------|--|
| <i>memoryAddress</i> | integer representation of memory address.      |
| <i>type</i>          | of lock issued to the provided memory address. |
| <i>transactionId</i> | integer representation of transaction id.      |

**Returns**

pointer to the newly created Locks doubly linked element.

**5.102.2.6 AK\_create\_new\_transaction\_thread()**

```
int AK_create_new_transaction_thread (
    AK_transaction_data * transaction_data )
```

Function for creating new thread. Function also adds thread ID to pthread\_t array.

**Author**

Ivan Pusic

**Parameters**

|                         |                                |
|-------------------------|--------------------------------|
| <i>transaction_data</i> | Data for executing transaction |
|-------------------------|--------------------------------|

**Returns**

Exit status (OK or NOT\_OK)

**5.102.2.7 AK\_delete\_hash\_entry\_list()**

```
int AK_delete_hash_entry_list (
    int blockAddress )
```

Deletes a specific element in the lockTable doubly linked list.

**Author**

Frane Jakelić

**Parameters**

|                     |   |
|---------------------|---|
| <i>blockAddress</i> | integer representation of memory address. |
|---------------------|---|

**Returns**

integer OK or NOT\_OK based on success of finding the specific element in the list.

**5.102.2.8 AK\_delete\_lock\_entry\_list()**

```
int AK_delete_lock_entry_list (
    int blockAddress,
    pthread_t id )
```

Deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.

**Author**

Frane Jakelić

**Parameters**

|                     |   |
|---------------------|---|
| <i>blockAddress</i> | integer representation of memory address. |
| <i>id</i>           | integer representation of transaction id. |

**Returns**

int OK or NOT\_OK based on success of finding the specific element in the list.

**5.102.2.9 AK\_execute\_commands()**

```
int AK_execute_commands (
    command * commandArray,
    int lengthOfArray )
```

Method that is called in a separate thread that is responsible for acquiring locks releasing them and finding the associated block addresses.

**Author**

Frane Jakelić updated by Ivan Pusic

**Todo** Check multithreading, check if it's working correctly

**Parameters**

|                      |   |
|----------------------|---|
| <i>commandArray</i>  | array filled with commands that need to be secured using transactions |
| <i>lengthOfArray</i> | length of commandArray  |
| <i>transactionId</i> | associated with the transaction                                       |

**Returns**

ABORT or COMMIT based on the success of the function.

**5.102.2.10 AK\_execute\_transaction()**

```
void* AK_execute_transaction (
    void * params )
```

thread start point all relevant functions are called from this function. It acts as an intermediary between the main thread and other threads

**Author**

Frane Jakelić updated by Ivan Pusic

**Parameters**

|             |  |
|-------------|--|
| <i>data</i> | transmitted to the thread from the main thread |
|-------------|--|

**5.102.2.11 AK\_get\_memory\_blocks()**

```
int AK_get_memory_blocks (
    char * tblName,
    AK_memoryAddresses_link addressList )
```

Method that appends all addresses affected by the transaction.

**Author**

Frane Jakelić

**Parameters**

|                    |  |
|--------------------|--|
| <i>addressList</i> | pointer to the linked list where the addresses are stored. |
| <i>tblName</i>     | table name used in the transaction                         |

**Returns**

OK or NOT\_OK based on the success of the function.

#### 5.102.2.12 AK\_handle\_observable\_transaction\_action()

```
void AK_handle_observable_transaction_action (
    NoticeType * noticeType )
```

Function for handling action which is called from [observable\\_transaction](#) type.

##### Author

Ivan Pusic

##### Parameters

|                   |                        |
|-------------------|------------------------|
| <i>noticeType</i> | Type of action (event) |
|-------------------|------------------------|

#### 5.102.2.13 AK\_init\_observable\_transaction()

```
AK_observable_transaction* AK_init_observable_transaction ( )
```

Function for initialization of [AK\\_observable\\_transaction](#) type.

##### Author

Ivan Pusic

##### Returns

Pointer to new [AK\\_observable\\_transaction](#) instance

#### 5.102.2.14 AK\_init\_observer\_lock()

```
AK_observer_lock* AK_init_observer_lock ( )
```

Function for initialization of [AK\\_observer\\_lock](#) type.

##### Author

Ivan Pusic

##### Returns

Pointer to new [AK\\_observer\\_lock](#) instance

#### 5.102.2.15 AK\_isLock\_waiting()

```
int AK_isLock_waiting (
    AK_transaction_elem_P lockHolder,
    int type,
    pthread_t transactionId,
    AK_transaction_lock_elem_P lock )
```

Based on the parameters puts an transaction action in waiting phase or let's the transaction do it's actions.

#### Author

Frane Jakelić updated by Ivan Pusic

#### Parameters

|                      |   |
|----------------------|---|
| <i>lockHolder</i>    | pointer to the hash list entry that is entitled to the specific memory address. |
| <i>type</i>          | of lock issued to the provided memory address.                                  |
| <i>transactionId</i> | integer representation of transaction id.                                       |
| <i>lock</i>          | pointer to the lock element that is being tested.                               |

#### Returns

int PASS\_LOCK\_QUEUE or WAIT\_FOR\_UNLOCK based on the rules described inside the function.

#### 5.102.2.16 AK\_lock\_released()

```
void AK_lock_released ( )
```

Function which is called when lock is released.

#### Author

Ivan Pusic

#### 5.102.2.17 AK\_memory\_block\_hash()

```
int AK_memory_block_hash (
    int blockMemoryAddress )
```

Calculates hash value for a given memory address. Hash values are used to identify location of locked resources.

#### Author

Frane Jakelić

**Todo** The current implementation is very limited it doesn't cope well with collision. recommendation use some better version of hash calculation. Maybe Knuth's memory address hashing function.

**Parameters**

|                           |   |
|---------------------------|---|
| <i>blockMemoryAddress</i> | integer representation of memory address, the hash value is calculated from this parameter. |
|---------------------------|---|

**Returns**

integer containing the hash value of the passed memory address

**5.102.2.18 AK\_on\_all\_transactions\_end()**

```
void AK_on_all_transactions_end ( )
```

Function for handling event when all transactions are finished.

**Author**

Ivan Pusic

**5.102.2.19 AK\_on\_lock\_release()**

```
void AK_on_lock_release ( )
```

Function for handling event when one of lock is released.

**Author**

Ivan Pusic

**5.102.2.20 AK\_on\_observable\_notify()**

```
void AK_on_observable_notify (
    void * observer,
    void * observable,
    AK_ObservableType_Enum type )
```

Function for handling notify from some observable type.

**Author**

Ivan Pusic

## Parameters

|                   |   |
|-------------------|---|
| <i>observer</i>   | <a href="#">Observer</a> type           |
| <i>observable</i> | <a href="#">Observable</a> type         |
| <i>type</i>       | Type of observable who sent some notice |

## 5.102.2.21 AK\_on\_transaction\_end()

```
void AK_on_transaction_end (
    pthread_t transaction_thread )
```

Function for handling event when some transaction is finished.

## Author

Ivan Pusic

## Parameters

|                           |  |
|---------------------------|--|
| <i>transaction_thread</i> | Thread ID of transaction which is finished |
|---------------------------|--|

## 5.102.2.22 AK\_release\_locks()

```
void AK_release_locks (
    AK\_memoryAddresses\_link addressesTmp,
    pthread_t transactionId )
```

Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .

## Author

Frane Jakelić updated by Ivan Pusic

## Parameters

|                            |  |
|----------------------------|--|
| <i>adresses</i>            | linked list of memory addresses locked by the transaction. |
| <i>transaction↔<br/>Id</i> | integer representation of transaction id.                  |



#### 5.102.2.23 AK\_remove\_transaction\_thread()

```
int AK_remove_transaction_thread (
    pthread_t transaction_thread )
```

Function for deleting one of active threads from array of all active transactions threads.

##### Author

Ivan Pusic

##### Parameters

|                           |                         |
|---------------------------|-------------------------|
| <i>transaction_thread</i> | Active thread to delete |
|---------------------------|-------------------------|

##### Returns

Exit status (OK or NOT\_OK)

#### 5.102.2.24 AK\_search\_empty\_link\_for\_hook()

```
AK_transaction_elem_P AK_search_empty_link_for_hook (
    int blockAddress )
```

Searches for a empty link for new active block, helper method in case of address collision.

##### Author

Frane Jakelić

##### Parameters

|                     |   |
|---------------------|---|
| <i>blockAddress</i> | integer representation of memory address. |
|---------------------|---|

##### Returns

pointer to empty location to store new active address

#### 5.102.2.25 AK\_search\_existing\_link\_for\_hook()

```
AK_transaction_elem_P AK_search_existing_link_for_hook (
    int blockAddress )
```

Searches for a existing entry in hash list of active blocks.

##### Author

Frane Jakelić

**Parameters**

|                     |   |
|---------------------|---|
| <i>blockAddress</i> | integer representation of memory address. |
|---------------------|---|

**Returns**

pointer to the existing hash list entry

**5.102.2.26 AK\_search\_lock\_entry\_list\_by\_key()**

```
AK_transaction_lock_elem_P AK_search_lock_entry_list_by_key (
    AK_transaction_elem_P Lockslist,
    int memoryAddress,
    pthread_t id )
```

Searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.

**Author**

Frane Jakelić

**Parameters**

|                      |   |
|----------------------|---|
| <i>memoryAddress</i> | integer representation of memory address. |
| <i>id</i>            | integer representation of transaction id. |

**Returns**

NULL pointer if the element is not found otherwise it returns a pointer to the found element

**5.102.2.27 AK\_transaction\_finished()**

```
void AK_transaction_finished ( )
```

Function which is called when some transaction is finished.

**Author**

Ivan Pusic

#### 5.102.2.28 AK\_transaction\_manager()

```
void AK_transaction_manager (
    command * commandArray,
    int lengthOfArray )
```

method that receives all the data and gives an id to that data and starts a thread that executes the transaction

##### Author

Frane Jakelić updated by Ivan Pusic

##### Parameters

|                      |   |
|----------------------|---|
| <i>commandArray</i>  | array filled with commands that need to be secured using transactions |
| <i>lengthOfArray</i> | length of commandArray  |

#### 5.102.2.29 AK\_transaction\_register\_observer()

```
int AK_transaction_register_observer (
    AK_observable_transaction * observable_transaction,
    AK_observer * observer )
```

Function for registering new observer of AK\_observable\_transaction type.

##### Author

Ivan Pusic

##### Parameters

|                               |                          |
|-------------------------------|--------------------------|
| <i>observable_transaction</i> | Observable type instance |
| <i>observer</i>               | Observer instance        |

##### Returns

Exit status (OK or NOT\_OK)

#### 5.102.2.30 AK\_transaction\_unregister\_observer()

```
int AK_transaction_unregister_observer (
    AK_observable_transaction * observable_transaction,
    AK_observer * observer )
```

Function for unregistering observer from AK\_observable\_transaction type.

**Author**

Ivan Pusic

**Parameters**

|   |  |
|---|--|
| <a href="#"><i>observable_transaction</i></a> | <a href="#">Observable</a> type instance |
| <i>observer</i>                               | <a href="#">Observer</a> instance        |

**Returns**

Exit status (OK or NOT\_OK)

**5.102.2.31 handle\_transaction\_notify()**

```
void handle_transaction_notify (
    AK\_observer\_lock * observer_lock )
```

Function for handling AK\_observable\_transaction notify. Function is associated to some observer instance.

**Author**

Ivan Pusic

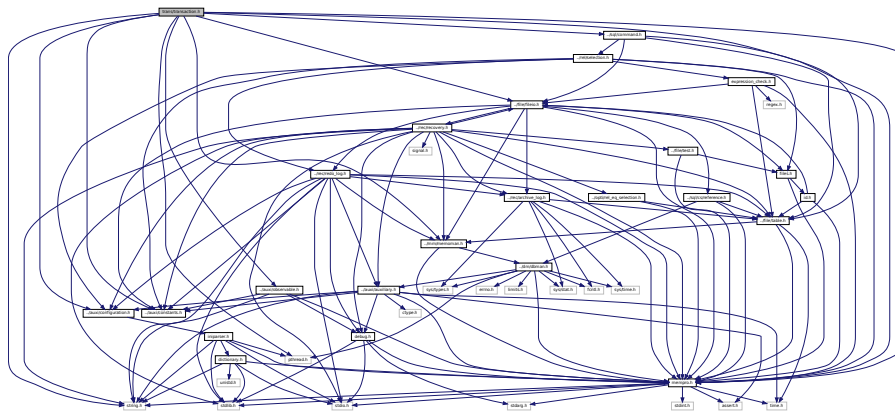
**Parameters**

|                               |  |
|-------------------------------|--|
| <a href="#">observer_lock</a> | <a href="#">Observer</a> type instance |
|-------------------------------|--|

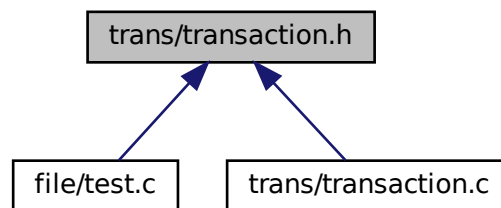
**5.103 trans/transaction.h File Reference**

```
#include <pthread.h>
#include "../auxi/constants.h"
#include "../auxi/configuration.h"
#include "../mm/memoman.h"
#include "../sql/command.h"
#include "../auxi/observable.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include <string.h>
#include "../auxi/mempro.h"
```

Include dependency graph for transaction.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [observable\\_transaction\\_struct](#)
- struct [observer\\_lock](#)  
Structure which defines transaction lock observer type.
- struct [transaction\\_locks\\_list\\_elem](#)  
Structure that represents LockTable entry about transaction resource lock.
- struct [transaction\\_list\\_elem](#)  
Structure that represents LockTable entry about transaction lock holder.Element indexed by Hash table.
- struct [transaction\\_list\\_head](#)  
Structure that represents LockTable entry about doubly linked list of collision in Hash table.
- struct [memoryAddresses](#)  
Structure that represents a linked list of locked addresses.
- struct [transactionData](#)  
Structure used to transport transaction data to the thread.
- struct [threadContainer](#)  
Structure that represents a linked list of threads.

## Typedefs

- typedef struct [observable\\_transaction\\_struct](#) **AK\_observable\_transaction**
- typedef struct [observer\\_lock](#) **AK\_observer\_lock**
- typedef struct [transactionData](#) **AK\_transaction\_data**
- typedef struct [memoryAddresses](#) **AK\_memoryAddresses**
- typedef struct [memoryAddresses](#) \* **AK\_memoryAddresses\_link**
- typedef struct [transaction\\_list\\_head](#) **AK\_transaction\_list**
- typedef struct [transaction\\_list\\_elem](#) \* **AK\_transaction\_elem\_P**
- typedef struct [transaction\\_list\\_elem](#) **AK\_transaction\_elem**
- typedef struct [transaction\\_locks\\_list\\_elem](#) \* **AK\_transaction\_lock\_elem\_P**
- typedef struct [transaction\\_locks\\_list\\_elem](#) **AK\_transaction\_lock\_elem**
- typedef struct [threadContainer](#) \* **AK\_thread\_elem**
- typedef struct [threadContainer](#) **AK\_thread\_Container**

## Enumerations

- enum [NoticeType](#) { **AK\_LOCK\_RELEASED**, **AK\_TRANSACTION\_FINISHED**, **AK\_ALL\_TRANSACTION\_↵**  
**\_FINISHED** }

*Enumeration which define notice types for transactions.*

## Functions

- int [AK\\_memory\\_block\\_hash](#) (int)  
*Calculates hash value for a given memory address. Hash values are used to identify location of locked resources.*
- [AK\\_transaction\\_elem\\_P AK\\_search\\_existing\\_link\\_for\\_hook](#) (int)  
*Searches for a existing entry in hash list of active blocks.*
- [AK\\_transaction\\_elem\\_P AK\\_search\\_empty\\_link\\_for\\_hook](#) (int)  
*Searches for a empty link for new active block, helper method in case of address collision.*
- [AK\\_transaction\\_elem\\_P AK\\_add\\_hash\\_entry\\_list](#) (int, int)  
*Adds an element to the doubly linked list.*
- int [AK\\_delete\\_hash\\_entry\\_list](#) (int)  
*Deletes a specific element in the lockTable doubly linked list.*
- [AK\\_transaction\\_lock\\_elem\\_P AK\\_search\\_lock\\_entry\\_list\\_by\\_key](#) ([AK\\_transaction\\_elem\\_P](#), int, pthread\_t)  
*Searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.*
- int [AK\\_delete\\_lock\\_entry\\_list](#) (int, pthread\_t)  
*Deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.*
- int [AK\\_isLock\\_waiting](#) ([AK\\_transaction\\_elem\\_P](#), int, pthread\_t, [AK\\_transaction\\_lock\\_elem\\_P](#))  
*Based on the parameters puts an transaction action in waiting phase or let's the transaction do it's actions.*
- [AK\\_transaction\\_lock\\_elem\\_P AK\\_add\\_lock](#) ([AK\\_transaction\\_elem\\_P](#), int, pthread\_t)  
*Adds an element to the locks doubly linked list.*
- [AK\\_transaction\\_lock\\_elem\\_P AK\\_create\\_lock](#) (int, int, pthread\_t)  
*Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.*
- int [AK\\_acquire\\_lock](#) (int, int, pthread\_t)  
*Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.*
- void [AK\\_release\\_locks](#) ([AK\\_memoryAddresses\\_link](#), pthread\_t)  
*Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .*
- int [AK\\_get\\_memory\\_blocks](#) (char \*, [AK\\_memoryAddresses\\_link](#))  
*Method that appends all addresses affected by the transaction.*

- int [AK\\_execute\\_commands](#) ([command](#) \*, int)
 

*Method that is called in a separate thread that is responsible for acquiring locks releasing them and finding the associated block addresses.*
- void \* [AK\\_execute\\_transaction](#) (void \*)
 

*thread start point all relevant functions are called from this function. It acts as an intermediary between the main thread and other threads*
- void [AK\\_transaction\\_manager](#) ([command](#) \*, int)
 

*method that receives all the data and gives an id to that data and starts a thread that executes the transaction*
- void [AK\\_test\\_Transaction](#) ()
- int [AK\\_create\\_new\\_transaction\\_thread](#) ([AK\\_transaction\\_data](#) \*)
 

*Function for creating new thread. Function also adds thread ID to pthread\_t array.*
- int [AK\\_remove\\_transaction\\_thread](#) (pthread\_t)
 

*Function for deleting one of active threads from array of all active transactions threads.*
- void [handle\\_transaction\\_notify](#) ([AK\\_observer\\_lock](#) \*)
 

*Function for handling AK\_observable\_transaction notify. Function is associated to some observer instance.*
- void [AK\\_on\\_observable\\_notify](#) (void \*, void \*, AK\_ObservableType\_Enum)
 

*Function for handling notify from some observable type.*
- void [AK\\_on\\_transaction\\_end](#) (pthread\_t)
 

*Function for handling event when some transaction is finished.*
- void [AK\\_on\\_lock\\_release](#) ()
 

*Function for handling event when one of lock is released.*
- void [AK\\_on\\_all\\_transactions\\_end](#) ()
 

*Function for handling event when all transactions are finished.*
- void [AK\\_handle\\_observable\\_transaction\\_action](#) ([NoticeType](#) \*)
 

*Function for handling action which is called from [observable\\_transaction](#) type.*
- void [AK\\_lock\\_released](#) ()
 

*Function which is called when lock is released.*
- void [AK\\_transaction\\_finished](#) ()
 

*Function which is called when some transaction is finished.*
- void [AK\\_all\\_transactions\\_finished](#) ()
 

*Function which is called when all transactions are finished.*
- int [AK\\_transaction\\_register\\_observer](#) ([AK\\_observable\\_transaction](#) \*, [AK\\_observer](#) \*)
 

*Function for registering new observer of AK\_observable\_transaction type.*
- int [AK\\_transaction\\_unregister\\_observer](#) ([AK\\_observable\\_transaction](#) \*, [AK\\_observer](#) \*)
 

*Function for unregistering observer from AK\_observable\_transaction type.*
- [AK\\_observable\\_transaction](#) \* [AK\\_init\\_observable\\_transaction](#) ()
 

*Function for initialization of AK\_observable\_transaction type.*
- [AK\\_observer\\_lock](#) \* [AK\\_init\\_observer\\_lock](#) ()
 

*Function for initialization of AK\_observer\_lock type.*

### 5.103.1 Detailed Description

Header file that defines includes and datastructures for the transaction execution

### 5.103.2 Enumeration Type Documentation

### 5.103.2.1 NoticeType

enum `NoticeType`

Enumeration which define notice types for transactions.

#### Author

Ivan Pusic

## 5.103.3 Function Documentation

### 5.103.3.1 AK\_acquire\_lock()

```
int AK_acquire_lock (
    int memoryAddress,
    int type,
    pthread_t transactionId )
```

Main interface function for the transaction API. It is responsible for the whole process of creating a new lock.

#### Author

Frane Jakelić updated by Ivan Pusic

**Todo** Implement a better deadlock detection. This method uses a very simple approach. It waits for 60sec before it restarts a transaction.

#### Parameters

|                      |  |
|----------------------|--|
| <i>memoryAddress</i> | integer representation of memory address.      |
| <i>type</i>          | of lock issued to the provided memory address. |
| <i>transactionId</i> | integer representation of transaction id.      |

#### Returns

OK or NOT\_OK based on the success of the function.

```
##### Lock Granted after wait##-----## Lock ID:lu TYPE:i ##-----## LockedAddress:i #####",
(unsigned long)lock->TransactionId, lock->lock_type, memoryAddress); */
```

```
##### Lock Granted ##-----## Lock ID:lu TYPE:i ##-----## LockedAddress:
:i #####", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress); */
```



## 5.103.3.2 AK\_add\_hash\_entry\_list()

```
AK_transaction_elem_P AK_add_hash_entry_list (
    int blockAddress,
    int type )
```

Adds an element to the doubly linked list.

## Author

Frane Jakelić

## Parameters

|                     |  |
|---------------------|--|
| <i>blockAddress</i> | integer representation of memory address.      |
| <i>type</i>         | of lock issued to the provided memory address. |

## Returns

pointer to the newly created doubly linked element.

## 5.103.3.3 AK\_add\_lock()

```
AK_transaction_lock_elem_P AK_add_lock (
    AK_transaction_elem_P HashList,
    int type,
    pthread_t transactionId )
```

Adds an element to the locks doubly linked list.

## Author

Frane Jakelić

## Parameters

|                      |  |
|----------------------|--|
| <i>memoryAddress</i> | integer representation of memory address.      |
| <i>type</i>          | of lock issued to the provided memory address. |
| <i>transactionId</i> | integer representation of transaction id.      |

## Returns

pointer to the newly created Locks doubly linked element.

#### 5.103.3.4 AK\_all\_transactions\_finished()

```
void AK_all_transactions_finished ( )
```

Function which is called when all transactions are finished.

##### Author

Ivan Pusic

#### 5.103.3.5 AK\_create\_lock()

```
AK_transaction_lock_elem_P AK_create_lock (
    int blockAddress,
    int type,
    pthread_t transactionId )
```

Helper function that determines if there is a hash LockTable entry that corresponds to the given memory address. And if there isn't an entry the function calls for the creation of the Locks list holder.

##### Author

Frane Jakelić

##### Parameters

|                      |  |
|----------------------|--|
| <i>memoryAddress</i> | integer representation of memory address.      |
| <i>type</i>          | of lock issued to the provided memory address. |
| <i>transactionId</i> | integer representation of transaction id.      |

##### Returns

pointer to the newly created Locks doubly linked element.

#### 5.103.3.6 AK\_create\_new\_transaction\_thread()

```
int AK_create_new_transaction_thread (
    AK_transaction_data * transaction_data )
```

Function for creating new thread. Function also adds thread ID to pthread\_t array.

##### Author

Ivan Pusic

**Parameters**

|                         |                                |
|-------------------------|--------------------------------|
| <i>transaction_data</i> | Data for executing transaction |
|-------------------------|--------------------------------|

**Returns**

Exit status (OK or NOT\_OK)

**5.103.3.7 AK\_delete\_hash\_entry\_list()**

```
int AK_delete_hash_entry_list (  
    int blockAddress )
```

Deletes a specific element in the lockTable doubly linked list.

**Author**

Frane Jakelić

**Parameters**

|                     |   |
|---------------------|---|
| <i>blockAddress</i> | integer representation of memory address. |
|---------------------|---|

**Returns**

integer OK or NOT\_OK based on success of finding the specific element in the list.

**5.103.3.8 AK\_delete\_lock\_entry\_list()**

```
int AK_delete_lock_entry_list (  
    int blockAddress,  
    pthread_t id )
```

Deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.

**Author**

Frane Jakelić

**Parameters**

|                     |   |
|---------------------|---|
| <i>blockAddress</i> | integer representation of memory address. |
| <i>id</i>           | integer representation of transaction id. |

**Returns**

int OK or NOT\_OK based on success of finding the specific element in the list.

**5.103.3.9 AK\_execute\_commands()**

```
int AK_execute_commands (
    command * commandArray,
    int lengthOfArray )
```

Method that is called in a separate thread that is responsible for acquiring locks releasing them and finding the associated block addresses.

**Author**

Frane Jakelić updated by Ivan Pusic

**Todo** Check multithreading, check if it's working correctly

**Parameters**

|                      |   |
|----------------------|---|
| <i>commandArray</i>  | array filled with commands that need to be secured using transactions |
| <i>lengthOfArray</i> | length of commandArray  |
| <i>transactionId</i> | associated with the transaction                                       |

**Returns**

ABORT or COMMIT based on the success of the function.

**5.103.3.10 AK\_execute\_transaction()**

```
void* AK_execute_transaction (
    void * params )
```

thread start point all relevant functions are called from this function. It acts as an intermediary between the main thread and other threads

**Author**

Frane Jakelić updated by Ivan Pusic

**Parameters**

|             |  |
|-------------|--|
| <i>data</i> | transmitted to the thread from the main thread |
|-------------|--|

#### 5.103.3.11 AK\_get\_memory\_blocks()

```
int AK_get_memory_blocks (
    char * tblName,
    AK_memoryAddresses_link addressList )
```

Method that appends all addresses affected by the transaction.

##### Author

Frane Jakelić

##### Parameters

|                    |  |
|--------------------|--|
| <i>addressList</i> | pointer to the linked list where the addresses are stored. |
| <i>tblName</i>     | table name used in the transaction                         |

##### Returns

OK or NOT\_OK based on the success of the function.

#### 5.103.3.12 AK\_handle\_observable\_transaction\_action()

```
void AK_handle_observable_transaction_action (
    NoticeType * noticeType )
```

Function for handling action which is called from [observable\\_transaction](#) type.

##### Author

Ivan Pusic

##### Parameters

|                   |                        |
|-------------------|------------------------|
| <i>noticeType</i> | Type of action (event) |
|-------------------|------------------------|

#### 5.103.3.13 AK\_init\_observable\_transaction()

```
AK_observable_transaction* AK_init_observable_transaction ( )
```

Function for initialization of [AK\\_observable\\_transaction](#) type.

**Author**

Ivan Pusic

**Returns**

Pointer to new AK\_observable\_transaction instance

**5.103.3.14 AK\_init\_observer\_lock()**

```
AK_observer_lock* AK_init_observer_lock ( )
```

Function for initialization of AK\_observer\_lock type.

**Author**

Ivan Pusic

**Returns**

Pointer to new AK\_observer\_lock instance

**5.103.3.15 AK\_isLock\_waiting()**

```
int AK_isLock_waiting (
    AK_transaction_elem_P lockHolder,
    int type,
    pthread_t transactionId,
    AK_transaction_lock_elem_P lock )
```

Based on the parameters puts an transaction action in waiting phase or let's the transaction do it's actions.

**Author**

Frane Jakelić updated by Ivan Pusic

**Parameters**

|                      |   |
|----------------------|---|
| <i>lockHolder</i>    | pointer to the hash list entry that is entitled to the specific memory address. |
| <i>type</i>          | of lock issued to the provided memory address.                                  |
| <i>transactionId</i> | integer representation of transaction id.                                       |
| <i>lock</i>          | pointer to the lock element that is being tested.                               |

**Returns**

int PASS\_LOCK\_QUEUE or WAIT\_FOR\_UNLOCK based on the rules described inside the function.

**5.103.3.16 AK\_lock\_released()**

```
void AK_lock_released ( )
```

Function which is called when lock is released.

**Author**

Ivan Pusic

**5.103.3.17 AK\_memory\_block\_hash()**

```
int AK_memory_block_hash (
    int blockMemoryAddress )
```

Calculates hash value for a given memory address. Hash values are used to identify location of locked resources.

**Author**

Frane Jakelić

**Todo** The current implementation is very limited it doesn't cope well with collision. recommendation use some better version of hash calculation. Maybe Knuth's memory address hashing function.

**Parameters**

|                           |   |
|---------------------------|---|
| <i>blockMemoryAddress</i> | integer representation of memory address, the hash value is calculated from this parameter. |
|---------------------------|---|

**Returns**

integer containing the hash value of the passed memory address

**5.103.3.18 AK\_on\_all\_transactions\_end()**

```
void AK_on_all_transactions_end ( )
```

Function for handling event when all transactions are finished.

**Author**

Ivan Pusic

**5.103.3.19 AK\_on\_lock\_release()**

```
void AK_on_lock_release ( )
```

Function for handling event when one of lock is released.

**Author**

Ivan Pusic

**5.103.3.20 AK\_on\_observable\_notify()**

```
void AK_on_observable_notify (
    void * observer,
    void * observable,
    AK_ObservableType_Enum type )
```

Function for handling notify from some observable type.

**Author**

Ivan Pusic

**Parameters**

|                   |   |
|-------------------|---|
| <i>observer</i>   | <a href="#">Observer</a> type           |
| <i>observable</i> | <a href="#">Observable</a> type         |
| <i>type</i>       | Type of observable who sent some notice |

**5.103.3.21 AK\_on\_transaction\_end()**

```
void AK_on_transaction_end (
    pthread_t transaction_thread )
```

Function for handling event when some transaction is finished.

**Author**

Ivan Pusic



## Parameters

|                           |  |
|---------------------------|--|
| <i>transaction_thread</i> | Thread ID of transaction which is finished |
|---------------------------|--|

## 5.103.3.22 AK\_release\_locks()

```
void AK_release_locks (
    AK_memoryAddresses_link addressesTmp,
    pthread_t transactionId )
```

Main interface function for the transaction API. It is responsible for the whole process releasing locks acquired by a transaction. The locks are released either by COMMIT or ABORT .

## Author

Frane Jakelić updated by Ivan Pusic

## Parameters

|                            |  |
|----------------------------|--|
| <i>adresses</i>            | linked list of memory addresses locked by the transaction. |
| <i>transaction↔<br/>Id</i> | integer representation of transaction id.                  |

## 5.103.3.23 AK\_remove\_transaction\_thread()

```
int AK_remove_transaction_thread (
    pthread_t transaction_thread )
```

Function for deleting one of active threads from array of all active transactions threads.

## Author

Ivan Pusic

## Parameters

|                           |                         |
|---------------------------|-------------------------|
| <i>transaction_thread</i> | Active thread to delete |
|---------------------------|-------------------------|

## Returns

Exit status (OK or NOT\_OK)

#### 5.103.3.24 AK\_search\_empty\_link\_for\_hook()

```
AK_transaction_elem_P AK_search_empty_link_for_hook (
    int blockAddress )
```

Searches for a empty link for new active block, helper method in case of address collision.

##### Author

Frane Jakelić

##### Parameters

|                     |   |
|---------------------|---|
| <i>blockAddress</i> | integer representation of memory address. |
|---------------------|---|

##### Returns

pointer to empty location to store new active address

#### 5.103.3.25 AK\_search\_existing\_link\_for\_hook()

```
AK_transaction_elem_P AK_search_existing_link_for_hook (
    int blockAddress )
```

Searches for a existing entry in hash list of active blocks.

##### Author

Frane Jakelić

##### Parameters

|                     |   |
|---------------------|---|
| <i>blockAddress</i> | integer representation of memory address. |
|---------------------|---|

##### Returns

pointer to the existing hash list entry

#### 5.103.3.26 AK\_search\_lock\_entry\_list\_by\_key()

```
AK_transaction_lock_elem_P AK_search_lock_entry_list_by_key (
    AK_transaction_elem_P Lockslist,
    int memoryAddress,
    pthread_t id )
```

Searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.

**Author**

Frane Jakelić

**Parameters**

|                      |   |
|----------------------|---|
| <i>memoryAddress</i> | integer representation of memory address. |
| <i>id</i>            | integer representation of transaction id. |

**Returns**

NULL pointer if the element is not found otherwise it returns a pointer to the found element

**5.103.3.27 AK\_transaction\_finished()**

```
void AK_transaction_finished ( )
```

Function which is called when some transaction is finished.

**Author**

Ivan Pusic

**5.103.3.28 AK\_transaction\_manager()**

```
void AK_transaction_manager (
    command * commandArray,
    int lengthOfArray )
```

method that receives all the data and gives an id to that data and starts a thread that executes the transaction

**Author**

Frane Jakelić updated by Ivan Pusic

**Parameters**

|                      |   |
|----------------------|---|
| <i>commandArray</i>  | array filled with commands that need to be secured using transactions |
| <i>lengthOfArray</i> | length of <i>commandArray</i>   |

#### 5.103.3.29 AK\_transaction\_register\_observer()

```
int AK_transaction_register_observer (
    AK_observable_transaction * observable_transaction,
    AK_observer * observer )
```

Function for registering new observer of AK\_observable\_transaction type.

##### Author

Ivan Pusic

##### Parameters

|  |                          |
|--|--------------------------|
| <a href="#">observable_transaction</a> | Observable type instance |
| <a href="#">observer</a>               | Observer instance        |

##### Returns

Exit status (OK or NOT\_OK)

#### 5.103.3.30 AK\_transaction\_unregister\_observer()

```
int AK_transaction_unregister_observer (
    AK_observable_transaction * observable_transaction,
    AK_observer * observer )
```

Function for unregistering observer from AK\_observable\_transaction type.

##### Author

Ivan Pusic

##### Parameters

|  |                          |
|--|--------------------------|
| <a href="#">observable_transaction</a> | Observable type instance |
| <a href="#">observer</a>               | Observer instance        |

##### Returns

Exit status (OK or NOT\_OK)

#### 5.103.3.31 handle\_transaction\_notify()

```
void handle_transaction_notify (
    AK_observer_lock * observer_lock )
```

Function for handling AK\_observable\_transaction notify. Function is associated to some observer instance.

**Author**

Ivan Pusic

**Parameters**

|                      |                        |
|----------------------|------------------------|
| <i>observer_lock</i> | Observer type instance |
|----------------------|------------------------|



# Index

- [\\_dictionary\\_, 11](#)
    - [hash, 11](#)
    - [key, 11](#)
    - [size, 12](#)
    - [val, 12](#)
  - [\\_file\\_metadata, 12](#)
  - [\\_line\\_status\\_](#)
    - [iniparser.c, 110](#)
  - [\\_notifyDetails, 12](#)
- [AK\\_ALLOCATION\\_TABLE\\_SIZE](#)
  - [dbman.h, 183](#)
- [AK\\_GUID](#)
  - [blobs.c, 203](#)
  - [blobs.h, 208](#)
- [AK\\_acquire\\_lock](#)
  - [transaction.c, 553](#)
  - [transaction.h, 570](#)
- [AK\\_add\\_hash\\_entry\\_list](#)
  - [transaction.c, 554](#)
  - [transaction.h, 570](#)
- [AK\\_add\\_lock](#)
  - [transaction.c, 555](#)
  - [transaction.h, 571](#)
- [AK\\_add\\_reference](#)
  - [reference.c, 484](#)
  - [reference.h, 490](#)
- [AK\\_add\\_succesor](#)
  - [auxiliary.c, 57](#)
  - [auxiliary.h, 75](#)
- [AK\\_add\\_to\\_bitmap\\_index](#)
  - [bitmap.c, 240](#)
  - [bitmap.h, 247](#)
- [AK\\_add\\_to\\_redolog](#)
  - [redo\\_log.c, 395](#)
- [AK\\_add\\_user\\_to\\_group](#)
  - [privileges.c, 523](#)
- [AK\\_add\\_vertex](#)
  - [auxiliary.c, 57](#)
  - [auxiliary.h, 75](#)
- [AK\\_agg\\_input, 13](#)
- [AK\\_agg\\_input\\_add](#)
  - [aggregation.c, 397](#)
  - [aggregation.h, 403](#)
- [AK\\_agg\\_input\\_add\\_to\\_beginning](#)
  - [aggregation.c, 398](#)
  - [aggregation.h, 404](#)
- [AK\\_agg\\_input\\_fix](#)
  - [aggregation.c, 398](#)
  - [aggregation.h, 404](#)
- [AK\\_agg\\_input\\_init](#)
  - [aggregation.c, 399](#)
  - [aggregation.h, 405](#)
- [AK\\_agg\\_value, 13](#)
- [AK\\_aggregation](#)
  - [aggregation.c, 399](#)
  - [aggregation.h, 405](#)
- [AK\\_all\\_transactions\\_finished](#)
  - [transaction.c, 555](#)
  - [transaction.h, 571](#)
- [AK\\_allocate\\_block\\_activity\\_modes](#)
  - [dbman.c, 165](#)
- [AK\\_allocate\\_blocks](#)
  - [dbman.c, 165](#)
  - [dbman.h, 185](#)
- [AK\\_allocation\\_set\\_mode](#)
  - [dbman.h, 184](#)
- [AK\\_allocationbit](#)
  - [dbman.h, 200](#)
- [AK\\_allocationtable\\_dump](#)
  - [dbman.c, 165](#)
  - [dbman.h, 185](#)
- [AK\\_archive\\_log](#)
  - [archive\\_log.h, 390](#)
- [AK\\_block, 14](#)
- [AK\\_block\\_activity, 15](#)
- [AK\\_block\\_sort](#)
  - [filesort.h, 234](#)
- [AK\\_blocktable, 16](#)
- [AK\\_blocktable\\_dump](#)
  - [dbman.c, 166](#)
  - [dbman.h, 185](#)
- [AK\\_blocktable\\_flush](#)
  - [dbman.c, 166](#)
  - [dbman.h, 186](#)
- [AK\\_blocktable\\_get](#)
  - [dbman.c, 166](#)
  - [dbman.h, 186](#)
- [AK\\_btree\\_create](#)
  - [btree.c, 254](#)
  - [btree.h, 256](#)
- [AK\\_btree\\_search\\_delete](#)
  - [btree.c, 254](#)
  - [btree.h, 256](#)
- [AK\\_cache\\_AK\\_malloc](#)
  - [memoman.c, 333](#)
  - [memoman.h, 342](#)
- [AK\\_cache\\_block](#)
  - [memoman.c, 333](#)

- memoman.h, [342](#)
- AK\_cache\_result
  - memoman.c, [334](#)
  - memoman.h, [343](#)
- AK\_calloc
  - mempro.c, [130](#)
  - mempro.h, [145](#)
- AK\_change\_hash\_info
  - hash.c, [258](#)
  - hash.h, [265](#)
- AK\_chars\_num\_from\_number
  - auxiliary.c, [58](#)
  - auxiliary.h, [76](#)
- AK\_check\_arithmetic\_statement
  - expression\_check.c, [411](#)
  - expression\_check.h, [415](#)
- AK\_check\_attributes
  - redo\_log.c, [395](#)
- AK\_check\_constraint
  - check\_constraint.c, [468](#)
  - check\_constraint.h, [472](#)
- AK\_check\_constraint\_test
  - check\_constraint.c, [469](#)
  - check\_constraint.h, [472](#)
- AK\_check\_constraints
  - theta\_join.c, [452](#)
  - theta\_join.h, [455](#)
- AK\_check\_folder\_blobs
  - blobs.c, [202](#)
  - blobs.h, [207](#)
- AK\_check\_for\_writes
  - mempro.c, [130](#)
  - mempro.h, [145](#)
- AK\_check\_function\_arguments
  - function.c, [510](#)
  - function.h, [516](#)
- AK\_check\_function\_arguments\_type
  - function.c, [510](#)
  - function.h, [517](#)
- AK\_check\_group\_privilege
  - privileges.c, [523](#)
- AK\_check\_if\_row\_satisfies\_expression
  - expression\_check.c, [412](#)
  - expression\_check.h, [416](#)
- AK\_check\_privilege
  - privileges.c, [524](#)
- AK\_check\_tables\_scheme
  - table.c, [299](#)
  - table.h, [313](#)
- AK\_check\_user\_privilege
  - privileges.c, [524](#)
- AK\_command\_recovery\_struct, [16](#)
- AK\_command\_struct, [17](#)
- AK\_compare
  - rel\_eq\_assoc.c, [355](#)
  - rel\_eq\_assoc.h, [358](#)
- AK\_concat
  - blobs.c, [202](#)
- blobs.h, [207](#)
- AK\_constraint\_names\_test
  - constraint\_names.c, [475](#)
  - constraint\_names.h, [477](#)
- AK\_copy\_block\_projection
  - projection.c, [433](#)
  - projection.h, [440](#)
- AK\_copy\_blocks\_join
  - nat\_join.c, [422](#)
  - nat\_join.h, [425](#)
- AK\_copy\_header
  - dbman.c, [166](#)
  - dbman.h, [186](#)
- AK\_create\_Index\_Table
  - bitmap.c, [241](#)
  - bitmap.h, [249](#)
- AK\_create\_block\_header
  - projection.c, [433](#)
  - projection.h, [440](#)
- AK\_create\_hash\_index
  - hash.c, [258](#)
  - hash.h, [266](#)
- AK\_create\_header
  - dbman.c, [167](#)
  - dbman.h, [187](#)
- AK\_create\_header\_name
  - projection.c, [434](#)
  - projection.h, [441](#)
- AK\_create\_join\_block\_header
  - nat\_join.c, [422](#)
  - nat\_join.h, [426](#)
- AK\_create\_lock
  - transaction.c, [555](#)
  - transaction.h, [572](#)
- AK\_create\_new\_transaction\_thread
  - transaction.c, [556](#)
  - transaction.h, [572](#)
- AK\_create\_table\_struct, [17](#)
- AK\_create\_test\_tables
  - test.c, [324](#)
  - test.h, [328](#)
- AK\_create\_theta\_join\_header
  - theta\_join.c, [452](#)
  - theta\_join.h, [455](#)
- AK\_db\_cache, [18](#)
- AK\_deallocate\_search\_result
  - filesearch.c, [228](#)
  - filesearch.h, [231](#)
- AK\_debmod\_calloc
  - mempro.c, [130](#)
  - mempro.h, [146](#)
- AK\_debmod\_d
  - mempro.c, [131](#)
  - mempro.h, [146](#)
- AK\_debmod\_die
  - mempro.c, [131](#)
  - mempro.h, [147](#)
- AK\_debmod\_dv



- mempro.c, [132](#)
- mempro.h, [147](#)
- AK\_debmod\_enter\_critical\_sec
  - mempro.c, [132](#)
  - mempro.h, [148](#)
- AK\_debmod\_free
  - mempro.c, [132](#)
  - mempro.h, [148](#)
- AK\_debmod\_fstack\_pop
  - mempro.c, [133](#)
  - mempro.h, [148](#)
- AK\_debmod\_fstack\_push
  - mempro.c, [133](#)
  - mempro.h, [149](#)
- AK\_debmod\_func\_add
  - mempro.c, [134](#)
  - mempro.h, [149](#)
- AK\_debmod\_func\_get\_name
  - mempro.c, [134](#)
  - mempro.h, [150](#)
- AK\_debmod\_func\_id
  - mempro.c, [135](#)
  - mempro.h, [150](#)
- AK\_debmod\_function\_current
  - mempro.c, [135](#)
  - mempro.h, [151](#)
- AK\_debmod\_function\_epilogue
  - mempro.c, [136](#)
  - mempro.h, [151](#)
- AK\_debmod\_function\_prologue
  - mempro.c, [136](#)
  - mempro.h, [152](#)
- AK\_debmod\_init
  - mempro.c, [137](#)
  - mempro.h, [152](#)
- AK\_debmod\_leave\_critical\_sec
  - mempro.c, [137](#)
  - mempro.h, [152](#)
- AK\_debmod\_log\_memory\_alloc
  - mempro.c, [137](#)
  - mempro.h, [153](#)
- AK\_debmod\_print\_function\_use
  - mempro.c, [138](#)
  - mempro.h, [153](#)
- AK\_debmod\_state, [19](#)
- AK\_delete\_bitmap\_index
  - bitmap.c, [242](#)
  - bitmap.h, [249](#)
- AK\_delete\_block
  - dbman.c, [168](#)
  - dbman.h, [187](#)
- AK\_delete\_constraint\_between
  - between.c, [462](#)
- AK\_delete\_constraint\_not\_null
  - nnull.c, [478](#)
  - nnull.h, [481](#)
- AK\_delete\_constraint\_unique
  - unique.c, [498](#)
  - unique.h, [501](#)
- AK\_delete\_extent
  - dbman.c, [168](#)
  - dbman.h, [188](#)
- AK\_delete\_hash\_entry\_list
  - transaction.c, [556](#)
  - transaction.h, [573](#)
- AK\_delete\_in\_hash\_index
  - hash.c, [259](#)
  - hash.h, [266](#)
- AK\_delete\_lock\_entry\_list
  - transaction.c, [557](#)
  - transaction.h, [573](#)
- AK\_delete\_segment
  - dbman.c, [169](#)
  - dbman.h, [188](#)
- AK\_destroy\_critical\_section
  - auxiliary.c, [59](#)
  - auxiliary.h, [77](#)
- AK\_determine\_header\_type
  - projection.c, [434](#)
  - projection.h, [441](#)
- AK\_difference
  - difference.c, [407](#)
  - difference.h, [409](#)
- AK\_drop
  - drop.c, [504](#)
  - drop.h, [507](#)
- AK\_drop\_help\_function
  - drop.c, [505](#)
- AK\_drop\_test
  - drop.c, [505](#)
  - drop.h, [508](#)
- AK\_elem\_hash\_value
  - hash.c, [259](#)
  - hash.h, [267](#)
- AK\_empty\_archive\_log
  - archive\_log.h, [390](#)
- AK\_enter\_critical\_section
  - auxiliary.c, [60](#)
  - auxiliary.h, [79](#)
- AK\_execute\_commands
  - transaction.c, [557](#)
  - transaction.h, [574](#)
- AK\_execute\_rel\_eq
  - query\_optimization.c, [350](#)
  - query\_optimization.h, [353](#)
- AK\_execute\_transaction
  - transaction.c, [558](#)
  - transaction.h, [574](#)
- AK\_find\_AK\_free\_space
  - memoman.c, [334](#)
  - memoman.h, [343](#)
- AK\_find\_available\_result\_block
  - memoman.c, [335](#)
  - memoman.h, [344](#)
- AK\_find\_delete\_in\_hash\_index
  - hash.c, [260](#)

- hash.h, [267](#)
- AK\_find\_in\_hash\_index
  - hash.c, [260](#)
  - hash.h, [268](#)
- AK\_find\_table\_address
  - between.c, [462](#)
  - between.h, [466](#)
- AK\_flush\_cache
  - memoman.c, [335](#)
  - memoman.h, [344](#)
- AK\_folder\_exists
  - blobs.c, [202](#)
  - blobs.h, [207](#)
- AK\_fread
  - mempro.c, [138](#)
- AK\_free
  - mempro.c, [139](#)
  - mempro.h, [154](#)
- AK\_function\_add
  - function.c, [511](#)
  - function.h, [517](#)
- AK\_function\_arguments\_add
  - function.c, [511](#)
  - function.h, [518](#)
- AK\_function\_arguments\_remove\_by\_obj\_id
  - function.c, [512](#)
  - function.h, [518](#)
- AK\_function\_change\_return\_type
  - function.c, [512](#)
  - function.h, [519](#)
- AK\_function\_remove\_by\_name
  - function.c, [513](#)
  - function.h, [519](#)
- AK\_function\_remove\_by\_obj\_id
  - function.c, [513](#)
  - function.h, [520](#)
- AK\_function\_rename
  - function.c, [513](#)
  - function.h, [520](#)
- AK\_function\_test
  - function.c, [514](#)
  - function.h, [521](#)
- AK\_fwrite
  - mempro.c, [139](#)
- AK\_generate\_result\_id
  - memoman.c, [335](#)
  - memoman.h, [344](#)
- AK\_get\_Attribute
  - bitmap.c, [243](#)
  - bitmap.h, [250](#)
- AK\_get\_allocation\_set
  - dbman.c, [169](#)
  - dbman.h, [189](#)
- AK\_get\_array\_perms
  - auxiliary.c, [61](#)
  - auxiliary.h, [80](#)
- AK\_get\_attr\_index
  - table.c, [300](#)
- table.h, [313](#)
- AK\_get\_attr\_name
  - table.c, [300](#)
  - table.h, [314](#)
- AK\_get\_block
  - memoman.c, [335](#)
  - memoman.h, [344](#)
- AK\_get\_column
  - table.c, [301](#)
  - table.h, [314](#)
- AK\_get\_extent
  - dbman.c, [170](#)
  - dbman.h, [189](#)
- AK\_get\_function\_obj\_id
  - function.c, [514](#)
  - function.h, [521](#)
- AK\_get\_hash\_info
  - hash.c, [261](#)
  - hash.h, [268](#)
- AK\_get\_header
  - table.c, [301](#)
  - table.h, [315](#)
- AK\_get\_id
  - id.c, [236](#)
  - id.h, [238](#)
- AK\_get\_index\_addresses
  - memoman.c, [336](#)
  - memoman.h, [345](#)
- AK\_get\_index\_header
  - index.c, [273](#)
- AK\_get\_index\_num\_records
  - index.c, [274](#)
  - index.h, [283](#)
- AK\_get\_index\_segment\_addresses
  - memoman.c, [336](#)
  - memoman.h, [345](#)
- AK\_get\_index\_tuple
  - index.c, [274](#)
  - index.h, [283](#)
- AK\_get\_memory\_blocks
  - transaction.c, [558](#)
  - transaction.h, [575](#)
- AK\_get\_num\_records
  - table.c, [302](#)
  - table.h, [315](#)
- AK\_get\_operator
  - projection.c, [435](#)
  - projection.h, [443](#)
- AK\_get\_reference
  - reference.c, [484](#)
  - reference.h, [491](#)
- AK\_get\_rel\_exp
  - view.c, [546](#)
- AK\_get\_row
  - table.c, [303](#)
  - table.h, [316](#)
- AK\_get\_segment\_addresses
  - memoman.c, [337](#)

- memoman.h, 346
- AK\_get\_table\_addresses
  - memoman.c, 337
  - memoman.h, 346
- AK\_get\_table\_attribute\_types
  - test.c, 324
  - test.h, 328
- AK\_get\_table\_id
  - id.c, 236
- AK\_get\_table\_obj\_id
  - table.c, 303
  - table.h, 316
- AK\_get\_timestamp
  - archive\_log.h, 390
- AK\_get\_tuple
  - table.c, 303
  - table.h, 317
- AK\_get\_view\_obj\_id
  - view.c, 546
- AK\_get\_view\_query
  - view.c, 547
- AK\_grant\_privilege\_group
  - privileges.c, 525
- AK\_grant\_privilege\_user
  - privileges.c, 525
- AK\_group\_add
  - privileges.c, 526
- AK\_group\_get\_id
  - privileges.c, 526
- AK\_group\_remove\_by\_name
  - privileges.c, 527
- AK\_group\_rename
  - privileges.c, 527
- AK\_handle\_observable\_transaction\_action
  - transaction.c, 558
  - transaction.h, 575
- AK\_header, 19
- AK\_header\_size
  - aggregation.c, 400
  - aggregation.h, 406
- AK\_if\_exist
  - drop.c, 505
  - drop.h, 508
- AK\_increase\_extent
  - dbman.c, 170
  - dbman.h, 190
- AK\_index\_table\_exist
  - index.c, 277
  - index.h, 286
- AK\_index\_test
  - index.c, 277
  - index.h, 286
- AK\_init\_allocation\_table
  - dbman.c, 171
  - dbman.h, 191
- AK\_init\_block
  - dbman.c, 171
  - dbman.h, 191
- AK\_init\_critical\_section
  - auxiliary.c, 62
  - auxiliary.h, 81
- AK\_init\_db\_file
  - dbman.c, 171
  - dbman.h, 191
- AK\_init\_disk\_manager
  - dbman.c, 172
  - dbman.h, 192
- AK\_init\_new\_extent
  - memoman.c, 338
  - memoman.h, 347
- AK\_init\_observable
  - observable.c, 159
  - observable.h, 161
- AK\_init\_observable\_transaction
  - transaction.c, 559
  - transaction.h, 575
- AK\_init\_observer
  - observable.c, 159
  - observable.h, 161
- AK\_init\_observer\_lock
  - transaction.c, 559
  - transaction.h, 576
- AK\_init\_system\_catalog
  - dbman.c, 172
  - dbman.h, 192
- AK\_init\_system\_tables\_catalog
  - dbman.c, 172
  - dbman.h, 192
- AK\_initialize\_new\_index\_segment
  - files.c, 223
  - files.h, 226
- AK\_initialize\_new\_segment
  - files.c, 224
  - files.h, 226
  - reference.h, 491
- AK\_insert\_entry
  - dbman.c, 174
  - dbman.h, 194
- AK\_insert\_in\_hash\_index
  - hash.c, 263
  - hash.h, 270
- AK\_intersect
  - intersect.c, 418
  - intersect.h, 420
- AK\_isLock\_waiting
  - transaction.c, 559
  - transaction.h, 576
- AK\_join
  - nat\_join.c, 423
  - nat\_join.h, 427
- AK\_leave\_critical\_section
  - auxiliary.c, 66
  - auxiliary.h, 84
- AK\_lo\_export
  - blobs.c, 203
  - blobs.h, 208

- AK\_lo\_import
  - blobs.c, [203](#)
  - blobs.h, [208](#)
- AK\_lo\_test
  - blobs.c, [204](#)
  - blobs.h, [209](#)
- AK\_lo\_unlink
  - blobs.c, [204](#)
  - blobs.h, [209](#)
- AK\_lock\_released
  - transaction.c, [560](#)
  - transaction.h, [577](#)
- AK\_malloc
  - mempro.c, [140](#)
  - mempro.h, [154](#)
- AK\_mem\_block, [20](#)
- AK\_mem\_block\_modify
  - memoman.c, [338](#)
  - memoman.h, [347](#)
- AK\_memoman\_init
  - memoman.c, [338](#)
  - memoman.h, [347](#)
- AK\_memory\_block\_hash
  - transaction.c, [560](#)
  - transaction.h, [577](#)
- AK\_mempro\_test
  - mempro.c, [140](#)
  - mempro.h, [155](#)
- AK\_memset\_int
  - dbman.c, [174](#)
  - dbman.h, [194](#)
- AK\_merge\_block\_join
  - nat\_join.c, [423](#)
  - nat\_join.h, [427](#)
- AK\_mkdir
  - blobs.c, [204](#)
  - blobs.h, [209](#)
- AK\_new\_extent
  - dbman.c, [175](#)
  - dbman.h, [195](#)
- AK\_new\_segment
  - dbman.c, [176](#)
  - dbman.h, [196](#)
- AK\_null\_test
  - nnull.c, [478](#)
  - nnull.h, [481](#)
- AK\_num\_attr
  - table.c, [304](#)
  - table.h, [317](#)
- AK\_num\_index\_attr
  - index.c, [278](#)
  - index.h, [287](#)
- AK\_observable\_test
  - observable.c, [159](#)
  - observable.h, [162](#)
- AK\_on\_all\_transactions\_end
  - transaction.c, [561](#)
  - transaction.h, [577](#)
- AK\_on\_lock\_release
  - transaction.c, [561](#)
  - transaction.h, [578](#)
- AK\_on\_observable\_notify
  - transaction.c, [561](#)
  - transaction.h, [578](#)
- AK\_on\_transaction\_end
  - transaction.c, [562](#)
  - transaction.h, [578](#)
- AK\_op\_join\_test
  - nat\_join.c, [424](#)
  - nat\_join.h, [428](#)
- AK\_op\_product\_test
  - product.c, [429](#)
  - product.h, [431](#)
- AK\_op\_projection\_test
  - projection.c, [435](#)
  - projection.h, [443](#)
- AK\_op\_rename\_test
  - table.c, [304](#)
  - table.h, [318](#)
- AK\_op\_selection\_test
  - selection.c, [447](#)
  - selection.h, [450](#)
- AK\_op\_selection\_test\_pattern
  - selection.c, [447](#)
  - selection.h, [450](#)
- AK\_op\_selection\_test\_redolog
  - selection.c, [448](#)
  - selection.h, [450](#)
- AK\_op\_theta\_join\_test
  - theta\_join.c, [453](#)
  - theta\_join.h, [456](#)
- AK\_op\_union\_test
  - union.c, [457](#)
  - union.h, [459](#)
- AK\_operand, [21](#)
- AK\_perform\_operatrion
  - projection.c, [435](#)
  - projection.h, [443](#)
- AK\_pop\_from\_stack
  - auxiliary.c, [66](#)
  - auxiliary.h, [85](#)
- AK\_print\_active\_functions
  - mempro.c, [140](#)
  - mempro.h, [155](#)
- AK\_print\_block
  - dbman.c, [176](#)
  - dbman.h, [196](#)
- AK\_print\_constraints
  - between.c, [463](#)
- AK\_print\_function\_use
  - mempro.c, [140](#)
  - mempro.h, [155](#)
- AK\_print\_function\_uses
  - mempro.c, [141](#)
  - mempro.h, [156](#)
- AK\_print\_index\_table

- index.c, [279](#)
- index.h, [288](#)
- AK\_print\_optimized\_query
  - query\_optimization.c, [350](#)
  - query\_optimization.h, [353](#)
- AK\_print\_rel\_eq\_assoc
  - rel\_eq\_assoc.c, [356](#)
  - rel\_eq\_assoc.h, [359](#)
- AK\_print\_rel\_eq\_comut
  - rel\_eq\_comut.c, [361](#)
  - rel\_eq\_comut.h, [364](#)
- AK\_print\_rel\_eq\_projection
  - rel\_eq\_projection.c, [367](#)
  - rel\_eq\_projection.h, [373](#)
- AK\_print\_rel\_eq\_selection
  - rel\_eq\_selection.c, [378](#)
  - rel\_eq\_selection.h, [384](#)
- AK\_print\_row
  - table.c, [305](#)
  - table.h, [318](#)
- AK\_print\_row Spacer
  - table.c, [305](#)
  - table.h, [318](#)
- AK\_print\_row Spacer\_to\_file
  - table.c, [306](#)
  - table.h, [319](#)
- AK\_print\_row\_to\_file
  - table.c, [306](#)
  - table.h, [319](#)
- AK\_print\_table
  - table.c, [307](#)
  - table.h, [320](#)
- AK\_print\_table\_to\_file
  - table.c, [307](#)
  - table.h, [320](#)
- AK\_printout\_redolog
  - redo\_log.c, [396](#)
- AK\_privileges\_test
  - privileges.c, [528](#)
- AK\_product
  - product.c, [429](#)
  - product.h, [431](#)
- AK\_projection
  - projection.c, [437](#)
  - projection.h, [445](#)
- AK\_push\_to\_stack
  - auxiliary.c, [67](#)
  - auxiliary.h, [86](#)
- AK\_query\_mem, [22](#)
- AK\_query\_mem\_AK\_malloc
  - memoman.c, [339](#)
  - memoman.h, [348](#)
- AK\_query\_mem\_dict, [23](#)
- AK\_query\_mem\_lib, [23](#)
- AK\_query\_mem\_result, [24](#)
- AK\_query\_optimization
  - query\_optimization.c, [351](#)
  - query\_optimization.h, [354](#)
- AK\_query\_optimization\_test
  - query\_optimization.c, [351](#)
  - query\_optimization.h, [354](#)
- AK\_read\_block
  - dbman.c, [176](#)
  - dbman.h, [196](#)
- AK\_read\_block\_for\_testing
  - dbman.c, [177](#)
  - dbman.h, [197](#)
- AK\_read\_constraint\_between
  - between.c, [463](#)
  - between.h, [466](#)
- AK\_read\_constraint\_not\_null
  - nnull.c, [478](#)
  - nnull.h, [481](#)
- AK\_read\_constraint\_unique
  - unique.c, [499](#)
  - unique.h, [502](#)
- AK\_realloc
  - mempro.c, [141](#)
  - mempro.h, [156](#)
- AK\_recover\_archive\_log
  - recovery.c, [392](#)
- AK\_recover\_operation
  - recovery.c, [392](#)
- AK\_recovery\_insert\_row
  - recovery.c, [393](#)
- AK\_recovery\_test
  - recovery.c, [393](#)
- AK\_recovery\_tokenize
  - recovery.c, [393](#)
- AK\_redo\_log, [25](#)
- AK\_redo\_log\_AK\_malloc
  - memoman.c, [339](#)
  - memoman.h, [348](#)
- AK\_ref\_item, [26](#)
- AK\_reference\_check\_attribute
  - reference.c, [485](#)
  - reference.h, [494](#)
- AK\_reference\_check\_entry
  - reference.c, [485](#)
  - reference.h, [494](#)
- AK\_reference\_check\_if\_update\_needed
  - reference.c, [486](#)
  - reference.h, [495](#)
- AK\_reference\_check\_restricion
  - reference.c, [486](#)
  - reference.h, [495](#)
- AK\_reference\_test
  - reference.c, [487](#)
  - reference.h, [496](#)
- AK\_reference\_update
  - reference.c, [487](#)
  - reference.h, [496](#)
- AK\_refresh\_cache
  - memoman.c, [339](#)
  - memoman.h, [348](#)
- AK\_register\_system\_tables

- dbman.c, [177](#)
- dbman.h, [197](#)
- AK\_rel\_eq\_assoc
  - rel\_eq\_assoc.c, [356](#)
  - rel\_eq\_assoc.h, [359](#)
- AK\_rel\_eq\_assoc\_test
  - rel\_eq\_assoc.c, [357](#)
  - rel\_eq\_assoc.h, [360](#)
- AK\_rel\_eq\_can\_commute
  - rel\_eq\_projection.c, [367](#)
  - rel\_eq\_projection.h, [373](#)
- AK\_rel\_eq\_collect\_cond\_attributes
  - rel\_eq\_projection.c, [368](#)
  - rel\_eq\_projection.h, [374](#)
- AK\_rel\_eq\_commute\_with\_theta\_join
  - rel\_eq\_comut.c, [361](#)
  - rel\_eq\_comut.h, [364](#)
- AK\_rel\_eq\_comut
  - rel\_eq\_comut.c, [362](#)
  - rel\_eq\_comut.h, [365](#)
- AK\_rel\_eq\_comut\_test
  - rel\_eq\_comut.c, [362](#)
  - rel\_eq\_comut.h, [365](#)
- AK\_rel\_eq\_cond\_attributes
  - rel\_eq\_selection.c, [379](#)
  - rel\_eq\_selection.h, [385](#)
- AK\_rel\_eq\_get\_attributes\_char
  - rel\_eq\_selection.c, [379](#)
  - rel\_eq\_selection.h, [385](#)
- AK\_rel\_eq\_get\_attributes
  - rel\_eq\_projection.c, [368](#)
  - rel\_eq\_projection.h, [374](#)
- AK\_rel\_eq\_is\_attr\_subset
  - rel\_eq\_selection.c, [380](#)
  - rel\_eq\_selection.h, [386](#)
- AK\_rel\_eq\_is\_subset
  - rel\_eq\_projection.c, [369](#)
  - rel\_eq\_projection.h, [375](#)
- AK\_rel\_eq\_projection
  - rel\_eq\_projection.c, [370](#)
  - rel\_eq\_projection.h, [376](#)
- AK\_rel\_eq\_projection\_attributes
  - rel\_eq\_projection.c, [370](#)
  - rel\_eq\_projection.h, [376](#)
- AK\_rel\_eq\_projection\_test
  - rel\_eq\_projection.c, [371](#)
  - rel\_eq\_projection.h, [377](#)
- AK\_rel\_eq\_remove\_duplicates
  - rel\_eq\_projection.c, [371](#)
  - rel\_eq\_projection.h, [377](#)
- AK\_rel\_eq\_selection
  - rel\_eq\_selection.c, [381](#)
  - rel\_eq\_selection.h, [386](#)
- AK\_rel\_eq\_selection\_test
  - rel\_eq\_selection.c, [381](#)
  - rel\_eq\_selection.h, [387](#)
- AK\_rel\_eq\_share\_attributes
  - rel\_eq\_selection.c, [381](#)
- rel\_eq\_selection.h, [387](#)
- AK\_rel\_eq\_split\_condition
  - rel\_eq\_selection.c, [382](#)
  - rel\_eq\_selection.h, [388](#)
- AK\_release\_locks
  - transaction.c, [562](#)
  - transaction.h, [579](#)
- AK\_remove\_all\_users\_from\_group
  - privileges.c, [528](#)
- AK\_remove\_transaction\_thread
  - transaction.c, [562](#)
  - transaction.h, [579](#)
- AK\_remove\_user\_from\_all\_groups
  - privileges.c, [528](#)
- AK\_rename
  - table.c, [308](#)
  - table.h, [321](#)
- AK\_replace\_wild\_card
  - expression\_check.c, [413](#)
- AK\_results, [27](#)
- AK\_revoke\_all\_privileges\_group
  - privileges.c, [529](#)
- AK\_revoke\_all\_privileges\_user
  - privileges.c, [529](#)
- AK\_revoke\_privilege\_group
  - privileges.c, [530](#)
- AK\_revoke\_privilege\_user
  - privileges.c, [530](#)
- AK\_search\_empty\_link
  - auxiliary.c, [68](#)
  - auxiliary.h, [87](#)
- AK\_search\_empty\_link\_for\_hook
  - transaction.c, [563](#)
  - transaction.h, [579](#)
- AK\_search\_empty\_stack\_link
  - auxiliary.c, [69](#)
  - auxiliary.h, [87](#)
- AK\_search\_existing\_link\_for\_hook
  - transaction.c, [563](#)
  - transaction.h, [580](#)
- AK\_search\_in\_stack
  - auxiliary.c, [69](#)
  - auxiliary.h, [88](#)
- AK\_search\_lock\_entry\_list\_by\_key
  - transaction.c, [564](#)
  - transaction.h, [580](#)
- AK\_search\_unsorted
  - aggregation.c, [401](#)
  - filesearch.c, [229](#)
  - filesearch.h, [232](#)
- AK\_search\_vertex
  - auxiliary.c, [69](#)
  - auxiliary.h, [88](#)
- AK\_select
  - select.c, [533](#)
- AK\_select\_test
  - select.c, [534](#)
- AK\_selection

- reference.h, [496](#)
- selection.c, [448](#)
- selection.h, [450](#)
- AK\_sequence\_add
  - sequence.c, [289](#)
  - sequence.h, [294](#)
- AK\_sequence\_current\_value
  - sequence.c, [290](#)
  - sequence.h, [295](#)
- AK\_sequence\_get\_id
  - sequence.c, [290](#)
  - sequence.h, [295](#)
- AK\_sequence\_modify
  - sequence.c, [290](#)
  - sequence.h, [295](#)
- AK\_sequence\_next\_value
  - sequence.c, [291](#)
  - sequence.h, [296](#)
- AK\_sequence\_remove
  - sequence.c, [292](#)
  - sequence.h, [297](#)
- AK\_sequence\_rename
  - sequence.c, [292](#)
  - sequence.h, [297](#)
- AK\_sequence\_test
  - sequence.c, [292](#)
  - sequence.h, [297](#)
- AK\_set\_check\_constraint
  - check\_constraint.c, [469](#)
  - check\_constraint.h, [472](#)
- AK\_set\_constraint\_between
  - between.c, [464](#)
  - between.h, [467](#)
- AK\_set\_constraint\_not\_null
  - nnull.c, [479](#)
  - nnull.h, [482](#)
- AK\_split\_path\_file
  - blobs.c, [205](#)
  - blobs.h, [210](#)
- AK\_strcmp
  - auxiliary.c, [70](#)
  - auxiliary.h, [89](#)
- AK\_synchronization\_info, [28](#)
- AK\_table\_empty
  - table.c, [308](#)
  - table.h, [321](#)
- AK\_table\_exist
  - table.c, [309](#)
- AK\_table\_test
  - table.c, [309](#)
  - table.h, [322](#)
- AK\_tarjan
  - auxiliary.c, [71](#)
  - auxiliary.h, [90](#)
- AK\_temp\_create\_table
  - projection.c, [437](#)
  - projection.h, [445](#)
- AK\_theta\_join
  - theta\_join.c, [453](#)
  - theta\_join.h, [456](#)
- AK\_thread\_safe\_block\_access\_test
  - dbman.c, [178](#)
  - dbman.h, [198](#)
- AK\_transaction\_finished
  - transaction.c, [564](#)
  - transaction.h, [581](#)
- AK\_transaction\_manager
  - transaction.c, [564](#)
  - transaction.h, [581](#)
- AK\_transaction\_register\_observer
  - transaction.c, [565](#)
  - transaction.h, [581](#)
- AK\_transaction\_unregister\_observer
  - transaction.c, [565](#)
  - transaction.h, [582](#)
- AK\_trigger\_add
  - trigger.c, [535](#)
  - trigger.h, [541](#)
- AK\_trigger\_edit
  - trigger.c, [536](#)
  - trigger.h, [541](#)
- AK\_trigger\_get\_conditions
  - trigger.c, [536](#)
  - trigger.h, [542](#)
- AK\_trigger\_get\_id
  - trigger.c, [537](#)
  - trigger.h, [542](#)
- AK\_trigger\_remove\_by\_name
  - trigger.c, [537](#)
  - trigger.h, [543](#)
- AK\_trigger\_remove\_by\_obj\_id
  - trigger.c, [538](#)
  - trigger.h, [543](#)
- AK\_trigger\_rename
  - trigger.c, [538](#)
  - trigger.h, [544](#)
- AK\_trigger\_save\_conditions
  - trigger.c, [539](#)
  - trigger.h, [544](#)
- AK\_trigger\_test
  - trigger.c, [539](#)
  - trigger.h, [545](#)
- AK\_tuple\_dict, [28](#)
- AK\_tuple\_to\_string
  - table.c, [309](#)
  - table.h, [322](#)
- AK\_type\_size
  - auxiliary.c, [71](#)
  - auxiliary.h, [90](#)
- AK\_union
  - union.c, [458](#)
  - union.h, [460](#)
- AK\_unique\_test
  - unique.c, [500](#)
  - unique.h, [503](#)
- AK\_update

- bitmap.c, [244](#)
- bitmap.h, [252](#)
- AK\_user\_add
  - privileges.c, [531](#)
- AK\_user\_get\_id
  - privileges.c, [531](#)
- AK\_user\_remove\_by\_name
  - privileges.c, [532](#)
- AK\_user\_rename
  - privileges.c, [532](#)
- AK\_view\_add
  - view.c, [547](#)
- AK\_view\_change\_query
  - view.c, [548](#)
- AK\_view\_remove\_by\_name
  - view.c, [548](#)
- AK\_view\_remove\_by\_obj\_id
  - view.c, [549](#)
- AK\_view\_rename
  - view.c, [549](#)
- AK\_view\_test
  - view.c, [549](#)
- AK\_write\_block
  - dbman.c, [179](#)
  - dbman.h, [199](#)
- AK\_write\_block\_for\_testing
  - dbman.c, [179](#)
  - dbman.h, [199](#)
- AK\_write\_protect
  - mempro.c, [142](#)
  - mempro.h, [156](#)
- AK\_write\_unprotect
  - mempro.c, [142](#)
  - mempro.h, [157](#)
- aggregation.c
  - AK\_agg\_input\_add, [397](#)
  - AK\_agg\_input\_add\_to\_beginning, [398](#)
  - AK\_agg\_input\_fix, [398](#)
  - AK\_agg\_input\_init, [399](#)
  - AK\_aggregation, [399](#)
  - AK\_header\_size, [400](#)
  - AK\_search\_unsorted, [401](#)
  - Ak\_aggregation\_test, [400](#)
- aggregation.h
  - AK\_agg\_input\_add, [403](#)
  - AK\_agg\_input\_add\_to\_beginning, [404](#)
  - AK\_agg\_input\_fix, [404](#)
  - AK\_agg\_input\_init, [405](#)
  - AK\_aggregation, [405](#)
  - AK\_header\_size, [406](#)
  - Ak\_aggregation\_test, [406](#)
- Ak\_Delete\_All\_elementsAd
  - index.c, [272](#)
  - index.h, [281](#)
- Ak\_Delete\_L3
  - auxiliary.c, [58](#)
  - auxiliary.h, [76](#)
- Ak\_Delete\_elementAd
  - index.c, [272](#)
  - index.h, [282](#)
- Ak\_DeleteAll\_L3
  - auxiliary.c, [59](#)
  - auxiliary.h, [76](#)
- Ak\_End\_L2
  - auxiliary.c, [59](#)
  - auxiliary.h, [77](#)
- Ak\_First\_L2
  - auxiliary.c, [60](#)
  - auxiliary.h, [79](#)
- Ak\_Get\_First\_elementAd
  - index.c, [273](#)
  - index.h, [282](#)
- Ak\_Get\_Last\_elementAd
  - index.c, [275](#)
  - index.h, [284](#)
- Ak\_Get\_Next\_elementAd
  - index.c, [275](#)
  - index.h, [284](#)
- Ak\_Get\_Position\_Of\_elementAd
  - index.c, [276](#)
  - index.h, [285](#)
- Ak\_Get\_Previous\_elementAd
  - index.c, [276](#)
  - index.h, [285](#)
- Ak\_GetNth\_L2
  - auxiliary.c, [61](#)
  - auxiliary.h, [80](#)
- Ak\_If\_ExistOp
  - bitmap.c, [243](#)
  - bitmap.h, [250](#)
- Ak\_Init\_L3
  - auxiliary.c, [63](#)
  - auxiliary.h, [81](#)
- Ak\_InitializelistAd
  - index.c, [277](#)
  - index.h, [286](#)
- Ak\_Insert\_New\_Element
  - fileio.c, [213](#)
  - fileio.h, [219](#)
  - reference.h, [492](#)
- Ak\_Insert\_New\_Element\_For\_Update
  - fileio.c, [214](#)
  - fileio.h, [220](#)
  - reference.h, [493](#)
- Ak\_Insert\_NewelementAd
  - index.c, [278](#)
  - index.h, [287](#)
- Ak\_InsertAfter\_L2
  - auxiliary.c, [63](#)
  - auxiliary.h, [82](#)
- Ak\_InsertAtBegin\_L3
  - auxiliary.c, [64](#)
  - auxiliary.h, [82](#)
- Ak\_InsertAtEnd\_L3
  - auxiliary.c, [64](#)
  - auxiliary.h, [83](#)



- Ak\_InsertBefore\_L2
  - auxiliary.c, [65](#)
  - auxiliary.h, [83](#)
- Ak\_IsEmpty\_L2
  - auxiliary.c, [65](#)
  - auxiliary.h, [84](#)
- Ak\_Next\_L2
  - auxiliary.c, [66](#)
  - auxiliary.h, [85](#)
- Ak\_Previous\_L2
  - auxiliary.c, [67](#)
  - auxiliary.h, [86](#)
- Ak\_Retrieve\_L2
  - auxiliary.c, [68](#)
  - auxiliary.h, [87](#)
- Ak\_Size\_L2
  - auxiliary.c, [70](#)
  - auxiliary.h, [89](#)
- Ak\_aggregation\_test
  - aggregation.c, [400](#)
  - aggregation.h, [406](#)
- Ak\_bitmap\_test
  - bitmap.c, [240](#)
  - bitmap.h, [248](#)
- Ak\_check\_constraint\_name
  - constraint\_names.c, [474](#)
  - constraint\_names.h, [476](#)
- Ak\_check\_regex\_expression
  - expression\_check.c, [413](#)
  - expression\_check.h, [416](#)
- Ak\_check\_regex\_operator\_expression
  - expression\_check.c, [413](#)
  - expression\_check.h, [417](#)
- Ak\_constraint\_between\_test
  - between.c, [461](#)
  - between.h, [466](#)
- Ak\_create\_Index
  - bitmap.c, [241](#)
  - bitmap.h, [248](#)
- Ak\_dbg\_messg
  - debug.c, [96](#)
  - debug.h, [98](#)
- Ak\_delete\_row
  - fileio.c, [211](#)
  - fileio.h, [217](#)
  - reference.h, [491](#)
- Ak\_delete\_row\_by\_id
  - fileio.c, [212](#)
  - fileio.h, [218](#)
- Ak\_delete\_row\_from\_block
  - fileio.c, [212](#)
  - fileio.h, [218](#)
- Ak\_delete\_update\_segment
  - fileio.c, [212](#)
  - fileio.h, [219](#)
- Ak\_files\_test
  - files.c, [223](#)
  - files.h, [226](#)
- Ak\_filesearch\_test
  - filesearch.c, [228](#)
  - filesearch.h, [232](#)
- Ak\_get\_Attribute
  - bitmap.c, [242](#)
  - bitmap.h, [249](#)
- Ak\_get\_header\_number
  - filesort.h, [234](#)
- Ak\_get\_nth\_main\_bucket\_add
  - hash.c, [261](#)
  - hash.h, [268](#)
- Ak\_get\_num\_of\_tuples
  - filesort.h, [234](#)
- Ak\_get\_total\_headers
  - filesort.h, [235](#)
- Ak\_hash\_test
  - hash.c, [262](#)
  - hash.h, [269](#)
- Ak\_id\_test
  - id.c, [236](#)
  - id.h, [238](#)
- Ak\_insert\_bucket\_to\_block
  - hash.c, [262](#)
  - hash.h, [269](#)
- Ak\_insert\_row
  - fileio.c, [214](#)
  - fileio.h, [220](#)
  - reference.h, [493](#)
- Ak\_insert\_row\_to\_block
  - fileio.c, [215](#)
  - fileio.h, [221](#)
- Ak\_op\_difference\_test
  - difference.c, [408](#)
  - difference.h, [410](#)
- Ak\_op\_intersect\_test
  - intersect.c, [418](#)
  - intersect.h, [420](#)
- Ak\_print\_Att\_Test
  - bitmap.c, [244](#)
  - bitmap.h, [251](#)
- Ak\_print\_Header\_Test
  - bitmap.c, [244](#)
  - bitmap.h, [251](#)
- Ak\_set\_constraint\_unique
  - unique.c, [499](#)
  - unique.h, [502](#)
- Ak\_update\_bucket\_in\_block
  - hash.c, [263](#)
  - hash.h, [270](#)
- Ak\_update\_row
  - fileio.c, [215](#)
  - fileio.h, [221](#)
  - reference.h, [497](#)
- Ak\_update\_row\_from\_block
  - fileio.c, [216](#)
  - fileio.h, [222](#)
- Ak\_write\_block
  - bitmap.c, [245](#)

- bitmap.h, 252
- archive\_log.h
  - AK\_archive\_log, 390
  - AK\_empty\_archive\_log, 390
  - AK\_get\_timestamp, 390
- auxi/auxiliary.c, 55
- auxi/auxiliary.h, 72
- auxi/constants.h, 91
- auxi/debug.c, 95
- auxi/debug.h, 96
- auxi/dictionary.c, 99
- auxi/dictionary.h, 103
- auxi/iniparser.c, 108
- auxi/iniparser.h, 118
- auxi/mempro.c, 128
- auxi/mempro.h, 143
- auxi/observable.c, 157
- auxi/observable.h, 160
- auxiliary.c
  - AK\_add\_sucesor, 57
  - AK\_add\_vertex, 57
  - AK\_chars\_num\_from\_number, 58
  - AK\_destroy\_critical\_section, 59
  - AK\_enter\_critical\_section, 60
  - AK\_get\_array\_perms, 61
  - AK\_init\_critical\_section, 62
  - AK\_leave\_critical\_section, 66
  - AK\_pop\_from\_stack, 66
  - AK\_push\_to\_stack, 67
  - AK\_search\_empty\_link, 68
  - AK\_search\_empty\_stack\_link, 69
  - AK\_search\_in\_stack, 69
  - AK\_search\_vertex, 69
  - AK\_strcmp, 70
  - AK\_tarjan, 71
  - AK\_type\_size, 71
  - Ak\_Delete\_L3, 58
  - Ak\_DeleteAll\_L3, 59
  - Ak\_End\_L2, 59
  - Ak\_First\_L2, 60
  - Ak\_GetNth\_L2, 61
  - Ak\_Init\_L3, 63
  - Ak\_InsertAfter\_L2, 63
  - Ak\_InsertAtBegin\_L3, 64
  - Ak\_InsertAtEnd\_L3, 64
  - Ak\_InsertBefore\_L2, 65
  - Ak\_IsEmpty\_L2, 65
  - Ak\_Next\_L2, 66
  - Ak\_Previous\_L2, 67
  - Ak\_Retrieve\_L2, 68
  - Ak\_Size\_L2, 70
- auxiliary.h
  - AK\_add\_sucesor, 75
  - AK\_add\_vertex, 75
  - AK\_chars\_num\_from\_number, 76
  - AK\_destroy\_critical\_section, 77
  - AK\_enter\_critical\_section, 79
  - AK\_get\_array\_perms, 80
  - AK\_init\_critical\_section, 81
  - AK\_leave\_critical\_section, 84
  - AK\_pop\_from\_stack, 85
  - AK\_push\_to\_stack, 86
  - AK\_search\_empty\_link, 87
  - AK\_search\_empty\_stack\_link, 87
  - AK\_search\_in\_stack, 88
  - AK\_search\_vertex, 88
  - AK\_strcmp, 89
  - AK\_tarjan, 90
  - AK\_type\_size, 90
  - Ak\_Delete\_L3, 76
  - Ak\_DeleteAll\_L3, 76
  - Ak\_End\_L2, 77
  - Ak\_First\_L2, 79
  - Ak\_GetNth\_L2, 80
  - Ak\_Init\_L3, 81
  - Ak\_InsertAfter\_L2, 82
  - Ak\_InsertAtBegin\_L3, 82
  - Ak\_InsertAtEnd\_L3, 83
  - Ak\_InsertBefore\_L2, 83
  - Ak\_IsEmpty\_L2, 84
  - Ak\_Next\_L2, 85
  - Ak\_Previous\_L2, 86
  - Ak\_Retrieve\_L2, 87
  - Ak\_Size\_L2, 89
  - testMode, 91
- between.c
  - AK\_delete\_constraint\_between, 462
  - AK\_find\_table\_address, 462
  - AK\_print\_constraints, 463
  - AK\_read\_constraint\_between, 463
  - AK\_set\_constraint\_between, 464
  - Ak\_constraint\_between\_test, 461
- between.h
  - AK\_find\_table\_address, 466
  - AK\_read\_constraint\_between, 466
  - AK\_set\_constraint\_between, 467
  - Ak\_constraint\_between\_test, 466
- bitmap.c
  - AK\_add\_to\_bitmap\_index, 240
  - AK\_create\_Index\_Table, 241
  - AK\_delete\_bitmap\_index, 242
  - AK\_get\_Attribute, 243
  - AK\_update, 244
  - Ak\_If\_ExistOp, 243
  - Ak\_bitmap\_test, 240
  - Ak\_create\_Index, 241
  - Ak\_get\_Attribute, 242
  - Ak\_print\_Att\_Test, 244
  - Ak\_print\_Header\_Test, 244
  - Ak\_write\_block, 245
- bitmap.h
  - AK\_add\_to\_bitmap\_index, 247
  - AK\_create\_Index\_Table, 249
  - AK\_delete\_bitmap\_index, 249
  - AK\_get\_Attribute, 250
  - AK\_update, 252

- Ak\_if\_ExistOp, [250](#)
- Ak\_bitmap\_test, [248](#)
- Ak\_create\_Index, [248](#)
- Ak\_get\_Attribute, [249](#)
- Ak\_print\_Att\_Test, [251](#)
- Ak\_print\_Header\_Test, [251](#)
- Ak\_write\_block, [252](#)
- blobs.c
  - AK\_GUID, [203](#)
  - AK\_check\_folder\_blobs, [202](#)
  - AK\_concat, [202](#)
  - AK\_folder\_exists, [202](#)
  - AK\_lo\_export, [203](#)
  - AK\_lo\_import, [203](#)
  - AK\_lo\_test, [204](#)
  - AK\_lo\_unlink, [204](#)
  - AK\_mkdir, [204](#)
  - AK\_split\_path\_file, [205](#)
- blobs.h
  - AK\_GUID, [208](#)
  - AK\_check\_folder\_blobs, [207](#)
  - AK\_concat, [207](#)
  - AK\_folder\_exists, [207](#)
  - AK\_lo\_export, [208](#)
  - AK\_lo\_import, [208](#)
  - AK\_lo\_test, [209](#)
  - AK\_lo\_unlink, [209](#)
  - AK\_mkdir, [209](#)
  - AK\_split\_path\_file, [210](#)
- blocktable, [29](#)
- btree.c
  - AK\_btree\_create, [254](#)
  - AK\_btree\_search\_delete, [254](#)
- btree.h
  - AK\_btree\_create, [256](#)
  - AK\_btree\_search\_delete, [256](#)
- btree\_node, [29](#)
- bucket\_elem, [30](#)
- CHAR\_IN\_LINE
  - dbman.h, [184](#)
- check\_constraint.c
  - AK\_check\_constraint, [468](#)
  - AK\_check\_constraint\_test, [469](#)
  - AK\_set\_check\_constraint, [469](#)
  - condition\_passed, [470](#)
- check\_constraint.h
  - AK\_check\_constraint, [472](#)
  - AK\_check\_constraint\_test, [472](#)
  - AK\_set\_check\_constraint, [472](#)
  - condition\_passed, [473](#)
- condition\_passed
  - check\_constraint.c, [470](#)
  - check\_constraint.h, [473](#)
- constraint\_names.c
  - AK\_constraint\_names\_test, [475](#)
  - AK\_check\_constraint\_name, [474](#)
- constraint\_names.h
  - AK\_constraint\_names\_test, [477](#)
  - AK\_check\_constraint\_name, [476](#)
- cost\_eval\_t, [31](#)
- create\_header\_test
  - test.c, [324](#)
  - test.h, [329](#)
- DEBUG\_ALL
  - debug.h, [98](#)
- DEBUG\_LEVEL, [31](#)
- DEBUG\_TYPE, [32](#)
- DICT\_INVALID\_KEY
  - dictionary.c, [100](#)
- DICTMINSZ
  - dictionary.c, [100](#)
- db
  - dbman.h, [200](#)
- db\_file\_size
  - dbman.h, [200](#)
- dbman.c
  - AK\_allocate\_block\_activity\_modes, [165](#)
  - AK\_allocate\_blocks, [165](#)
  - AK\_allocationtable\_dump, [165](#)
  - AK\_blocktable\_dump, [166](#)
  - AK\_blocktable\_flush, [166](#)
  - AK\_blocktable\_get, [166](#)
  - AK\_copy\_header, [166](#)
  - AK\_create\_header, [167](#)
  - AK\_delete\_block, [168](#)
  - AK\_delete\_extent, [168](#)
  - AK\_delete\_segment, [169](#)
  - AK\_get\_allocation\_set, [169](#)
  - AK\_get\_extent, [170](#)
  - AK\_increase\_extent, [170](#)
  - AK\_init\_allocation\_table, [171](#)
  - AK\_init\_block, [171](#)
  - AK\_init\_db\_file, [171](#)
  - AK\_init\_disk\_manager, [172](#)
  - AK\_init\_system\_catalog, [172](#)
  - AK\_init\_system\_tables\_catalog, [172](#)
  - AK\_insert\_entry, [174](#)
  - AK\_memset\_int, [174](#)
  - AK\_new\_extent, [175](#)
  - AK\_new\_segment, [176](#)
  - AK\_print\_block, [176](#)
  - AK\_read\_block, [176](#)
  - AK\_read\_block\_for\_testing, [177](#)
  - AK\_register\_system\_tables, [177](#)
  - AK\_thread\_safe\_block\_access\_test, [178](#)
  - AK\_write\_block, [179](#)
  - AK\_write\_block\_for\_testing, [179](#)
  - fsize, [179](#)
- dbman.h
  - AK\_ALLOCATION\_TABLE\_SIZE, [183](#)
  - AK\_allocate\_blocks, [185](#)
  - AK\_allocation\_set\_mode, [184](#)
  - AK\_allocationbit, [200](#)
  - AK\_allocationtable\_dump, [185](#)
  - AK\_blocktable\_dump, [185](#)
  - AK\_blocktable\_flush, [186](#)

- AK\_blocktable\_get, 186
- AK\_copy\_header, 186
- AK\_create\_header, 187
- AK\_delete\_block, 187
- AK\_delete\_extent, 188
- AK\_delete\_segment, 188
- AK\_get\_allocation\_set, 189
- AK\_get\_extent, 189
- AK\_increase\_extent, 190
- AK\_init\_allocation\_table, 191
- AK\_init\_block, 191
- AK\_init\_db\_file, 191
- AK\_init\_disk\_manager, 192
- AK\_init\_system\_catalog, 192
- AK\_init\_system\_tables\_catalog, 192
- AK\_insert\_entry, 194
- AK\_memset\_int, 194
- AK\_new\_extent, 195
- AK\_new\_segment, 196
- AK\_print\_block, 196
- AK\_read\_block, 196
- AK\_read\_block\_for\_testing, 197
- AK\_register\_system\_tables, 197
- AK\_thread\_safe\_block\_access\_test, 198
- AK\_write\_block, 199
- AK\_write\_block\_for\_testing, 199
- CHAR\_IN\_LINE, 184
- db, 200
- db\_file\_size, 200
- fsize, 199
- MAX\_BLOCK\_INIT\_NUM, 184
- debug.c
  - Ak\_dbg\_messg, 96
- debug.h
  - Ak\_dbg\_messg, 98
  - DEBUG\_ALL, 98
- dictionary
  - dictionary.h, 105
- dictionary.c
  - DICT\_INVALID\_KEY, 100
  - DICTMINSZ, 100
  - dictionary\_del, 100
  - dictionary\_dump, 101
  - dictionary\_get, 101
  - dictionary\_hash, 101
  - dictionary\_new, 102
  - dictionary\_set, 102
  - dictionary\_unset, 103
  - MAXVALSZ, 100
- dictionary.h
  - dictionary, 105
  - dictionary\_del, 105
  - dictionary\_dump, 106
  - dictionary\_get, 106
  - dictionary\_hash, 106
  - dictionary\_new, 107
  - dictionary\_set, 107
  - dictionary\_unset, 108
- dictionary\_del
  - dictionary.c, 100
  - dictionary.h, 105
- dictionary\_dump
  - dictionary.c, 101
  - dictionary.h, 106
- dictionary\_get
  - dictionary.c, 101
  - dictionary.h, 106
- dictionary\_hash
  - dictionary.c, 101
  - dictionary.h, 106
- dictionary\_new
  - dictionary.c, 102
  - dictionary.h, 107
- dictionary\_set
  - dictionary.c, 102
  - dictionary.h, 107
- dictionary\_unset
  - dictionary.c, 103
  - dictionary.h, 108
- difference.c
  - AK\_difference, 407
  - Ak\_op\_difference\_test, 408
- difference.h
  - AK\_difference, 409
  - Ak\_op\_difference\_test, 410
- dm/dbman.c, 162
- dm/dbman.h, 180
- drop.c
  - AK\_drop, 504
  - AK\_drop\_help\_function, 505
  - AK\_drop\_test, 505
  - AK\_if\_exist, 505
  - system\_catalog, 506
- drop.h
  - AK\_drop, 507
  - AK\_drop\_test, 508
  - AK\_if\_exist, 508
- drop\_arguments, 32
- expression\_check.c
  - AK\_check\_arithmetic\_statement, 411
  - AK\_check\_if\_row\_satisfies\_expression, 412
  - AK\_replace\_wild\_card, 413
  - Ak\_check\_regex\_expression, 413
  - Ak\_check\_regex\_operator\_expression, 413
- expression\_check.h
  - AK\_check\_arithmetic\_statement, 415
  - AK\_check\_if\_row\_satisfies\_expression, 416
  - Ak\_check\_regex\_expression, 416
  - Ak\_check\_regex\_operator\_expression, 417
- file/blobs.c, 201
- file/blobs.h, 205
- file/fileio.c, 210
- file/fileio.h, 216
- file/files.c, 222
- file/files.h, 225

- file/filesearch.c, [227](#)
- file/filesearch.h, [230](#)
- file/filesort.h, [233](#)
- file/id.c, [235](#)
- file/id.h, [237](#)
- file/idx/bitmap.c, [239](#)
- file/idx/bitmap.h, [246](#)
- file/idx/btree.c, [253](#)
- file/idx/btree.h, [255](#)
- file/idx/hash.c, [257](#)
- file/idx/hash.h, [264](#)
- file/idx/index.c, [271](#)
- file/idx/index.h, [279](#)
- file/sequence.c, [288](#)
- file/sequence.h, [293](#)
- file/table.c, [298](#)
- file/table.h, [310](#)
- file/test.c, [323](#)
- file/test.h, [327](#)
- fileio.c
  - Ak\_Insert\_New\_Element, [213](#)
  - Ak\_Insert\_New\_Element\_For\_Update, [214](#)
  - Ak\_delete\_row, [211](#)
  - Ak\_delete\_row\_by\_id, [212](#)
  - Ak\_delete\_row\_from\_block, [212](#)
  - Ak\_delete\_update\_segment, [212](#)
  - Ak\_insert\_row, [214](#)
  - Ak\_insert\_row\_to\_block, [215](#)
  - Ak\_update\_row, [215](#)
  - Ak\_update\_row\_from\_block, [216](#)
- fileio.h
  - Ak\_Insert\_New\_Element, [219](#)
  - Ak\_Insert\_New\_Element\_For\_Update, [220](#)
  - Ak\_delete\_row, [217](#)
  - Ak\_delete\_row\_by\_id, [218](#)
  - Ak\_delete\_row\_from\_block, [218](#)
  - Ak\_delete\_update\_segment, [219](#)
  - Ak\_insert\_row, [220](#)
  - Ak\_insert\_row\_to\_block, [221](#)
  - Ak\_update\_row, [221](#)
  - Ak\_update\_row\_from\_block, [222](#)
- files.c
  - AK\_initialize\_new\_index\_segment, [223](#)
  - AK\_initialize\_new\_segment, [224](#)
  - Ak\_files\_test, [223](#)
- files.h
  - AK\_initialize\_new\_index\_segment, [226](#)
  - AK\_initialize\_new\_segment, [226](#)
  - Ak\_files\_test, [226](#)
- filesearch.c
  - AK\_deallocate\_search\_result, [228](#)
  - AK\_search\_unsorted, [229](#)
  - Ak\_filesearch\_test, [228](#)
- filesearch.h
  - AK\_deallocate\_search\_result, [231](#)
  - AK\_search\_unsorted, [232](#)
  - Ak\_filesearch\_test, [232](#)
- filesort.h
  - AK\_block\_sort, [234](#)
  - Ak\_get\_header\_number, [234](#)
  - Ak\_get\_num\_of\_tuples, [234](#)
  - Ak\_get\_total\_headers, [235](#)
- fsize
  - dbman.c, [179](#)
  - dbman.h, [199](#)
- function.c
  - AK\_check\_function\_arguments, [510](#)
  - AK\_check\_function\_arguments\_type, [510](#)
  - AK\_function\_add, [511](#)
  - AK\_function\_arguments\_add, [511](#)
  - AK\_function\_arguments\_remove\_by\_obj\_id, [512](#)
  - AK\_function\_change\_return\_type, [512](#)
  - AK\_function\_remove\_by\_name, [513](#)
  - AK\_function\_remove\_by\_obj\_id, [513](#)
  - AK\_function\_rename, [513](#)
  - AK\_function\_test, [514](#)
  - AK\_get\_function\_obj\_id, [514](#)
- function.h
  - AK\_check\_function\_arguments, [516](#)
  - AK\_check\_function\_arguments\_type, [517](#)
  - AK\_function\_add, [517](#)
  - AK\_function\_arguments\_add, [518](#)
  - AK\_function\_arguments\_remove\_by\_obj\_id, [518](#)
  - AK\_function\_change\_return\_type, [519](#)
  - AK\_function\_remove\_by\_name, [519](#)
  - AK\_function\_remove\_by\_obj\_id, [520](#)
  - AK\_function\_rename, [520](#)
  - AK\_function\_test, [521](#)
  - AK\_get\_function\_obj\_id, [521](#)
- get\_column\_test
  - test.c, [325](#)
  - test.h, [329](#)
- get\_row\_attr\_data
  - table.c, [310](#)
  - table.h, [322](#)
- get\_row\_test
  - test.c, [325](#)
  - test.h, [330](#)
- grandfailure
  - recovery.c, [394](#)
- handle\_transaction\_notify
  - transaction.c, [566](#)
  - transaction.h, [582](#)
- hash
  - \_dictionary\_, [11](#)
- hash.c
  - AK\_change\_hash\_info, [258](#)
  - AK\_create\_hash\_index, [258](#)
  - AK\_delete\_in\_hash\_index, [259](#)
  - AK\_elem\_hash\_value, [259](#)
  - AK\_find\_delete\_in\_hash\_index, [260](#)
  - AK\_find\_in\_hash\_index, [260](#)
  - AK\_get\_hash\_info, [261](#)
  - AK\_insert\_in\_hash\_index, [263](#)
  - Ak\_get\_nth\_main\_bucket\_add, [261](#)

- Ak\_hash\_test, 262
- Ak\_insert\_bucket\_to\_block, 262
- Ak\_update\_bucket\_in\_block, 263
- hash.h
  - AK\_change\_hash\_info, 265
  - AK\_create\_hash\_index, 266
  - AK\_delete\_in\_hash\_index, 266
  - AK\_elem\_hash\_value, 267
  - AK\_find\_delete\_in\_hash\_index, 267
  - AK\_find\_in\_hash\_index, 268
  - AK\_get\_hash\_info, 268
  - AK\_insert\_in\_hash\_index, 270
  - Ak\_get\_nth\_main\_bucket\_add, 268
  - Ak\_hash\_test, 269
  - Ak\_insert\_bucket\_to\_block, 269
  - Ak\_update\_bucket\_in\_block, 270
- hash\_bucket, 33
- hash\_info, 34
- id.c
  - AK\_get\_id, 236
  - AK\_get\_table\_id, 236
  - Ak\_id\_test, 236
- id.h
  - AK\_get\_id, 238
  - Ak\_id\_test, 238
- index.c
  - AK\_get\_index\_header, 273
  - AK\_get\_index\_num\_records, 274
  - AK\_get\_index\_tuple, 274
  - AK\_index\_table\_exist, 277
  - AK\_index\_test, 277
  - AK\_num\_index\_attr, 278
  - AK\_print\_index\_table, 279
  - Ak\_Delete\_All\_elementsAd, 272
  - Ak\_Delete\_elementAd, 272
  - Ak\_Get\_First\_elementAd, 273
  - Ak\_Get\_Last\_elementAd, 275
  - Ak\_Get\_Next\_elementAd, 275
  - Ak\_Get\_Position\_Of\_elementAd, 276
  - Ak\_Get\_Previous\_elementAd, 276
  - Ak\_InitializelistAd, 277
  - Ak\_Insert\_NewelementAd, 278
- index.h
  - AK\_get\_index\_num\_records, 283
  - AK\_get\_index\_tuple, 283
  - AK\_index\_table\_exist, 286
  - AK\_index\_test, 286
  - AK\_num\_index\_attr, 287
  - AK\_print\_index\_table, 288
  - Ak\_Delete\_All\_elementsAd, 281
  - Ak\_Delete\_elementAd, 282
  - Ak\_Get\_First\_elementAd, 282
  - Ak\_Get\_Last\_elementAd, 284
  - Ak\_Get\_Next\_elementAd, 284
  - Ak\_Get\_Position\_Of\_elementAd, 285
  - Ak\_Get\_Previous\_elementAd, 285
  - Ak\_InitializelistAd, 286
  - Ak\_Insert\_NewelementAd, 287
- iniparser.c
  - \_line\_status\_, 110
  - iniparser\_AK\_freedict, 111
  - iniparser\_dump, 111
  - iniparser\_dump\_ini, 111
  - iniparser\_dumpsection\_ini, 112
  - iniparser\_find\_entry, 112
  - iniparser\_getboolean, 113
  - iniparser\_getdouble, 113
  - iniparser\_getint, 114
  - iniparser\_getnsec, 114
  - iniparser\_getseckey, 115
  - iniparser\_getsecname, 115
  - iniparser\_getsecnkeys, 116
  - iniparser\_getstring, 116
  - iniparser\_load, 117
  - iniparser\_set, 117
  - iniparser\_unset, 117
  - line\_status, 110
- iniparser.h
  - iniparser\_AK\_freedict, 120
  - iniparser\_dump, 120
  - iniparser\_dump\_ini, 121
  - iniparser\_dumpsection\_ini, 121
  - iniparser\_find\_entry, 121
  - iniparser\_getboolean, 122
  - iniparser\_getdouble, 123
  - iniparser\_getint, 123
  - iniparser\_getnsec, 124
  - iniparser\_getseckey, 125
  - iniparser\_getsecname, 125
  - iniparser\_getsecnkeys, 126
  - iniparser\_getstring, 126
  - iniparser\_load, 126
  - iniparser\_set, 127
  - iniparser\_unset, 127
- iniparser\_AK\_freedict
  - iniparser.c, 111
  - iniparser.h, 120
- iniparser\_dump
  - iniparser.c, 111
  - iniparser.h, 120
- iniparser\_dump\_ini
  - iniparser.c, 111
  - iniparser.h, 121
- iniparser\_dumpsection\_ini
  - iniparser.c, 112
  - iniparser.h, 121
- iniparser\_find\_entry
  - iniparser.c, 112
  - iniparser.h, 121
- iniparser\_getboolean
  - iniparser.c, 113
  - iniparser.h, 122
- iniparser\_getdouble
  - iniparser.c, 113
  - iniparser.h, 123
- iniparser\_getint
  - iniparser.c, 114
  - iniparser.h, 124

- iniparser.c, 114
- iniparser.h, 123
- iniparser\_getnsec
  - iniparser.c, 114
  - iniparser.h, 124
- iniparser\_getseckeys
  - iniparser.c, 115
  - iniparser.h, 125
- iniparser\_getsecname
  - iniparser.c, 115
  - iniparser.h, 125
- iniparser\_getsecnkeys
  - iniparser.c, 116
  - iniparser.h, 126
- iniparser\_getstring
  - iniparser.c, 116
  - iniparser.h, 126
- iniparser\_load
  - iniparser.c, 117
  - iniparser.h, 126
- iniparser\_set
  - iniparser.c, 117
  - iniparser.h, 127
- iniparser\_unset
  - iniparser.c, 117
  - iniparser.h, 127
- insert\_data\_test
  - test.c, 326
  - test.h, 330
- intersect.c
  - AK\_intersect, 418
  - Ak\_op\_intersect\_test, 418
- intersect.h
  - AK\_intersect, 420
  - Ak\_op\_intersect\_test, 420
- intersect\_attr, 34
- key
  - \_dictionary\_, 11
- line\_status
  - iniparser.c, 110
- list\_node, 35
- list\_structure\_ad, 36
- list\_structure\_add, 36
- MAX\_BLOCK\_INIT\_NUM
  - dbman.h, 184
- MAXVALSZ
  - dictionary.c, 100
- main\_bucket, 37
- memoman.c
  - AK\_cache\_AK\_malloc, 333
  - AK\_cache\_block, 333
  - AK\_cache\_result, 334
  - AK\_find\_AK\_free\_space, 334
  - AK\_find\_available\_result\_block, 335
  - AK\_flush\_cache, 335
  - AK\_generate\_result\_id, 335
  - AK\_get\_block, 335
  - AK\_get\_index\_addresses, 336
  - AK\_get\_index\_segment\_addresses, 336
  - AK\_get\_segment\_addresses, 337
  - AK\_get\_table\_addresses, 337
  - AK\_init\_new\_extent, 338
  - AK\_mem\_block\_modify, 338
  - AK\_memoman\_init, 338
  - AK\_query\_mem\_AK\_malloc, 339
  - AK\_redo\_log\_AK\_malloc, 339
  - AK\_refresh\_cache, 339
- memoman.h
  - AK\_cache\_AK\_malloc, 342
  - AK\_cache\_block, 342
  - AK\_cache\_result, 343
  - AK\_find\_AK\_free\_space, 343
  - AK\_find\_available\_result\_block, 344
  - AK\_flush\_cache, 344
  - AK\_generate\_result\_id, 344
  - AK\_get\_block, 344
  - AK\_get\_index\_addresses, 345
  - AK\_get\_index\_segment\_addresses, 345
  - AK\_get\_segment\_addresses, 346
  - AK\_get\_table\_addresses, 346
  - AK\_init\_new\_extent, 347
  - AK\_mem\_block\_modify, 347
  - AK\_memoman\_init, 347
  - AK\_query\_mem\_AK\_malloc, 348
  - AK\_redo\_log\_AK\_malloc, 348
  - AK\_refresh\_cache, 348
- memoryAddresses, 38
- mempro.c
  - AK\_calloc, 130
  - AK\_check\_for\_writes, 130
  - AK\_debmod\_calloc, 130
  - AK\_debmod\_d, 131
  - AK\_debmod\_die, 131
  - AK\_debmod\_dv, 132
  - AK\_debmod\_enter\_critical\_sec, 132
  - AK\_debmod\_free, 132
  - AK\_debmod\_fstack\_pop, 133
  - AK\_debmod\_fstack\_push, 133
  - AK\_debmod\_func\_add, 134
  - AK\_debmod\_func\_get\_name, 134
  - AK\_debmod\_func\_id, 135
  - AK\_debmod\_function\_current, 135
  - AK\_debmod\_function\_epilogue, 136
  - AK\_debmod\_function\_prologue, 136
  - AK\_debmod\_init, 137
  - AK\_debmod\_leave\_critical\_sec, 137
  - AK\_debmod\_log\_memory\_alloc, 137
  - AK\_debmod\_print\_function\_use, 138
  - AK\_fread, 138
  - AK\_free, 139
  - AK\_fwrite, 139
  - AK\_malloc, 140
  - AK\_mempro\_test, 140
  - AK\_print\_active\_functions, 140



- AK\_print\_function\_use, 140
- AK\_print\_function\_uses, 141
- AK\_realloc, 141
- AK\_write\_protect, 142
- AK\_write\_unprotect, 142
- mempro.h
  - AK\_calloc, 145
  - AK\_check\_for\_writes, 145
  - AK\_debmod\_calloc, 146
  - AK\_debmod\_d, 146
  - AK\_debmod\_die, 147
  - AK\_debmod\_dv, 147
  - AK\_debmod\_enter\_critical\_sec, 148
  - AK\_debmod\_free, 148
  - AK\_debmod\_fstack\_pop, 148
  - AK\_debmod\_fstack\_push, 149
  - AK\_debmod\_func\_add, 149
  - AK\_debmod\_func\_get\_name, 150
  - AK\_debmod\_func\_id, 150
  - AK\_debmod\_function\_current, 151
  - AK\_debmod\_function\_epilogue, 151
  - AK\_debmod\_function\_prologue, 152
  - AK\_debmod\_init, 152
  - AK\_debmod\_leave\_critical\_sec, 152
  - AK\_debmod\_log\_memory\_alloc, 153
  - AK\_debmod\_print\_function\_use, 153
  - AK\_free, 154
  - AK\_malloc, 154
  - AK\_mempro\_test, 155
  - AK\_print\_active\_functions, 155
  - AK\_print\_function\_use, 155
  - AK\_print\_function\_uses, 156
  - AK\_realloc, 156
  - AK\_write\_protect, 156
  - AK\_write\_unprotect, 157
- mm/memoman.c, 332
- mm/memoman.h, 340
- nat\_join.c
  - AK\_copy\_blocks\_join, 422
  - AK\_create\_join\_block\_header, 422
  - AK\_join, 423
  - AK\_merge\_block\_join, 423
  - AK\_op\_join\_test, 424
- nat\_join.h
  - AK\_copy\_blocks\_join, 425
  - AK\_create\_join\_block\_header, 426
  - AK\_join, 427
  - AK\_merge\_block\_join, 427
  - AK\_op\_join\_test, 428
- nnull.c
  - AK\_delete\_constraint\_not\_null, 478
  - AK\_null\_test, 478
  - AK\_read\_constraint\_not\_null, 478
  - AK\_set\_constraint\_not\_null, 479
- nnull.h
  - AK\_delete\_constraint\_not\_null, 481
  - AK\_null\_test, 481
  - AK\_read\_constraint\_not\_null, 481
  - AK\_set\_constraint\_not\_null, 482
- NoticeType
  - transaction.h, 569
- Observable, 38
- observable.c
  - AK\_init\_observable, 159
  - AK\_init\_observer, 159
  - AK\_observable\_test, 159
- observable.h
  - AK\_init\_observable, 161
  - AK\_init\_observer, 161
  - AK\_observable\_test, 162
- observable\_transaction, 39
- observable\_transaction\_struct, 40
- Observer, 40
- observer\_lock, 41
- opti/query\_optimization.c, 349
- opti/query\_optimization.h, 352
- opti/rel\_eq\_assoc.c, 355
- opti/rel\_eq\_assoc.h, 357
- opti/rel\_eq\_comut.c, 360
- opti/rel\_eq\_comut.h, 363
- opti/rel\_eq\_projection.c, 366
- opti/rel\_eq\_projection.h, 371
- opti/rel\_eq\_selection.c, 377
- opti/rel\_eq\_selection.h, 383
- privileges.c
  - AK\_add\_user\_to\_group, 523
  - AK\_check\_group\_privilege, 523
  - AK\_check\_privilege, 524
  - AK\_check\_user\_privilege, 524
  - AK\_grant\_privilege\_group, 525
  - AK\_grant\_privilege\_user, 525
  - AK\_group\_add, 526
  - AK\_group\_get\_id, 526
  - AK\_group\_remove\_by\_name, 527
  - AK\_group\_rename, 527
  - AK\_privileges\_test, 528
  - AK\_remove\_all\_users\_from\_group, 528
  - AK\_remove\_user\_from\_all\_groups, 528
  - AK\_revoke\_all\_privileges\_group, 529
  - AK\_revoke\_all\_privileges\_user, 529
  - AK\_revoke\_privilege\_group, 530
  - AK\_revoke\_privilege\_user, 530
  - AK\_user\_add, 531
  - AK\_user\_get\_id, 531
  - AK\_user\_remove\_by\_name, 532
  - AK\_user\_rename, 532
- product.c
  - AK\_op\_product\_test, 429
  - AK\_product, 429
- product.h
  - AK\_op\_product\_test, 431
  - AK\_product, 431
- projection.c
  - AK\_copy\_block\_projection, 433
  - AK\_create\_block\_header, 433



- AK\_create\_header\_name, 434
- AK\_determine\_header\_type, 434
- AK\_get\_operator, 435
- AK\_op\_projection\_test, 435
- AK\_perform\_operatrimon, 435
- AK\_projection, 437
- AK\_temp\_create\_table, 437
- removeSubstring, 438
- projection.h
  - AK\_copy\_block\_projection, 440
  - AK\_create\_block\_header, 440
  - AK\_create\_header\_name, 441
  - AK\_determine\_header\_type, 441
  - AK\_get\_operator, 443
  - AK\_op\_projection\_test, 443
  - AK\_perform\_operatrimon, 443
  - AK\_projection, 445
  - AK\_temp\_create\_table, 445
  - removeSubstring, 446
- query\_optimization.c
  - AK\_execute\_rel\_eq, 350
  - AK\_print\_optimized\_query, 350
  - AK\_query\_optimization, 351
  - AK\_query\_optimization\_test, 351
- query\_optimization.h
  - AK\_execute\_rel\_eq, 353
  - AK\_print\_optimized\_query, 353
  - AK\_query\_optimization, 354
  - AK\_query\_optimization\_test, 354
- REF\_TYPE\_NO\_ACTION
  - reference.h, 490
- rec/archive\_log.h, 389
- rec/recovery.c, 391
- rec/redo\_log.c, 394
- recovery.c
  - AK\_recover\_archive\_log, 392
  - AK\_recover\_operation, 392
  - AK\_recovery\_insert\_row, 393
  - AK\_recovery\_test, 393
  - AK\_recovery\_tokenize, 393
  - grandfailure, 394
- redo\_log.c
  - AK\_add\_to\_redolog, 395
  - AK\_check\_attributes, 395
  - AK\_printout\_redolog, 396
- reference.c
  - AK\_add\_reference, 484
  - AK\_get\_reference, 484
  - AK\_reference\_check\_attribute, 485
  - AK\_reference\_check\_entry, 485
  - AK\_reference\_check\_if\_update\_needed, 486
  - AK\_reference\_check\_restricion, 486
  - AK\_reference\_test, 487
  - AK\_reference\_update, 487
- reference.h
  - AK\_add\_reference, 490
  - AK\_get\_reference, 491
- AK\_initialize\_new\_segment, 491
- AK\_reference\_check\_attribute, 494
- AK\_reference\_check\_entry, 494
- AK\_reference\_check\_if\_update\_needed, 495
- AK\_reference\_check\_restricion, 495
- AK\_reference\_test, 496
- AK\_reference\_update, 496
- AK\_selection, 496
- Ak\_Insert\_New\_Element, 492
- Ak\_Insert\_New\_Element\_For\_Update, 493
- Ak\_delete\_row, 491
- Ak\_insert\_row, 493
- Ak\_update\_row, 497
- REF\_TYPE\_NO\_ACTION, 490
- rel/aggregation.c, 396
- rel/aggregation.h, 402
- rel/difference.c, 407
- rel/difference.h, 408
- rel/expression\_check.c, 410
- rel/expression\_check.h, 414
- rel/intersect.c, 417
- rel/intersect.h, 419
- rel/nat\_join.c, 421
- rel/nat\_join.h, 424
- rel/product.c, 428
- rel/product.h, 430
- rel/projection.c, 432
- rel/projection.h, 438
- rel/selection.c, 446
- rel/selection.h, 449
- rel/theta\_join.c, 451
- rel/theta\_join.h, 454
- rel/union.c, 457
- rel/union.h, 458
- rel\_eq\_assoc.c
  - AK\_compare, 355
  - AK\_print\_rel\_eq\_assoc, 356
  - AK\_rel\_eq\_assoc, 356
  - AK\_rel\_eq\_assoc\_test, 357
- rel\_eq\_assoc.h
  - AK\_compare, 358
  - AK\_print\_rel\_eq\_assoc, 359
  - AK\_rel\_eq\_assoc, 359
  - AK\_rel\_eq\_assoc\_test, 360
- rel\_eq\_comut.c
  - AK\_print\_rel\_eq\_comut, 361
  - AK\_rel\_eq\_commute\_with\_theta\_join, 361
  - AK\_rel\_eq\_comut, 362
  - AK\_rel\_eq\_comut\_test, 362
- rel\_eq\_comut.h
  - AK\_print\_rel\_eq\_comut, 364
  - AK\_rel\_eq\_commute\_with\_theta\_join, 364
  - AK\_rel\_eq\_comut, 365
  - AK\_rel\_eq\_comut\_test, 365
- rel\_eq\_projection.c
  - AK\_print\_rel\_eq\_projection, 367
  - AK\_rel\_eq\_can\_commute, 367
  - AK\_rel\_eq\_collect\_cond\_attributes, 368

- AK\_rel\_eq\_get\_attributes, 368
- AK\_rel\_eq\_is\_subset, 369
- AK\_rel\_eq\_projection, 370
- AK\_rel\_eq\_projection\_attributes, 370
- AK\_rel\_eq\_projection\_test, 371
- AK\_rel\_eq\_remove\_duplicates, 371
- rel\_eq\_projection.h
  - AK\_print\_rel\_eq\_projection, 373
  - AK\_rel\_eq\_can\_commute, 373
  - AK\_rel\_eq\_collect\_cond\_attributes, 374
  - AK\_rel\_eq\_get\_attributes, 374
  - AK\_rel\_eq\_is\_subset, 375
  - AK\_rel\_eq\_projection, 376
  - AK\_rel\_eq\_projection\_attributes, 376
  - AK\_rel\_eq\_projection\_test, 377
  - AK\_rel\_eq\_remove\_duplicates, 377
- rel\_eq\_selection.c
  - AK\_print\_rel\_eq\_selection, 378
  - AK\_rel\_eq\_cond\_attributes, 379
  - AK\_rel\_eq\_get\_attributes\_char, 379
  - AK\_rel\_eq\_is\_attr\_subset, 380
  - AK\_rel\_eq\_selection, 381
  - AK\_rel\_eq\_selection\_test, 381
  - AK\_rel\_eq\_share\_attributes, 381
  - AK\_rel\_eq\_split\_condition, 382
- rel\_eq\_selection.h
  - AK\_print\_rel\_eq\_selection, 384
  - AK\_rel\_eq\_cond\_attributes, 385
  - AK\_rel\_eq\_get\_attributes\_char, 385
  - AK\_rel\_eq\_is\_attr\_subset, 386
  - AK\_rel\_eq\_selection, 386
  - AK\_rel\_eq\_selection\_test, 387
  - AK\_rel\_eq\_share\_attributes, 387
  - AK\_rel\_eq\_split\_condition, 388
- removeSubstring
  - projection.c, 438
  - projection.h, 446
- root\_info, 42
- search\_params, 42
- search\_result, 43
- select.c
  - AK\_select, 533
  - AK\_select\_test, 534
- selection.c
  - AK\_op\_selection\_test, 447
  - AK\_op\_selection\_test\_pattern, 447
  - AK\_op\_selection\_test\_redolog, 448
  - AK\_selection, 448
- selection.h
  - AK\_op\_selection\_test, 450
  - AK\_op\_selection\_test\_pattern, 450
  - AK\_op\_selection\_test\_redolog, 450
  - AK\_selection, 450
- selection\_test
  - test.c, 326
  - test.h, 331
- sequence.c
  - AK\_sequence\_add, 289
  - AK\_sequence\_current\_value, 290
  - AK\_sequence\_get\_id, 290
  - AK\_sequence\_modify, 290
  - AK\_sequence\_next\_value, 291
  - AK\_sequence\_remove, 292
  - AK\_sequence\_rename, 292
  - AK\_sequence\_test, 292
- sequence.h
  - AK\_sequence\_add, 294
  - AK\_sequence\_current\_value, 295
  - AK\_sequence\_get\_id, 295
  - AK\_sequence\_modify, 295
  - AK\_sequence\_next\_value, 296
  - AK\_sequence\_remove, 297
  - AK\_sequence\_rename, 297
  - AK\_sequence\_test, 297
- size
  - \_dictionary\_, 12
- sql/cs/between.c, 460
- sql/cs/between.h, 464
- sql/cs/check\_constraint.c, 467
- sql/cs/check\_constraint.h, 470
- sql/cs/constraint\_names.c, 474
- sql/cs/constraint\_names.h, 475
- sql/cs/nnnull.c, 477
- sql/cs/nnnull.h, 480
- sql/cs/reference.c, 483
- sql/cs/reference.h, 488
- sql/cs/unique.c, 497
- sql/cs/unique.h, 500
- sql/drop.c, 503
- sql/drop.h, 506
- sql/function.c, 509
- sql/function.h, 515
- sql/privileges.c, 521
- sql/select.c, 533
- sql/trigger.c, 534
- sql/trigger.h, 539
- sql/view.c, 545
- Stack, 43
- struct\_add, 44
- Succesor, 45
- system\_catalog
  - drop.c, 506
- table.c
  - AK\_check\_tables\_scheme, 299
  - AK\_get\_attr\_index, 300
  - AK\_get\_attr\_name, 300
  - AK\_get\_column, 301
  - AK\_get\_header, 301
  - AK\_get\_num\_records, 302
  - AK\_get\_row, 303
  - AK\_get\_table\_obj\_id, 303
  - AK\_get\_tuple, 303
  - AK\_num\_attr, 304
  - AK\_op\_rename\_test, 304
  - AK\_print\_row, 305
  - AK\_print\_row\_spacer, 305

- AK\_print\_row\_spacer\_to\_file, 306
- AK\_print\_row\_to\_file, 306
- AK\_print\_table, 307
- AK\_print\_table\_to\_file, 307
- AK\_rename, 308
- AK\_table\_empty, 308
- AK\_table\_exist, 309
- AK\_table\_test, 309
- AK\_tuple\_to\_string, 309
- get\_row\_attr\_data, 310
- table.h
  - AK\_check\_tables\_scheme, 313
  - AK\_get\_attr\_index, 313
  - AK\_get\_attr\_name, 314
  - AK\_get\_column, 314
  - AK\_get\_header, 315
  - AK\_get\_num\_records, 315
  - AK\_get\_row, 316
  - AK\_get\_table\_obj\_id, 316
  - AK\_get\_tuple, 317
  - AK\_num\_attr, 317
  - AK\_op\_rename\_test, 318
  - AK\_print\_row, 318
  - AK\_print\_row\_spacer, 318
  - AK\_print\_row\_spacer\_to\_file, 319
  - AK\_print\_row\_to\_file, 319
  - AK\_print\_table, 320
  - AK\_print\_table\_to\_file, 320
  - AK\_rename, 321
  - AK\_table\_empty, 321
  - AK\_table\_test, 322
  - AK\_tuple\_to\_string, 322
  - get\_row\_attr\_data, 322
- table\_addresses, 45
- test.c
  - AK\_create\_test\_tables, 324
  - AK\_get\_table\_attribute\_types, 324
  - create\_header\_test, 324
  - get\_column\_test, 325
  - get\_row\_test, 325
  - insert\_data\_test, 326
  - selection\_test, 326
- test.h
  - AK\_create\_test\_tables, 328
  - AK\_get\_table\_attribute\_types, 328
  - create\_header\_test, 329
  - get\_column\_test, 329
  - get\_row\_test, 330
  - insert\_data\_test, 330
  - selection\_test, 331
- testMode
  - auxiliary.h, 91
- theta\_join.c
  - AK\_check\_constraints, 452
  - AK\_create\_theta\_join\_header, 452
  - AK\_op\_theta\_join\_test, 453
  - AK\_theta\_join, 453
- theta\_join.h
  - AK\_check\_constraints, 455
  - AK\_create\_theta\_join\_header, 455
  - AK\_op\_theta\_join\_test, 456
  - AK\_theta\_join, 456
- threadContainer, 46
- tools/comments.py, 550
- tools/getFiles.sh, 550
- tools/parseC.sh, 551
- tools/parsePy.sh, 551
- tools/updateVersion.sh, 551
- trans/transaction.c, 551
- trans/transaction.h, 566
- transaction.c
  - AK\_acquire\_lock, 553
  - AK\_add\_hash\_entry\_list, 554
  - AK\_add\_lock, 555
  - AK\_all\_transactions\_finished, 555
  - AK\_create\_lock, 555
  - AK\_create\_new\_transaction\_thread, 556
  - AK\_delete\_hash\_entry\_list, 556
  - AK\_delete\_lock\_entry\_list, 557
  - AK\_execute\_commands, 557
  - AK\_execute\_transaction, 558
  - AK\_get\_memory\_blocks, 558
  - AK\_handle\_observable\_transaction\_action, 558
  - AK\_init\_observable\_transaction, 559
  - AK\_init\_observer\_lock, 559
  - AK\_isLock\_waiting, 559
  - AK\_lock\_released, 560
  - AK\_memory\_block\_hash, 560
  - AK\_on\_all\_transactions\_end, 561
  - AK\_on\_lock\_release, 561
  - AK\_on\_observable\_notify, 561
  - AK\_on\_transaction\_end, 562
  - AK\_release\_locks, 562
  - AK\_remove\_transaction\_thread, 562
  - AK\_search\_empty\_link\_for\_hook, 563
  - AK\_search\_existing\_link\_for\_hook, 563
  - AK\_search\_lock\_entry\_list\_by\_key, 564
  - AK\_transaction\_finished, 564
  - AK\_transaction\_manager, 564
  - AK\_transaction\_register\_observer, 565
  - AK\_transaction\_unregister\_observer, 565
  - handle\_transaction\_notify, 566
- transaction.h
  - AK\_acquire\_lock, 570
  - AK\_add\_hash\_entry\_list, 570
  - AK\_add\_lock, 571
  - AK\_all\_transactions\_finished, 571
  - AK\_create\_lock, 572
  - AK\_create\_new\_transaction\_thread, 572
  - AK\_delete\_hash\_entry\_list, 573
  - AK\_delete\_lock\_entry\_list, 573
  - AK\_execute\_commands, 574
  - AK\_execute\_transaction, 574
  - AK\_get\_memory\_blocks, 575
  - AK\_handle\_observable\_transaction\_action, 575
  - AK\_init\_observable\_transaction, 575

- AK\_init\_observer\_lock, 576
- AK\_isLock\_waiting, 576
- AK\_lock\_released, 577
- AK\_memory\_block\_hash, 577
- AK\_on\_all\_transactions\_end, 577
- AK\_on\_lock\_release, 578
- AK\_on\_observable\_notify, 578
- AK\_on\_transaction\_end, 578
- AK\_release\_locks, 579
- AK\_remove\_transaction\_thread, 579
- AK\_search\_empty\_link\_for\_hook, 579
- AK\_search\_existing\_link\_for\_hook, 580
- AK\_search\_lock\_entry\_list\_by\_key, 580
- AK\_transaction\_finished, 581
- AK\_transaction\_manager, 581
- AK\_transaction\_register\_observer, 581
- AK\_transaction\_unregister\_observer, 582
- handle\_transaction\_notify, 582
- NoticeType, 569
- transaction\_list\_elem, 47
- transaction\_list\_head, 48
- transaction\_locks\_list\_elem, 49
- transactionData, 49
- trigger.c
  - AK\_trigger\_add, 535
  - AK\_trigger\_edit, 536
  - AK\_trigger\_get\_conditions, 536
  - AK\_trigger\_get\_id, 537
  - AK\_trigger\_remove\_by\_name, 537
  - AK\_trigger\_remove\_by\_obj\_id, 538
  - AK\_trigger\_rename, 538
  - AK\_trigger\_save\_conditions, 539
  - AK\_trigger\_test, 539
- trigger.h
  - AK\_trigger\_add, 541
  - AK\_trigger\_edit, 541
  - AK\_trigger\_get\_conditions, 542
  - AK\_trigger\_get\_id, 542
  - AK\_trigger\_remove\_by\_name, 543
  - AK\_trigger\_remove\_by\_obj\_id, 543
  - AK\_trigger\_rename, 544
  - AK\_trigger\_save\_conditions, 544
  - AK\_trigger\_test, 545
- TypeObservable, 51
- TypeObserver, 52
- union.c
  - AK\_op\_union\_test, 457
  - AK\_union, 458
- union.h
  - AK\_op\_union\_test, 459
  - AK\_union, 460
- unique.c
  - AK\_delete\_constraint\_unique, 498
  - AK\_read\_constraint\_unique, 499
  - AK\_unique\_test, 500
  - Ak\_set\_constraint\_unique, 499
- unique.h
  - AK\_delete\_constraint\_unique, 501
  - AK\_read\_constraint\_unique, 502
  - AK\_unique\_test, 503
  - Ak\_set\_constraint\_unique, 502
- val
  - \_dictionary\_, 12
- Vertex, 52
- view.c
  - AK\_get\_rel\_exp, 546
  - AK\_get\_view\_obj\_id, 546
  - AK\_get\_view\_query, 547
  - AK\_view\_add, 547
  - AK\_view\_change\_query, 548
  - AK\_view\_remove\_by\_name, 548
  - AK\_view\_remove\_by\_obj\_id, 549
  - AK\_view\_rename, 549
  - AK\_view\_test, 549