# Kalashnikov DB

0.9.3

Generated by Doxygen 1.8.17

**1 Todo List** 1

**2 Namespace Index** 3

2.1 Namespace List . . . 3

**3 Class Index** 5

3.1 Class List . . . 5

**4 File Index** 9

4.1 File List . . . 9

**5 Namespace Documentation** 13

5.1 comments Namespace Reference . . . 13

5.1.1 Function Documentation . . . 13

5.1.1.1 detectLanguage() . . . 13

5.1.1.2 getcommentsFiles() . . . 14

5.1.1.3 makeCommentsFile() . . . 14

5.1.2 Variable Documentation . . . 14

5.1.2.1 cFiles . . . 14

5.1.2.2 commentsFile . . . 14

5.1.2.3 pyFiles . . . 14

**6 Class Documentation** 15

6.1 _dictionary_ Struct Reference . . . 15

6.1.1 Detailed Description . . . 15

6.1.2 Member Data Documentation . . . 15

6.1.2.1 hash . . . 15

6.1.2.2 key . . . 16

6.1.2.3 n . . . 16

6.1.2.4 size . . . 16

6.1.2.5 val . . . 16

6.2 _file_metadata Struct Reference . . . 16

6.2.1 Member Data Documentation . . . 16

6.2.1.1 checksum . . . 17

6.2.1.2 new_name . . . 17

6.2.1.3 new_path . . . 17

6.2.1.4 old_name . . . 17

6.2.1.5 old_path . . . 17

6.3 _notifyDetails Struct Reference . . . 17

6.3.1 Member Data Documentation . . . 17

6.3.1.1 message . . . 18

6.3.1.2 type . . . 18

6.4 AK_agg_input Struct Reference . . . 18

6.4.1 Detailed Description . . . 18

6.4.2 Member Data Documentation . . . 18
6.4.2.1 attributes . . . 18
6.4.2.2 counter . . . 19
6.4.2.3 tasks . . . 19
6.5 AK_agg_value Struct Reference . . . 19
6.5.1 Detailed Description . . . 19
6.5.2 Member Data Documentation . . . 19
6.5.2.1 agg_task . . . 19
6.5.2.2 att_name . . . 20
6.5.2.3 data . . . 20
6.6 AK_block Struct Reference . . . 20
6.6.1 Detailed Description . . . 20
6.6.2 Member Data Documentation . . . 21
6.6.2.1 address . . . 21
6.6.2.2 AK_free_space . . . 21
6.6.2.3 chained_with . . . 21
6.6.2.4 data . . . 21
6.6.2.5 header . . . 21
6.6.2.6 last_tuple_dict_id . . . 21
6.6.2.7 tuple_dict . . . 22
6.6.2.8 type . . . 22
6.7 AK_block_activity Struct Reference . . . 22
6.7.1 Detailed Description . . . 23
6.7.2 Member Data Documentation . . . 23
6.7.2.1 block_lock . . . 23
6.7.2.2 locked_for_reading . . . 23
6.7.2.3 locked_for_writing . . . 23
6.7.2.4 reading_done . . . 23
6.7.2.5 thread_holding_lock . . . 24
6.7.2.6 writing_done . . . 24
6.8 AK_blocktable Struct Reference . . . 24
6.8.1 Member Data Documentation . . . 24
6.8.1.1 allocationtable . . . 24
6.8.1.2 bittable . . . 24
6.8.1.3 last_allocated . . . 25
6.8.1.4 last_initialized . . . 25
6.8.1.5 ltime . . . 25
6.8.1.6 prepared . . . 25
6.9 AK_command_recovery_struct Struct Reference . . . 25
6.9.1 Detailed Description . . . 25
6.9.2 Member Data Documentation . . . 26
6.9.2.1 arguments . . . 26

6.9.2.2 condition . . . 26
6.9.2.3 finished . . . 26
6.9.2.4 operation . . . 26
6.9.2.5 table_name . . . 26
6.10 AK_command_struct Struct Reference . . . 26
6.10.1 Member Data Documentation . . . 27
6.10.1.1 id_command . . . 27
6.10.1.2 parameters . . . 27
6.10.1.3 tblName . . . 27
6.11 AK_create_table_struct Struct Reference . . . 27
6.11.1 Member Data Documentation . . . 27
6.11.1.1 name . . . 27
6.11.1.2 type . . . 28
6.12 AK_db_cache Struct Reference . . . 28
6.12.1 Detailed Description . . . 28
6.12.2 Member Data Documentation . . . 28
6.12.2.1 cache . . . 28
6.12.2.2 next_replace . . . 29
6.13 AK_debmod_state Struct Reference . . . 29
6.13.1 Detailed Description . . . 29
6.13.2 Member Data Documentation . . . 29
6.13.2.1 alloc_owner . . . 30
6.13.2.2 dirty . . . 30
6.13.2.3 free_owner . . . 30
6.13.2.4 fstack_items . . . 30
6.13.2.5 fstack_size . . . 30
6.13.2.6 func_used_by . . . 30
6.13.2.7 function . . . 30
6.13.2.8 init . . . 30
6.13.2.9 last_function_id . . . 31
6.13.2.10 nomi . . . 31
6.13.2.11 page . . . 31
6.13.2.12 page_size . . . 31
6.13.2.13 print . . . 31
6.13.2.14 ready . . . 31
6.13.2.15 real . . . 31
6.13.2.16 used . . . 32
6.14 AK_header Struct Reference . . . 32
6.14.1 Detailed Description . . . 32
6.14.2 Member Data Documentation . . . 32
6.14.2.1 att_name . . . 32
6.14.2.2 constr_code . . . 33

6.14.2.3 constr_name . . . 33
6.14.2.4 integrity . . . 33
6.14.2.5 type . . . 33
6.15 AK_mem_block Struct Reference . . . 33
6.15.1 Detailed Description . . . 34
6.15.2 Member Data Documentation . . . 34
6.15.2.1 block . . . 34
6.15.2.2 dirty . . . 34
6.15.2.3 timestamp_last_change . . . 34
6.15.2.4 timestamp_read . . . 35
6.16 AK_operand Struct Reference . . . 35
6.16.1 Member Data Documentation . . . 35
6.16.1.1 type . . . 35
6.16.1.2 value . . . 35
6.17 AK_query_mem Struct Reference . . . 35
6.17.1 Detailed Description . . . 36
6.17.2 Member Data Documentation . . . 36
6.17.2.1 dictionary . . . 36
6.17.2.2 parsed . . . 36
6.17.2.3 result . . . 36
6.18 AK_query_mem_dict Struct Reference . . . 37
6.18.1 Detailed Description . . . 37
6.18.2 Member Data Documentation . . . 37
6.18.2.1 dictionary . . . 37
6.18.2.2 next_replace . . . 37
6.19 AK_query_mem_lib Struct Reference . . . 38
6.19.1 Detailed Description . . . 38
6.19.2 Member Data Documentation . . . 38
6.19.2.1 next_replace . . . 38
6.19.2.2 parsed . . . 38
6.20 AK_query_mem_result Struct Reference . . . 39
6.20.1 Detailed Description . . . 39
6.20.2 Member Data Documentation . . . 39
6.20.2.1 next_replace . . . 39
6.20.2.2 results . . . 39
6.21 AK_redo_log Struct Reference . . . 40
6.21.1 Detailed Description . . . 40
6.21.2 Member Data Documentation . . . 40
6.21.2.1 command_recovery . . . 40
6.21.2.2 number . . . 40
6.22 AK_ref_item Struct Reference . . . 41
6.22.1 Detailed Description . . . 41

6.22.2 Member Data Documentation . . . 41
6.22.2.1 attributes . . . 41
6.22.2.2 attributes_number . . . 41
6.22.2.3 constraint . . . 41
6.22.2.4 parent . . . 42
6.22.2.5 parent_attributes . . . 42
6.22.2.6 table . . . 42
6.22.2.7 type . . . 42
6.23 AK_results Struct Reference . . . 42
6.23.1 Detailed Description . . . 43
6.23.2 Member Data Documentation . . . 43
6.23.2.1 date_created . . . 43
6.23.2.2 free . . . 43
6.23.2.3 header . . . 43
6.23.2.4 result_block . . . 43
6.23.2.5 result_id . . . 43
6.23.2.6 result_size . . . 43
6.23.2.7 source_table . . . 44
6.24 AK_synchronization_info Struct Reference . . . 44
6.24.1 Detailed Description . . . 44
6.24.2 Member Data Documentation . . . 44
6.24.2.1 init . . . 44
6.24.2.2 ready . . . 44
6.25 AK_tuple_dict Struct Reference . . . 45
6.25.1 Detailed Description . . . 45
6.25.2 Member Data Documentation . . . 45
6.25.2.1 address . . . 45
6.25.2.2 size . . . 45
6.25.2.3 type . . . 46
6.26 blocktable Struct Reference . . . 46
6.26.1 Detailed Description . . . 46
6.27 btree_node Struct Reference . . . 46
6.27.1 Member Data Documentation . . . 46
6.27.1.1 pointers . . . 47
6.27.1.2 values . . . 47
6.28 bucket_elem Struct Reference . . . 47
6.28.1 Detailed Description . . . 47
6.28.2 Member Data Documentation . . . 47
6.28.2.1 add . . . 47
6.28.2.2 value . . . 48
6.29 cost_eval_t Struct Reference . . . 48
6.29.1 Detailed Description . . . 48

6.29.2 Member Data Documentation . . . 48
6.29.2.1 data . . . 48
6.29.2.2 value . . . 48
6.30 DEBUG_LEVEL Struct Reference . . . 49
6.30.1 Detailed Description . . . 49
6.31 DEBUG_TYPE Struct Reference . . . 49
6.31.1 Detailed Description . . . 49
6.32 drop_arguments Struct Reference . . . 50
6.32.1 Member Data Documentation . . . 50
6.32.1.1 next . . . 50
6.32.1.2 value . . . 50
6.33 hash_bucket Struct Reference . . . 50
6.33.1 Detailed Description . . . 51
6.33.2 Member Data Documentation . . . 51
6.33.2.1 bucket_level . . . 51
6.33.2.2 element . . . 51
6.34 hash_info Struct Reference . . . 51
6.34.1 Detailed Description . . . 52
6.34.2 Member Data Documentation . . . 52
6.34.2.1 hash_bucket_num . . . 52
6.34.2.2 main_bucket_num . . . 52
6.34.2.3 modulo . . . 52
6.35 intersect_attr Struct Reference . . . 52
6.35.1 Detailed Description . . . 53
6.35.2 Member Data Documentation . . . 53
6.35.2.1 att_name . . . 53
6.35.2.2 type . . . 53
6.36 list_node Struct Reference . . . 53
6.36.1 Detailed Description . . . 54
6.36.2 Member Data Documentation . . . 54
6.36.2.1 attribute_name . . . 54
6.36.2.2 constraint . . . 54
6.36.2.3 data . . . 54
6.36.2.4 next . . . 55
6.36.2.5 size . . . 55
6.36.2.6 table . . . 55
6.36.2.7 type . . . 55
6.37 list_structure_ad Struct Reference . . . 55
6.37.1 Member Data Documentation . . . 56
6.37.1.1 add . . . 56
6.37.1.2 attName . . . 56
6.37.1.3 next . . . 56

6.38 list_structure_add Struct Reference . . . 56
6.38.1 Detailed Description . . . 56
6.39 main_bucket Struct Reference . . . 57
6.39.1 Detailed Description . . . 57
6.39.2 Member Data Documentation . . . 57
6.39.2.1 element . . . 57
6.40 memoryAddresses Struct Reference . . . 57
6.40.1 Detailed Description . . . 58
6.40.2 Member Data Documentation . . . 58
6.40.2.1 adresa . . . 58
6.40.2.2 nextElement . . . 58
6.41 Observable Struct Reference . . . 58
6.41.1 Detailed Description . . . 59
6.41.2 Member Data Documentation . . . 59
6.41.2.1 AK_destroy_observable . . . 59
6.41.2.2 AK_get_observer_by_id . . . 59
6.41.2.3 AK_notify_observer . . . 59
6.41.2.4 AK_notify_observers . . . 59
6.41.2.5 AK_observable_type . . . 59
6.41.2.6 AK_ObservableType_Def . . . 59
6.41.2.7 AK_register_observer . . . 60
6.41.2.8 AK_run_custom_action . . . 60
6.41.2.9 AK_unregister_observer . . . 60
6.41.2.10 observer_id_counter . . . 60
6.41.2.11 observers . . . 60
6.42 observable_transaction Struct Reference . . . 60
6.42.1 Detailed Description . . . 60
6.43 observable_transaction_struct Struct Reference . . . 61
6.43.1 Member Data Documentation . . . 61
6.43.1.1 AK_all_transactions_finished . . . 61
6.43.1.2 AK_lock_released . . . 61
6.43.1.3 AK_transaction_finished . . . 61
6.43.1.4 AK_transaction_register_observer . . . 61
6.43.1.5 AK_transaction_unregister_observer . . . 62
6.43.1.6 observable . . . 62
6.44 Observer Struct Reference . . . 62
6.44.1 Detailed Description . . . 62
6.44.2 Member Data Documentation . . . 62
6.44.2.1 AK_destroy_observer . . . 62
6.44.2.2 AK_notify . . . 63
6.44.2.3 AK_observer_type . . . 63
6.44.2.4 AK_observer_type_event_handler . . . 63

6.44.2.5 observer_id . . . 63
6.45 observer_lock Struct Reference . . . 63
6.45.1 Detailed Description . . . 63
6.45.2 Member Data Documentation . . . 64
6.45.2.1 observer . . . 64
6.46 projection_att_struct Struct Reference . . . 64
6.46.1 Detailed Description . . . 64
6.46.2 Member Data Documentation . . . 64
6.46.2.1 projection_att . . . 64
6.47 root_info Struct Reference . . . 65
6.47.1 Member Data Documentation . . . 65
6.47.1.1 level . . . 65
6.47.1.2 root . . . 65
6.48 rowroot_struct Struct Reference . . . 65
6.48.1 Detailed Description . . . 65
6.48.2 Member Data Documentation . . . 66
6.48.2.1 row_root . . . 66
6.49 search_params Struct Reference . . . 66
6.49.1 Detailed Description . . . 66
6.49.2 Member Data Documentation . . . 66
6.49.2.1 iSearchType . . . 67
6.49.2.2 pData_lower . . . 67
6.49.2.3 pData_upper . . . 67
6.49.2.4 szAttribute . . . 67
6.50 search_result Struct Reference . . . 67
6.50.1 Detailed Description . . . 68
6.50.2 Member Data Documentation . . . 68
6.50.2.1 aiBlocks . . . 68
6.50.2.2 aiSearch_attributes . . . 68
6.50.2.3 aiTuple_addresses . . . 68
6.50.2.4 iNum_search_attributes . . . 69
6.50.2.5 iNum_tuple_addresses . . . 69
6.50.2.6 iNum_tuple_attributes . . . 69
6.51 Stack Struct Reference . . . 69
6.51.1 Detailed Description . . . 69
6.51.2 Member Data Documentation . . . 70
6.51.2.1 link . . . 70
6.51.2.2 nextElement . . . 70
6.52 struct_add Struct Reference . . . 70
6.52.1 Detailed Description . . . 70
6.52.2 Member Data Documentation . . . 70
6.52.2.1 addBlock . . . 71

6.52.2.2 indexTd . . . 71
6.53 Succesor Struct Reference . . . 71
6.53.1 Detailed Description . . . 71
6.53.2 Member Data Documentation . . . 71
6.53.2.1 link . . . 72
6.53.2.2 nextSuccesor . . . 72
6.54 table_addresses Struct Reference . . . 72
6.54.1 Detailed Description . . . 72
6.54.2 Member Data Documentation . . . 72
6.54.2.1 address_from . . . 72
6.54.2.2 address_to . . . 73
6.55 TestResult Struct Reference . . . 73
6.55.1 Detailed Description . . . 73
6.55.2 Member Data Documentation . . . 73
6.55.2.1 implemented . . . 73
6.55.2.2 testFailed . . . 73
6.55.2.3 testSucceded . . . 74
6.56 threadContainer Struct Reference . . . 74
6.56.1 Detailed Description . . . 74
6.56.2 Member Data Documentation . . . 74
6.56.2.1 nextThread . . . 74
6.56.2.2 thread . . . 75
6.57 transaction_list_elem Struct Reference . . . 75
6.57.1 Detailed Description . . . 75
6.57.2 Member Data Documentation . . . 75
6.57.2.1 address . . . 75
6.57.2.2 DLLLocksHead . . . 76
6.57.2.3 isWaiting . . . 76
6.57.2.4 lock_type . . . 76
6.57.2.5 nextBucket . . . 76
6.57.2.6 observer_lock . . . 76
6.57.2.7 prevBucket . . . 76
6.58 transaction_list_head Struct Reference . . . 76
6.58.1 Detailed Description . . . 77
6.58.2 Member Data Documentation . . . 77
6.58.2.1 DLLHead . . . 77
6.59 transaction_locks_list_elem Struct Reference . . . 77
6.59.1 Detailed Description . . . 77
6.59.2 Member Data Documentation . . . 78
6.59.2.1 isWaiting . . . 78
6.59.2.2 lock_type . . . 78
6.59.2.3 nextLock . . . 78

6.59.2.4 prevLock . . . 78
6.59.2.5 TransactionId . . . 78
6.60 transactionData Struct Reference . . . 78
6.60.1 Detailed Description . . . 79
6.60.2 Member Data Documentation . . . 79
6.60.2.1 array . . . 79
6.60.2.2 lengthOfArray . . . 79
6.61 TypeObservable Struct Reference . . . 79
6.61.1 Member Data Documentation . . . 80
6.61.1.1 AK_custom_register_observer . . . 80
6.61.1.2 AK_custom_unregister_observer . . . 80
6.61.1.3 AK_get_message . . . 80
6.61.1.4 AK_set_notify_info_details . . . 80
6.61.1.5 notifyDetails . . . 80
6.61.1.6 observable . . . 80
6.62 TypeObserver Struct Reference . . . 81
6.62.1 Member Data Documentation . . . 81
6.62.1.1 observable . . . 81
6.62.1.2 observer . . . 81
6.63 Vertex Struct Reference . . . 81
6.63.1 Detailed Description . . . 82
6.63.2 Member Data Documentation . . . 82
6.63.2.1 index . . . 82
6.63.2.2 lowLink . . . 82
6.63.2.3 nextSuccesor . . . 82
6.63.2.4 nextVertex . . . 82
6.63.2.5 vertexId . . . 82

**7 File Documentation** **83**

7.1 auxi/auxiliary.c File Reference . . . 83
7.2 auxi/auxiliary.h File Reference . . . 83
7.2.1 Detailed Description . . . 85
7.2.2 Macro Definition Documentation . . . 86
7.2.2.1 MAX_LOOP_ITERATIONS . . . 86
7.2.2.2 TBL_BOX_OFFSET . . . 86
7.2.3 Typedef Documentation . . . 86
7.2.3.1 AK_graph . . . 86
7.2.3.2 AK_list . . . 86
7.2.3.3 AK_list_elem . . . 86
7.2.3.4 AK_stack . . . 86
7.2.3.5 AK_stackHead . . . 87
7.2.3.6 AK_succesor . . . 87

7.2.3.7 AK_vertex . . . 87
7.2.4 Function Documentation . . . 87
7.2.4.1 AK_add_succesor() . . . 87
7.2.4.2 AK_add_vertex() . . . 87
7.2.4.3 AK_chars_num_from_number() . . . 88
7.2.4.4 AK_convert_type() . . . 88
7.2.4.5 AK_define_tarjan_graph() . . . 89
7.2.4.6 AK_Delete_L3() . . . 89
7.2.4.7 AK_DeleteAll_L3() . . . 90
7.2.4.8 AK_destroy_critical_section() . . . 90
7.2.4.9 AK_End_L2() . . . 91
7.2.4.10 AK_enter_critical_section() . . . 91
7.2.4.11 AK_First_L2() . . . 92
7.2.4.12 AK_get_array_perms() . . . 92
7.2.4.13 AK_GetNth_L2() . . . 93
7.2.4.14 AK_init_critical_section() . . . 94
7.2.4.15 AK_Init_L3() . . . 95
7.2.4.16 AK_InsertAfter_L2() . . . 95
7.2.4.17 AK_InsertAtBegin_L3() . . . 96
7.2.4.18 AK_InsertAtEnd_L3() . . . 96
7.2.4.19 AK_InsertBefore_L2() . . . 97
7.2.4.20 AK_IsEmpty_L2() . . . 97
7.2.4.21 AK_leave_critical_section() . . . 98
7.2.4.22 AK_Next_L2() . . . 98
7.2.4.23 AK_pop_from_stack() . . . 99
7.2.4.24 AK_Previous_L2() . . . 99
7.2.4.25 AK_push_to_stack() . . . 99
7.2.4.26 AK_Retrieve_L2() . . . 100
7.2.4.27 AK_search_empty_link() . . . 100
7.2.4.28 AK_search_empty_stack_link() . . . 101
7.2.4.29 AK_search_in_stack() . . . 101
7.2.4.30 AK_search_vertex() . . . 102
7.2.4.31 AK_Size_L2() . . . 102
7.2.4.32 AK_strcmp() . . . 102
7.2.4.33 AK_tarjan() . . . 103
7.2.4.34 AK_tarjan_test() . . . 103
7.2.4.35 AK_type_size() . . . 104
7.2.4.36 MIN() . . . 104
7.2.5 Variable Documentation . . . 104
7.2.5.1 testMode . . . 105
7.3 auxi/configuration.h File Reference . . . 105
7.3.1 Macro Definition Documentation . . . 106

7.3.1.1 AK_BLOBS_PATH . . . 106
7.3.1.2 ARCHIVELOG_PATH . . . 106
7.3.1.3 DB_FILE . . . 106
7.3.1.4 DB_FILE_BLOCKS_NUM . . . 106
7.3.1.5 DB_FILE_SIZE . . . 106
7.3.1.6 EXTENT_GROWTH_INDEX . . . 106
7.3.1.7 EXTENT_GROWTH_TABLE . . . 107
7.3.1.8 EXTENT_GROWTH_TEMP . . . 107
7.3.1.9 EXTENT_GROWTH_TRANSACTION . . . 107
7.3.1.10 INITIAL_EXTENT_SIZE . . . 107
7.3.1.11 MAX_EXTENTS_IN_SEGMENT . . . 107
7.3.1.12 MAX_FREE_SPACE_SIZE . . . 107
7.3.1.13 MAX_LAST_TUPLE_DICT_SIZE_TO_USE . . . 108
7.3.1.14 MAX_NUM_OF_BLOCKS . . . 108
7.3.1.15 MAX_REDO_LOG_ENTRIES . . . 108
7.3.1.16 MAX_REDO_LOG_MEMORY . . . 108
7.3.1.17 NUMBER_OF_THREADS . . . 108
7.4 auxi/constants.h File Reference . . . 108
7.4.1 Detailed Description . . . 112
7.4.2 Macro Definition Documentation . . . 113
7.4.2.1 ABORT . . . 113
7.4.2.2 AK_CONSTRAINTS_BEWTEEN . . . 113
7.4.2.3 AK_CONSTRAINTS_CHECK_CONSTRAINT . . . 113
7.4.2.4 AK_CONSTRAINTS_DEFAULT . . . 113
7.4.2.5 AK_CONSTRAINTS_FOREIGN_KEY . . . 113
7.4.2.6 AK_CONSTRAINTS_INDEX . . . 114
7.4.2.7 AK_CONSTRAINTS_NOT_NULL . . . 114
7.4.2.8 AK_CONSTRAINTS_PRIMARY_KEY . . . 114
7.4.2.9 AK_CONSTRAINTS_UNIQUE . . . 114
7.4.2.10 AK_REFERENCE . . . 114
7.4.2.11 ATTR_DELIMITER . . . 115
7.4.2.12 ATTR_ESCAPE . . . 115
7.4.2.13 BLOCK_CLEAN . . . 115
7.4.2.14 BLOCK_DIRTY . . . 115
7.4.2.15 BLOCK_TYPE_CHAINED . . . 115
7.4.2.16 BLOCK_TYPE_FREE . . . 115
7.4.2.17 BLOCK_TYPE_NORMAL . . . 116
7.4.2.18 COMMIT . . . 116
7.4.2.19 DATA_BLOCK_SIZE . . . 116
7.4.2.20 DATA_ENTRY_SIZE . . . 116
7.4.2.21 DELETE . . . 116
7.4.2.22 DROP_CONSTRAINT . . . 116

7.4.2.23 DROP_FUNCTION . . . 117
7.4.2.24 DROP_GROUP . . . 117
7.4.2.25 DROP_INDEX . . . 117
7.4.2.26 DROP_SEQUENCE . . . 117
7.4.2.27 DROP_TABLE . . . 117
7.4.2.28 DROP_TRIGGER . . . 117
7.4.2.29 DROP_USER . . . 118
7.4.2.30 DROP_VIEW . . . 118
7.4.2.31 EXCLUSIVE_LOCK . . . 118
7.4.2.32 EXIT_ERROR . . . 118
7.4.2.33 EXIT_SUCCESS . . . 118
7.4.2.34 EXIT_WARNING . . . 118
7.4.2.35 FIND . . . 119
7.4.2.36 FREE_CHAR . . . 119
7.4.2.37 FREE_INT . . . 119
7.4.2.38 HASH_BUCKET . . . 119
7.4.2.39 HASH_BUCKET_SIZE . . . 119
7.4.2.40 INFO_BUCKET . . . 119
7.4.2.41 INSERT . . . 120
7.4.2.42 MAIN_BUCKET . . . 120
7.4.2.43 MAIN_BUCKET_SIZE . . . 120
7.4.2.44 MAX_ACTIVE_TRANSACTIONS_COUNT . . . 120
7.4.2.45 MAX_ATT_NAME . . . 120
7.4.2.46 MAX_ATTRIBUTES . . . 120
7.4.2.47 MAX_BLOCKS_CURRENTLY_ACCESSED . . . 121
7.4.2.48 MAX_CACHE_MEMORY . . . 121
7.4.2.49 MAX_CONSTR_CODE . . . 121
7.4.2.50 MAX_CONSTR_NAME . . . 121
7.4.2.51 MAX_CONSTRAINTS . . . 121
7.4.2.52 MAX_MAIN_BUCKETS . . . 121
7.4.2.53 MAX_OBSERVABLE_OBSERVERS . . . 122
7.4.2.54 MAX_QUERY_DICT_MEMORY . . . 122
7.4.2.55 MAX_QUERY_LIB_MEMORY . . . 122
7.4.2.56 MAX_QUERY_RESULT_MEMORY . . . 122
7.4.2.57 MAX_TOKENS . . . 122
7.4.2.58 MAX_VARCHAR_LENGTH . . . 122
7.4.2.59 NEW_ID . . . 123
7.4.2.60 NEW_VALUE . . . 123
7.4.2.61 NOT_CHAINED . . . 123
7.4.2.62 NOT_OK . . . 123
7.4.2.63 NULLL . . . 123
7.4.2.64 NUM_SYS_TABLES . . . 123

7.4.2.65 NUMBER_OF_KEYS . . . 124
7.4.2.66 OK . . . 124
7.4.2.67 PASS_LOCK_QUEUE . . . 124
7.4.2.68 RO_EXCEPT . . . 124
7.4.2.69 RO_INTERSECT . . . 124
7.4.2.70 RO_NAT_JOIN . . . 124
7.4.2.71 RO_PROJECTION . . . 124
7.4.2.72 RO_RENAME . . . 125
7.4.2.73 RO_SELECTION . . . 125
7.4.2.74 RO_THETA_JOIN . . . 125
7.4.2.75 RO_UNION . . . 125
7.4.2.76 SEARCH_CONSTRAINT . . . 125
7.4.2.77 SEGMENT_TYPE_INDEX . . . 125
7.4.2.78 SEGMENT_TYPE_SYSTEM_TABLE . . . 125
7.4.2.79 SEGMENT_TYPE_TABLE . . . 126
7.4.2.80 SEGMENT_TYPE_TEMP . . . 126
7.4.2.81 SEGMENT_TYPE_TRANSACTION . . . 126
7.4.2.82 SELECT . . . 126
7.4.2.83 SEPARATOR . . . 126
7.4.2.84 SHARED_LOCK . . . 126
7.4.2.85 TEST_MODE_OFF . . . 127
7.4.2.86 TEST_MODE_ON . . . 127
7.4.2.87 TYPE_ATTRIBS . . . 127
7.4.2.88 TYPE_BLOB . . . 127
7.4.2.89 TYPE_BOOL . . . 127
7.4.2.90 TYPE_CONDITION . . . 127
7.4.2.91 TYPE_DATE . . . 128
7.4.2.92 TYPE_DATETIME . . . 128
7.4.2.93 TYPE_FLOAT . . . 128
7.4.2.94 TYPE_INT . . . 128
7.4.2.95 TYPE_INTERNAL . . . 128
7.4.2.96 TYPE_NUMBER . . . 128
7.4.2.97 TYPE_OPERAND . . . 129
7.4.2.98 TYPE_OPERATOR . . . 129
7.4.2.99 TYPE_TIME . . . 129
7.4.2.100 TYPE_VARCHAR . . . 129
7.4.2.101 UPDATE . . . 129
7.4.2.102 WAIT_FOR_UNLOCK . . . 129
7.5 auxi/debug.c File Reference . . . 130
7.5.1 Detailed Description . . . 130
7.5.2 Function Documentation . . . 130
7.5.2.1 AK_dbg_messg() . . . 130

7.6 auxi/debug.h File Reference . . . 131
7.6.1 Detailed Description . . . 131
7.6.2 Macro Definition Documentation . . . 131
7.6.2.1 DEBUG_ALL . . . 132
7.6.3 Typedef Documentation . . . 132
7.6.3.1 DEBUG_LEVEL . . . 132
7.6.3.2 DEBUG_TYPE . . . 132
7.6.4 Enumeration Type Documentation . . . 132
7.6.4.1 debug_level . . . 132
7.6.4.2 debug_type . . . 132
7.6.5 Function Documentation . . . 133
7.6.5.1 AK_dbg_messg() . . . 133
7.7 auxi/dictionary.c File Reference . . . 134
7.7.1 Detailed Description . . . 134
7.7.2 Macro Definition Documentation . . . 135
7.7.2.1 DICT_INVALID_KEY . . . 135
7.7.2.2 DICTMINSZ . . . 135
7.7.2.3 MAXVALSZ . . . 135
7.7.3 Function Documentation . . . 135
7.7.3.1 dictionary_del() . . . 135
7.7.3.2 dictionary_dump() . . . 136
7.7.3.3 dictionary_get() . . . 136
7.7.3.4 dictionary_hash() . . . 136
7.7.3.5 dictionary_new() . . . 137
7.7.3.6 dictionary_set() . . . 137
7.7.3.7 dictionary_unset() . . . 138
7.8 auxi/dictionary.h File Reference . . . 138
7.8.1 Detailed Description . . . 139
7.8.2 Typedef Documentation . . . 139
7.8.2.1 dictionary . . . 139
7.8.3 Function Documentation . . . 139
7.8.3.1 dictionary_del() . . . 139
7.8.3.2 dictionary_dump() . . . 140
7.8.3.3 dictionary_get() . . . 140
7.8.3.4 dictionary_hash() . . . 141
7.8.3.5 dictionary_new() . . . 141
7.8.3.6 dictionary_set() . . . 141
7.8.3.7 dictionary_unset() . . . 142
7.9 auxi/iniparser.c File Reference . . . 142
7.9.1 Detailed Description . . . 144
7.9.2 Macro Definition Documentation . . . 144
7.9.2.1 ASCIILINESZ . . . 144

7.9.2.2 INI_INVALID_KEY . . . 144
7.9.3 Typedef Documentation . . . 144
7.9.3.1 line_status . . . 144
7.9.4 Enumeration Type Documentation . . . 144
7.9.4.1 _line_status_ . . . 144
7.9.5 Function Documentation . . . 145
7.9.5.1 AK_inflate_config() . . . 145
7.9.5.2 iniparser_AK_freedict() . . . 145
7.9.5.3 iniparser_dump() . . . 145
7.9.5.4 iniparser_dump_ini() . . . 146
7.9.5.5 iniparser_dumpsection_ini() . . . 146
7.9.5.6 iniparser_find_entry() . . . 147
7.9.5.7 iniparser_getboolean() . . . 147
7.9.5.8 iniparser_getdouble() . . . 148
7.9.5.9 iniparser_getint() . . . 148
7.9.5.10 iniparser_getnsec() . . . 149
7.9.5.11 iniparser_getseckeys() . . . 149
7.9.5.12 iniparser_getsecname() . . . 150
7.9.5.13 iniparser_getsecnkeys() . . . 150
7.9.5.14 iniparser_getstring() . . . 151
7.9.5.15 iniparser_load() . . . 151
7.9.5.16 iniparser_set() . . . 152
7.9.5.17 iniparser_unset() . . . 152
7.9.6 Variable Documentation . . . 152
7.9.6.1 AK_config . . . 152
7.9.6.2 iniParserMutex . . . 153
7.10 auxi/iniparser.h File Reference . . . 153
7.10.1 Detailed Description . . . 154
7.10.2 Function Documentation . . . 154
7.10.2.1 AK_inflate_config() . . . 154
7.10.2.2 iniparser_AK_freedict() . . . 154
7.10.2.3 iniparser_dump() . . . 154
7.10.2.4 iniparser_dump_ini() . . . 155
7.10.2.5 iniparser_dumpsection_ini() . . . 155
7.10.2.6 iniparser_find_entry() . . . 156
7.10.2.7 iniparser_getboolean() . . . 156
7.10.2.8 iniparser_getdouble() . . . 157
7.10.2.9 iniparser_getint() . . . 157
7.10.2.10 iniparser_getnsec() . . . 158
7.10.2.11 iniparser_getseckeys() . . . 159
7.10.2.12 iniparser_getsecname() . . . 159
7.10.2.13 iniparser_getsecnkeys() . . . 160

7.10.2.14 iniparser_getstring() . . . 160
7.10.2.15 iniparser_load() . . . 161
7.10.2.16 iniparser_set() . . . 161
7.10.2.17 iniparser_unset() . . . 161
7.10.3 Variable Documentation . . . 162
7.10.3.1 AK_config . . . 162
7.11 auxi/mempro.c File Reference . . . 162
7.11.1 Detailed Description . . . 163
7.11.2 Function Documentation . . . 164
7.11.2.1 AK_calloc() . . . 164
7.11.2.2 AK_check_for_writes() . . . 164
7.11.2.3 AK_debmod_calloc() . . . 164
7.11.2.4 AK_debmod_d() . . . 165
7.11.2.5 AK_debmod_die() . . . 165
7.11.2.6 AK_debmod_dv() . . . 166
7.11.2.7 AK_debmod_enter_critical_sec() . . . 166
7.11.2.8 AK_debmod_free() . . . 167
7.11.2.9 AK_debmod_fstack_pop() . . . 167
7.11.2.10 AK_debmod_fstack_push() . . . 167
7.11.2.11 AK_debmod_func_add() . . . 168
7.11.2.12 AK_debmod_func_get_name() . . . 168
7.11.2.13 AK_debmod_func_id() . . . 169
7.11.2.14 AK_debmod_function_current() . . . 169
7.11.2.15 AK_debmod_function_epilogue() . . . 170
7.11.2.16 AK_debmod_function_prologue() . . . 170
7.11.2.17 AK_debmod_init() . . . 171
7.11.2.18 AK_debmod_leave_critical_sec() . . . 171
7.11.2.19 AK_debmod_log_memory_alloc() . . . 172
7.11.2.20 AK_debmod_print_function_use() . . . 172
7.11.2.21 AK_fread() . . . 172
7.11.2.22 AK_free() . . . 173
7.11.2.23 AK_fwrite() . . . 173
7.11.2.24 AK_malloc() . . . 174
7.11.2.25 AK_mempro_test() . . . 174
7.11.2.26 AK_print_active_functions() . . . 174
7.11.2.27 AK_print_function_use() . . . 175
7.11.2.28 AK_print_function_uses() . . . 175
7.11.2.29 AK_realloc() . . . 175
7.11.2.30 AK_write_protect() . . . 176
7.11.2.31 AK_write_unprotect() . . . 176
7.12 auxi/mempro.h File Reference . . . 177
7.12.1 Detailed Description . . . 179

7.12.2 Macro Definition Documentation . . . 179
7.12.2.1 AK_DEBMOD_MAX_FUNC_NAME . . . 179
7.12.2.2 AK_DEBMOD_MAX_FUNCTIONS . . . 179
7.12.2.3 AK_DEBMOD_MAX_WRITE_DETECTIONS . . . 179
7.12.2.4 AK_DEBMOD_ON . . . 180
7.12.2.5 AK_DEBMOD_PAGES_NUM . . . 180
7.12.2.6 AK_DEBMOD_PRINT . . . 180
7.12.2.7 AK_DEBMOD_STACKSIZE . . . 180
7.12.2.8 AK_EPI . . . 180
7.12.2.9 AK_INLINE . . . 180
7.12.2.10 AK_PRO . . . 181
7.12.2.11 NEW . . . 181
7.12.3 Function Documentation . . . 181
7.12.3.1 AK_calloc() . . . 181
7.12.3.2 AK_check_for_writes() . . . 181
7.12.3.3 AK_debmod_calloc() . . . 182
7.12.3.4 AK_debmod_d() . . . 182
7.12.3.5 AK_debmod_die() . . . 183
7.12.3.6 AK_debmod_dv() . . . 183
7.12.3.7 AK_debmod_enter_critical_sec() . . . 184
7.12.3.8 AK_debmod_free() . . . 184
7.12.3.9 AK_debmod_fstack_pop() . . . 184
7.12.3.10 AK_debmod_fstack_push() . . . 185
7.12.3.11 AK_debmod_func_add() . . . 185
7.12.3.12 AK_debmod_func_get_name() . . . 186
7.12.3.13 AK_debmod_func_id() . . . 186
7.12.3.14 AK_debmod_function_current() . . . 187
7.12.3.15 AK_debmod_function_epilogue() . . . 187
7.12.3.16 AK_debmod_function_prologue() . . . 188
7.12.3.17 AK_debmod_init() . . . 188
7.12.3.18 AK_debmod_leave_critical_sec() . . . 189
7.12.3.19 AK_debmod_log_memory_alloc() . . . 189
7.12.3.20 AK_debmod_print_function_use() . . . 189
7.12.3.21 AK_free() . . . 190
7.12.3.22 AK_malloc() . . . 190
7.12.3.23 AK_mempro_test() . . . 191
7.12.3.24 AK_print_active_functions() . . . 191
7.12.3.25 AK_print_function_use() . . . 191
7.12.3.26 AK_print_function_uses() . . . 192
7.12.3.27 AK_realloc() . . . 192
7.12.3.28 AK_write_protect() . . . 193
7.12.3.29 AK_write_unprotect() . . . 193

7.12.4 Variable Documentation . . . 193
7.12.4.1 AK_DEBMOD_STATE . . . 193
7.13 auxi/observable.c File Reference . . . 194
7.13.1 Detailed Description . . . 194
7.13.2 Typedef Documentation . . . 195
7.13.2.1 AK_TypeObservable . . . 195
7.13.2.2 AK_TypeObserver . . . 195
7.13.2.3 AK_TypeObserver_Second . . . 195
7.13.2.4 NotifyDetails . . . 195
7.13.3 Enumeration Type Documentation . . . 195
7.13.3.1 NotifyType . . . 195
7.13.4 Function Documentation . . . 195
7.13.4.1 AK_custom_action() . . . 196
7.13.4.2 AK_custom_register_observer() . . . 196
7.13.4.3 AK_custom_unregister_observer() . . . 196
7.13.4.4 AK_get_message() . . . 196
7.13.4.5 AK_init_observable() . . . 196
7.13.4.6 AK_init_observer() . . . 197
7.13.4.7 AK_observable_pattern() . . . 197
7.13.4.8 AK_observable_test() . . . 197
7.13.4.9 AK_set_notify_info_details() . . . 197
7.13.4.10 custom_observer_event_handler() . . . 198
7.13.4.11 handle_AK_custom_type() . . . 198
7.13.4.12 init_observable_type() . . . 198
7.13.4.13 init_observer_type() . . . 198
7.13.4.14 init_observer_type_second() . . . 198
7.14 auxi/observable.h File Reference . . . 198
7.14.1 Detailed Description . . . 199
7.14.2 Typedef Documentation . . . 199
7.14.2.1 AK_observable . . . 199
7.14.2.2 AK_observer . . . 199
7.14.3 Enumeration Type Documentation . . . 199
7.14.3.1 AK_ObservableType_Enum . . . 199
7.14.4 Function Documentation . . . 200
7.14.4.1 AK_init_observable() . . . 200
7.14.4.2 AK_init_observer() . . . 200
7.14.4.3 AK_observable_pattern() . . . 201
7.14.4.4 AK_observable_test() . . . 201
7.15 auxi/test.c File Reference . . . 201
7.15.1 Detailed Description . . . 201
7.15.2 Function Documentation . . . 201
7.15.2.1 TEST_output_results() . . . 201

7.15.2.2 TEST_result() . . . 202
7.16 file/test.c File Reference . . . 202
7.16.1 Detailed Description . . . 203
7.16.2 Function Documentation . . . 203
7.16.2.1 AK_create_test_table_assistant() . . . 203
7.16.2.2 AK_create_test_table_course() . . . 203
7.16.2.3 AK_create_test_table_department() . . . 204
7.16.2.4 AK_create_test_table_employee() . . . 204
7.16.2.5 AK_create_test_table_professor() . . . 204
7.16.2.6 AK_create_test_table_professor2() . . . 205
7.16.2.7 AK_create_test_table_student() . . . 205
7.16.2.8 AK_create_test_tables() . . . 205
7.16.2.9 AK_get_table_atribute_types() . . . 205
7.16.2.10 create_header_test() . . . 206
7.16.2.11 get_column_test() . . . 206
7.16.2.12 get_row_test() . . . 207
7.16.2.13 insert_data_test() . . . 207
7.16.2.14 selection_test() . . . 208
7.17 auxi/test.h File Reference . . . 208
7.17.1 Macro Definition Documentation . . . 209
7.17.1.1 BLACK . . . 209
7.17.1.2 BLUE . . . 209
7.17.1.3 BOLDBLACK . . . 210
7.17.1.4 BOLDBLUE . . . 210
7.17.1.5 BOLDCYAN . . . 210
7.17.1.6 BOLDGREEN . . . 210
7.17.1.7 BOLDMAGENTA . . . 210
7.17.1.8 BOLDRED . . . 210
7.17.1.9 BOLDWHITE . . . 210
7.17.1.10 BOLDYELLOW . . . 210
7.17.1.11 CYAN . . . 211
7.17.1.12 GREEN . . . 211
7.17.1.13 MAGENTA . . . 211
7.17.1.14 RED . . . 211
7.17.1.15 RESET . . . 211
7.17.1.16 WHITE . . . 211
7.17.1.17 YELLOW . . . 211
7.17.2 Typedef Documentation . . . 211
7.17.2.1 TestResult . . . 212
7.17.3 Function Documentation . . . 212
7.17.3.1 TEST_output_results() . . . 212
7.17.3.2 TEST_result() . . . 212

7.18 file/test.h File Reference . . . 213
7.18.1 Detailed Description . . . 213
7.18.2 Function Documentation . . . 213
7.18.2.1 AK_create_test_tables() . . . 213
7.18.2.2 AK_get_table_atribute_types() . . . 214
7.18.2.3 create_header_test() . . . 214
7.18.2.4 get_column_test() . . . 215
7.18.2.5 get_row_test() . . . 215
7.18.2.6 insert_data_test() . . . 216
7.18.2.7 selection_test() . . . 216
7.19 dm/dbman.c File Reference . . . 217
7.19.1 Detailed Description . . . 219
7.19.2 Function Documentation . . . 219
7.19.2.1 AK_allocate_block_activity_modes() . . . 219
7.19.2.2 AK_allocate_blocks() . . . 220
7.19.2.3 AK_allocationbit_test() . . . 220
7.19.2.4 AK_allocationtable_dump() . . . 220
7.19.2.5 AK_allocationtable_test() . . . 220
7.19.2.6 AK_blocktable_dump() . . . 221
7.19.2.7 AK_blocktable_flush() . . . 221
7.19.2.8 AK_blocktable_get() . . . 221
7.19.2.9 AK_copy_header() . . . 222
7.19.2.10 AK_create_header() . . . 222
7.19.2.11 AK_delete_block() . . . 223
7.19.2.12 AK_delete_extent() . . . 223
7.19.2.13 AK_delete_segment() . . . 224
7.19.2.14 AK_get_allocation_set() . . . 224
7.19.2.15 AK_get_extent() . . . 225
7.19.2.16 AK_increase_extent() . . . 225
7.19.2.17 AK_init_allocation_table() . . . 226
7.19.2.18 AK_init_block() . . . 226
7.19.2.19 AK_init_db_file() . . . 227
7.19.2.20 AK_init_disk_manager() . . . 227
7.19.2.21 AK_init_system_catalog() . . . 227
7.19.2.22 AK_init_system_tables_catalog() . . . 228
7.19.2.23 AK_insert_entry() . . . 229
7.19.2.24 AK_memset_int() . . . 230
7.19.2.25 AK_new_extent() . . . 230
7.19.2.26 AK_new_segment() . . . 231
7.19.2.27 AK_print_block() . . . 231
7.19.2.28 AK_read_block() . . . 232
7.19.2.29 AK_read_block_for_testing() . . . 232

7.19.2.30 AK_register_system_tables() . . . 232
7.19.2.31 AK_thread_safe_block_access_test() . . . 233
7.19.2.32 AK_write_block() . . . 234
7.19.2.33 AK_write_block_for_testing() . . . 234
7.19.2.34 fsize() . . . 234
7.19.3 Variable Documentation . . . 235
7.19.3.1 fileLockMutex . . . 235
7.19.3.2 test_lastCharacterWritten . . . 235
7.19.3.3 test_threadSafeBlockAccessSucceeded . . . 235
7.20 dm/dbman.h File Reference . . . 235
7.20.1 Detailed Description . . . 239
7.20.2 Macro Definition Documentation . . . 239
7.20.2.1 AK_ALLOCATION_TABLE_SIZE . . . 239
7.20.2.2 BITCLEAR . . . 239
7.20.2.3 BITMASK . . . 239
7.20.2.4 BITNSLOTS . . . 239
7.20.2.5 BITSET . . . 239
7.20.2.6 BITSLOT . . . 240
7.20.2.7 BITTEST . . . 240
7.20.2.8 CHAR_IN_LINE . . . 240
7.20.2.9 DB_FILE_BLOCKS_NUM_EX . . . 240
7.20.2.10 DB_FILE_SIZE_EX . . . 240
7.20.2.11 MAX_BLOCK_INIT_NUM . . . 240
7.20.2.12 SEGMENTLENGTH . . . 241
7.20.3 Enumeration Type Documentation . . . 241
7.20.3.1 AK_allocation_set_mode . . . 241
7.20.4 Function Documentation . . . 241
7.20.4.1 AK_allocate_blocks() . . . 241
7.20.4.2 AK_allocationbit_test() . . . 242
7.20.4.3 AK_allocationtable_dump() . . . 242
7.20.4.4 AK_allocationtable_test() . . . 242
7.20.4.5 AK_blocktable_dump() . . . 242
7.20.4.6 AK_blocktable_flush() . . . 243
7.20.4.7 AK_blocktable_get() . . . 243
7.20.4.8 AK_copy_header() . . . 243
7.20.4.9 AK_create_header() . . . 244
7.20.4.10 AK_delete_block() . . . 244
7.20.4.11 AK_delete_extent() . . . 245
7.20.4.12 AK_delete_segment() . . . 245
7.20.4.13 AK_get_allocation_set() . . . 246
7.20.4.14 AK_get_extent() . . . 246
7.20.4.15 AK_increase_extent() . . . 247

7.20.4.16 AK_init_allocation_table() . . . 248
7.20.4.17 AK_init_block() . . . 248
7.20.4.18 AK_init_db_file() . . . 248
7.20.4.19 AK_init_disk_manager() . . . 249
7.20.4.20 AK_init_system_catalog() . . . 249
7.20.4.21 AK_init_system_tables_catalog() . . . 250
7.20.4.22 AK_insert_entry() . . . 251
7.20.4.23 AK_memset_int() . . . 252
7.20.4.24 AK_new_extent() . . . 252
7.20.4.25 AK_new_segment() . . . 253
7.20.4.26 AK_print_block() . . . 253
7.20.4.27 AK_read_block() . . . 254
7.20.4.28 AK_read_block_for_testing() . . . 254
7.20.4.29 AK_register_system_tables() . . . 254
7.20.4.30 AK_thread_safe_block_access_test() . . . 255
7.20.4.31 AK_write_block() . . . 256
7.20.4.32 AK_write_block_for_testing() . . . 256
7.20.4.33 fsize() . . . 256
7.20.5 Variable Documentation . . . 257
7.20.5.1 AK_allocationbit . . . 257
7.20.5.2 AK_block_activity_info . . . 257
7.20.5.3 db . . . 257
7.20.5.4 db_file_size . . . 257
7.20.5.5 dbmanFileLock . . . 258
7.21 file/blobs.c File Reference . . . 258
7.21.1 Detailed Description . . . 259
7.21.2 Function Documentation . . . 259
7.21.2.1 AK_check_folder_blobs() . . . 259
7.21.2.2 AK_clear_all_newline() . . . 259
7.21.2.3 AK_concat() . . . 259
7.21.2.4 AK_copy() . . . 260
7.21.2.5 AK_File_Metadata_malloc() . . . 260
7.21.2.6 AK_folder_exists() . . . 260
7.21.2.7 AK_GUID() . . . 260
7.21.2.8 AK_lo_export() . . . 261
7.21.2.9 AK_lo_import() . . . 261
7.21.2.10 AK_lo_test() . . . 261
7.21.2.11 AK_lo_unlink() . . . 262
7.21.2.12 AK_mkdir() . . . 262
7.21.2.13 AK_read_metadata() . . . 262
7.21.2.14 AK_split_path_file() . . . 262
7.21.2.15 AK_write_metadata() . . . 263

7.21.3 Variable Documentation . . . 263
7.21.3.1 failed . . . 263
7.21.3.2 success . . . 263
7.22 file/blobs.h File Reference . . . 263
7.22.1 Detailed Description . . . 264
7.22.2 Typedef Documentation . . . 264
7.22.2.1 AK_File_Metadata . . . 264
7.22.2.2 AK_Metadata . . . 264
7.22.3 Function Documentation . . . 265
7.22.3.1 AK_check_folder_blobs() . . . 265
7.22.3.2 AK_clear_all_newline() . . . 265
7.22.3.3 AK_concat() . . . 265
7.22.3.4 AK_copy() . . . 266
7.22.3.5 AK_File_Metadata_malloc() . . . 266
7.22.3.6 AK_folder_exists() . . . 266
7.22.3.7 AK_GUID() . . . 266
7.22.3.8 AK_lo_export() . . . 267
7.22.3.9 AK_lo_import() . . . 267
7.22.3.10 AK_lo_test() . . . 267
7.22.3.11 AK_lo_unlink() . . . 268
7.22.3.12 AK_mkdir() . . . 268
7.22.3.13 AK_read_metadata() . . . 268
7.22.3.14 AK_split_path_file() . . . 268
7.22.3.15 AK_write_metadata() . . . 269
7.23 file/fileio.c File Reference . . . 269
7.23.1 Detailed Description . . . 269
7.23.2 Function Documentation . . . 270
7.23.2.1 AK_delete_row() . . . 270
7.23.2.2 AK_delete_row_by_id() . . . 270
7.23.2.3 AK_delete_row_from_block() . . . 270
7.23.2.4 AK_delete_update_segment() . . . 271
7.23.2.5 AK_fileio_test() . . . 271
7.23.2.6 AK_Insert_New_Element() . . . 272
7.23.2.7 AK_Insert_New_Element_For_Update() . . . 272
7.23.2.8 AK_insert_row() . . . 273
7.23.2.9 AK_insert_row_to_block() . . . 273
7.23.2.10 AK_Update_Existing_Element() . . . 274
7.23.2.11 AK_update_row() . . . 274
7.23.2.12 AK_update_row_from_block() . . . 275
7.24 file/fileio.h File Reference . . . 275
7.24.1 Detailed Description . . . 276
7.24.2 Function Documentation . . . 276

7.24.2.1 AK_delete_row() . . . 276
7.24.2.2 AK_delete_row_by_id() . . . 277
7.24.2.3 AK_delete_row_from_block() . . . 277
7.24.2.4 AK_delete_update_segment() . . . 278
7.24.2.5 AK_fileio_test() . . . 278
7.24.2.6 AK_Insert_New_Element() . . . 278
7.24.2.7 AK_Insert_New_Element_For_Update() . . . 279
7.24.2.8 AK_insert_row() . . . 279
7.24.2.9 AK_insert_row_to_block() . . . 280
7.24.2.10 AK_update_row() . . . 280
7.24.2.11 AK_update_row_from_block() . . . 281
7.25 file/files.c File Reference . . . 281
7.25.1 Detailed Description . . . 282
7.25.2 Function Documentation . . . 282
7.25.2.1 AK_files_test() . . . 282
7.25.2.2 AK_initialize_new_index_segment() . . . 282
7.25.2.3 AK_initialize_new_segment() . . . 283
7.25.3 Variable Documentation . . . 283
7.25.3.1 fileMut . . . 283
7.26 file/files.h File Reference . . . 283
7.26.1 Detailed Description . . . 284
7.26.2 Function Documentation . . . 284
7.26.2.1 AK_files_test() . . . 284
7.26.2.2 AK_initialize_new_index_segment() . . . 284
7.26.2.3 AK_initialize_new_segment() . . . 285
7.27 file/filesearch.c File Reference . . . 285
7.27.1 Detailed Description . . . 286
7.27.2 Function Documentation . . . 286
7.27.2.1 AK_deallocate_search_result() . . . 286
7.27.2.2 AK_filesearch_test() . . . 286
7.27.2.3 AK_search_unsorted() . . . 287
7.28 file/filesearch.h File Reference . . . 288
7.28.1 Detailed Description . . . 288
7.28.2 Macro Definition Documentation . . . 288
7.28.2.1 SEARCH_ALL . . . 289
7.28.2.2 SEARCH_NULL . . . 289
7.28.2.3 SEARCH_PARTICULAR . . . 289
7.28.2.4 SEARCH_RANGE . . . 289
7.28.3 Function Documentation . . . 289
7.28.3.1 AK_deallocate_search_result() . . . 289
7.28.3.2 AK_filesearch_test() . . . 290
7.28.3.3 AK_search_unsorted() . . . 290

7.29 file/filesort.c File Reference . . . 291
7.29.1 Function Documentation . . . 291
7.29.1.1 AK_block_sort() . . . 291
7.29.1.2 AK_filesort_test() . . . 292
7.29.1.3 AK_get_header_number() . . . 292
7.29.1.4 AK_get_num_of_tuples() . . . 292
7.29.1.5 AK_get_total_headers() . . . 293
7.29.1.6 AK_reset_block() . . . 293
7.29.1.7 AK_sort_segment() . . . 293
7.30 file/filesort.h File Reference . . . 294
7.30.1 Detailed Description . . . 294
7.30.2 Macro Definition Documentation . . . 295
7.30.2.1 DATA_ROW_SIZE . . . 295
7.30.2.2 DATA_TUPLE_SIZE . . . 295
7.30.3 Function Documentation . . . 295
7.30.3.1 AK_block_sort() . . . 295
7.30.3.2 AK_filesort_test() . . . 296
7.30.3.3 AK_get_header_number() . . . 296
7.30.3.4 AK_get_num_of_tuples() . . . 296
7.30.3.5 AK_get_total_headers() . . . 296
7.30.3.6 AK_reset_block() . . . 297
7.30.3.7 AK_sort_segment() . . . 297
7.31 file/id.c File Reference . . . 298
7.31.1 Detailed Description . . . 298
7.31.2 Function Documentation . . . 298
7.31.2.1 AK_get_id() . . . 298
7.31.2.2 AK_get_table_id() . . . 299
7.31.2.3 AK_id_test() . . . 299
7.32 file/id.h File Reference . . . 299
7.32.1 Detailed Description . . . 300
7.32.2 Macro Definition Documentation . . . 300
7.32.2.1 ID_START_VALUE . . . 300
7.32.3 Function Documentation . . . 300
7.32.3.1 AK_get_id() . . . 300
7.32.3.2 AK_id_test() . . . 300
7.33 file/idx/bitmap.c File Reference . . . 301
7.33.1 Detailed Description . . . 301
7.33.2 Function Documentation . . . 301
7.33.2.1 AK_add_to_bitmap_index() . . . 302
7.33.2.2 AK_bitmap_test() . . . 302
7.33.2.3 AK_create_Index() . . . 303
7.33.2.4 AK_create_Index_Table() . . . 303

7.33.2.5 AK_delete_bitmap_index() . . . 304
7.33.2.6 AK_get_attribute() . . . 304
7.33.2.7 AK_get_Attribute() . . . 305
7.33.2.8 AK_If_ExistOp() . . . 305
7.33.2.9 AK_print_Att_Test() . . . 306
7.33.2.10 AK_print_Header_Test() . . . 306
7.33.2.11 AK_update() . . . 307
7.34 file/idx/bitmap.h File Reference . . . 307
7.34.1 Detailed Description . . . 308
7.34.2 Function Documentation . . . 308
7.34.2.1 AK_add_to_bitmap_index() . . . 308
7.34.2.2 AK_bitmap_test() . . . 309
7.34.2.3 AK_create_Index() . . . 310
7.34.2.4 AK_create_Index_Table() . . . 310
7.34.2.5 AK_create_List_Address_Test() . . . 311
7.34.2.6 AK_delete_bitmap_index() . . . 311
7.34.2.7 AK_get_attribute() . . . 311
7.34.2.8 AK_get_Attribute() . . . 312
7.34.2.9 AK_If_ExistOp() . . . 312
7.34.2.10 AK_print_Att_Test() . . . 313
7.34.2.11 AK_print_Header_Test() . . . 313
7.34.2.12 AK_update() . . . 314
7.34.2.13 AK_write_block() . . . 314
7.35 file/idx/btree.c File Reference . . . 315
7.35.1 Detailed Description . . . 316
7.35.2 Function Documentation . . . 316
7.35.2.1 AK_btree_create() . . . 316
7.35.2.2 AK_btree_delete() . . . 316
7.35.2.3 AK_btree_insert() . . . 317
7.35.2.4 AK_btree_search_delete() . . . 317
7.35.2.5 AK_btree_test() . . . 318
7.35.2.6 btree_delete() . . . 318
7.35.2.7 findCorrectNumber() . . . 319
7.35.2.8 findPointers() . . . 319
7.35.2.9 findValues() . . . 320
7.35.2.10 makevalues() . . . 320
7.35.2.11 searchValue() . . . 321
7.35.2.12 setNodePointers() . . . 321
7.36 file/idx/btree.h File Reference . . . 322
7.36.1 Detailed Description . . . 323
7.36.2 Macro Definition Documentation . . . 323
7.36.2.1 B . . . 323

7.36.2.2 LEAF . . . 323
7.36.2.3 NODE . . . 323
7.36.2.4 ORDER . . . 324
7.36.3 Function Documentation . . . 324
7.36.3.1 AK_btree_create() . . . 324
7.36.3.2 AK_btree_delete() . . . 324
7.36.3.3 AK_btree_insert() . . . 325
7.36.3.4 AK_btree_search_delete() . . . 325
7.36.3.5 AK_btree_test() . . . 326
7.36.3.6 btree_delete() . . . 326
7.36.3.7 findCorrectNumber() . . . 326
7.36.3.8 findPointers() . . . 327
7.36.3.9 findValues() . . . 327
7.36.3.10 makevalues() . . . 328
7.36.3.11 searchValue() . . . 328
7.36.3.12 setNodePointers() . . . 329
7.37 file/idx/hash.c File Reference . . . 329
7.37.1 Detailed Description . . . 330
7.37.2 Function Documentation . . . 330
7.37.2.1 AK_change_hash_info() . . . 330
7.37.2.2 AK_create_hash_index() . . . 331
7.37.2.3 AK_delete_hash_index() . . . 331
7.37.2.4 AK_delete_in_hash_index() . . . 332
7.37.2.5 AK_elem_hash_value() . . . 332
7.37.2.6 AK_find_delete_in_hash_index() . . . 332
7.37.2.7 AK_find_in_hash_index() . . . 333
7.37.2.8 AK_get_hash_info() . . . 333
7.37.2.9 AK_get_nth_main_bucket_add() . . . 334
7.37.2.10 AK_hash_test() . . . 334
7.37.2.11 AK_insert_bucket_to_block() . . . 335
7.37.2.12 AK_insert_in_hash_index() . . . 335
7.37.2.13 AK_update_bucket_in_block() . . . 336
7.38 file/idx/hash.h File Reference . . . 336
7.38.1 Detailed Description . . . 337
7.38.2 Function Documentation . . . 337
7.38.2.1 AK_change_hash_info() . . . 337
7.38.2.2 AK_create_hash_index() . . . 338
7.38.2.3 AK_delete_hash_index() . . . 338
7.38.2.4 AK_delete_in_hash_index() . . . 339
7.38.2.5 AK_elem_hash_value() . . . 339
7.38.2.6 AK_find_delete_in_hash_index() . . . 339
7.38.2.7 AK_find_in_hash_index() . . . 340

7.38.2.8 AK_get_hash_info() . . . 340
7.38.2.9 AK_get_nth_main_bucket_add() . . . 341
7.38.2.10 AK_hash_test() . . . 341
7.38.2.11 AK_insert_bucket_to_block() . . . 342
7.38.2.12 AK_insert_in_hash_index() . . . 342
7.38.2.13 AK_update_bucket_in_block() . . . 343
7.39 file/idx/index.c File Reference . . . 343
7.39.1 Detailed Description . . . 344
7.39.2 Function Documentation . . . 344
7.39.2.1 AK_Delete_All_elementsAd() . . . 344
7.39.2.2 AK_Delete_elementAd() . . . 345
7.39.2.3 AK_Get_First_elementAd() . . . 345
7.39.2.4 AK_get_index_header() . . . 345
7.39.2.5 AK_get_index_num_records() . . . 346
7.39.2.6 AK_get_index_tuple() . . . 347
7.39.2.7 AK_Get_Last_elementAd() . . . 347
7.39.2.8 AK_Get_Next_elementAd() . . . 347
7.39.2.9 AK_Get_Position_Of_elementAd() . . . 348
7.39.2.10 AK_Get_Previous_elementAd() . . . 348
7.39.2.11 AK_index_table_exist() . . . 349
7.39.2.12 AK_index_test() . . . 349
7.39.2.13 AK_InitializelistAd() . . . 350
7.39.2.14 AK_Insert_NewelementAd() . . . 350
7.39.2.15 AK_num_index_attr() . . . 351
7.39.2.16 AK_print_index_table() . . . 351
7.40 file/idx/index.h File Reference . . . 351
7.40.1 Detailed Description . . . 352
7.40.2 Typedef Documentation . . . 353
7.40.2.1 element_ad . . . 353
7.40.2.2 list_ad . . . 353
7.40.2.3 list_structure_ad . . . 353
7.40.3 Function Documentation . . . 353
7.40.3.1 AK_Delete_All_elementsAd() . . . 353
7.40.3.2 AK_Delete_elementAd() . . . 354
7.40.3.3 AK_Get_First_elementAd() . . . 354
7.40.3.4 AK_get_index_num_records() . . . 354
7.40.3.5 AK_get_index_tuple() . . . 355
7.40.3.6 AK_Get_Last_elementAd() . . . 356
7.40.3.7 AK_Get_Next_elementAd() . . . 356
7.40.3.8 AK_Get_Position_Of_elementAd() . . . 356
7.40.3.9 AK_Get_Previous_elementAd() . . . 357
7.40.3.10 AK_index_table_exist() . . . 357

7.40.3.11 AK_index_test() . . . 358
7.40.3.12 AK_InitializelistAd() . . . 358
7.40.3.13 AK_Insert_NewelementAd() . . . 359
7.40.3.14 AK_num_index_attr() . . . 359
7.40.3.15 AK_print_index_table() . . . 360
7.41 file/sequence.c File Reference . . . 360
7.41.1 Detailed Description . . . 360
7.41.2 Function Documentation . . . 361
7.41.2.1 AK_sequence_add() . . . 361
7.41.2.2 AK_sequence_current_value() . . . 361
7.41.2.3 AK_sequence_get_id() . . . 362
7.41.2.4 AK_sequence_modify() . . . 362
7.41.2.5 AK_sequence_next_value() . . . 363
7.41.2.6 AK_sequence_remove() . . . 363
7.41.2.7 AK_sequence_rename() . . . 364
7.41.2.8 AK_sequence_test() . . . 364
7.42 file/sequence.h File Reference . . . 364
7.42.1 Detailed Description . . . 365
7.42.2 Function Documentation . . . 365
7.42.2.1 AK_sequence_add() . . . 365
7.42.2.2 AK_sequence_current_value() . . . 366
7.42.2.3 AK_sequence_get_id() . . . 366
7.42.2.4 AK_sequence_modify() . . . 367
7.42.2.5 AK_sequence_next_value() . . . 367
7.42.2.6 AK_sequence_remove() . . . 368
7.42.2.7 AK_sequence_rename() . . . 368
7.42.2.8 AK_sequence_test() . . . 369
7.43 file/table.c File Reference . . . 369
7.43.1 Detailed Description . . . 370
7.43.2 Function Documentation . . . 370
7.43.2.1 AK_check_tables_scheme() . . . 371
7.43.2.2 AK_create_create_table_parameter() . . . 371
7.43.2.3 AK_create_table() . . . 371
7.43.2.4 AK_get_attr_index() . . . 372
7.43.2.5 AK_get_attr_name() . . . 372
7.43.2.6 AK_get_column() . . . 373
7.43.2.7 AK_get_header() . . . 373
7.43.2.8 AK_get_num_records() . . . 374
7.43.2.9 AK_get_row() . . . 374
7.43.2.10 AK_get_table_obj_id() . . . 375
7.43.2.11 AK_get_tuple() . . . 375
7.43.2.12 AK_num_attr() . . . 376

7.43.2.13 AK_op_rename_test() . . . 376
7.43.2.14 AK_print_row() . . . 377
7.43.2.15 AK_print_row_spacer() . . . 377
7.43.2.16 AK_print_row_spacer_to_file() . . . 377
7.43.2.17 AK_print_row_to_file() . . . 378
7.43.2.18 AK_print_table() . . . 378
7.43.2.19 AK_print_table_to_file() . . . 379
7.43.2.20 AK_rename() . . . 379
7.43.2.21 AK_table_empty() . . . 380
7.43.2.22 AK_table_exist() . . . 380
7.43.2.23 AK_table_test() . . . 381
7.43.2.24 AK_temp_create_table() . . . 381
7.43.2.25 AK_tuple_to_string() . . . 381
7.43.2.26 get_row_attr_data() . . . 382
7.44 file/table.h File Reference . . . 382
7.44.1 Detailed Description . . . 384
7.44.2 Macro Definition Documentation . . . 384
7.44.2.1 TABLE . . . 384
7.44.3 Typedef Documentation . . . 384
7.44.3.1 AK_create_table_parameter . . . 384
7.44.4 Function Documentation . . . 384
7.44.4.1 AK_check_tables_scheme() . . . 385
7.44.4.2 AK_create_create_table_parameter() . . . 385
7.44.4.3 AK_create_table() . . . 385
7.44.4.4 AK_get_attr_index() . . . 386
7.44.4.5 AK_get_attr_name() . . . 386
7.44.4.6 AK_get_column() . . . 387
7.44.4.7 AK_get_header() . . . 387
7.44.4.8 AK_get_num_records() . . . 388
7.44.4.9 AK_get_row() . . . 388
7.44.4.10 AK_get_table_obj_id() . . . 389
7.44.4.11 AK_get_tuple() . . . 389
7.44.4.12 AK_num_attr() . . . 390
7.44.4.13 AK_op_rename_test() . . . 390
7.44.4.14 AK_print_row() . . . 391
7.44.4.15 AK_print_row_spacer() . . . 391
7.44.4.16 AK_print_row_spacer_to_file() . . . 391
7.44.4.17 AK_print_row_to_file() . . . 392
7.44.4.18 AK_print_table() . . . 392
7.44.4.19 AK_print_table_to_file() . . . 393
7.44.4.20 AK_rename() . . . 393
7.44.4.21 AK_table_empty() . . . 394

7.44.4.22 AK_table_test() . . . 394
7.44.4.23 AK_temp_create_table() . . . 394
7.44.4.24 AK_tuple_to_string() . . . 395
7.44.4.25 get_row_attr_data() . . . 395
7.45 file/tableOld.c File Reference . . . 396
7.45.1 Function Documentation . . . 397
7.45.1.1 AK_check_tables_scheme() . . . 397
7.45.1.2 AK_create_create_table_parameter() . . . 397
7.45.1.3 AK_create_table() . . . 398
7.45.1.4 AK_get_attr_index() . . . 398
7.45.1.5 AK_get_attr_name() . . . 399
7.45.1.6 AK_get_column() . . . 399
7.45.1.7 AK_get_header() . . . 399
7.45.1.8 AK_get_num_records() . . . 400
7.45.1.9 AK_get_row() . . . 401
7.45.1.10 AK_get_table_obj_id() . . . 401
7.45.1.11 AK_get_tuple() . . . 401
7.45.1.12 AK_num_attr() . . . 402
7.45.1.13 AK_op_rename_test() . . . 402
7.45.1.14 AK_print_row() . . . 403
7.45.1.15 AK_print_row_spacer() . . . 403
7.45.1.16 AK_print_row_spacer_to_file() . . . 404
7.45.1.17 AK_print_row_to_file() . . . 404
7.45.1.18 AK_print_table() . . . 405
7.45.1.19 AK_print_table_to_file() . . . 405
7.45.1.20 AK_rename() . . . 405
7.45.1.21 AK_table_empty() . . . 406
7.45.1.22 AK_table_exist() . . . 406
7.45.1.23 AK_table_test() . . . 407
7.45.1.24 AK_temp_create_table() . . . 407
7.45.1.25 AK_tuple_to_string() . . . 408
7.45.1.26 get_row_attr_data() . . . 408
7.46 file/tableOld.h File Reference . . . 409
7.46.1 Macro Definition Documentation . . . 410
7.46.1.1 TABLE . . . 410
7.46.2 Typedef Documentation . . . 410
7.46.2.1 AK_create_table_parameter . . . 410
7.46.3 Function Documentation . . . 410
7.46.3.1 AK_check_tables_scheme() . . . 411
7.46.3.2 AK_create_create_table_parameter() . . . 411
7.46.3.3 AK_create_table() . . . 411
7.46.3.4 AK_get_attr_index() . . . 412

7.46.3.5 AK_get_attr_name() . . . 412
7.46.3.6 AK_get_column() . . . 413
7.46.3.7 AK_get_header() . . . 413
7.46.3.8 AK_get_num_records() . . . 414
7.46.3.9 AK_get_row() . . . 414
7.46.3.10 AK_get_table_obj_id() . . . 415
7.46.3.11 AK_get_tuple() . . . 415
7.46.3.12 AK_num_attr() . . . 416
7.46.3.13 AK_op_rename_test() . . . 416
7.46.3.14 AK_print_row() . . . 417
7.46.3.15 AK_print_row_spacer() . . . 417
7.46.3.16 AK_print_row_spacer_to_file() . . . 417
7.46.3.17 AK_print_row_to_file() . . . 418
7.46.3.18 AK_print_table() . . . 418
7.46.3.19 AK_print_table_to_file() . . . 419
7.46.3.20 AK_rename() . . . 419
7.46.3.21 AK_table_empty() . . . 420
7.46.3.22 AK_table_test() . . . 420
7.46.3.23 AK_temp_create_table() . . . 420
7.46.3.24 AK_tuple_to_string() . . . 421
7.46.3.25 get_row_attr_data() . . . 421
7.47 mm/memoman.c File Reference . . . 422
7.47.1 Detailed Description . . . 423
7.47.2 Function Documentation . . . 423
7.47.2.1 AK_cache_AK_malloc() . . . 423
7.47.2.2 AK_cache_block() . . . 423
7.47.2.3 AK_cache_result() . . . 424
7.47.2.4 AK_find_AK_free_space() . . . 424
7.47.2.5 AK_find_available_result_block() . . . 425
7.47.2.6 AK_flush_cache() . . . 425
7.47.2.7 AK_generate_result_id() . . . 425
7.47.2.8 AK_get_block() . . . 426
7.47.2.9 AK_get_index_addresses() . . . 426
7.47.2.10 AK_get_index_segment_addresses() . . . 427
7.47.2.11 AK_get_segment_addresses() . . . 427
7.47.2.12 AK_get_segment_addresses_internal() . . . 427
7.47.2.13 AK_get_system_table_address() . . . 428
7.47.2.14 AK_get_table_addresses() . . . 428
7.47.2.15 AK_init_new_extent() . . . 429
7.47.2.16 AK_mem_block_modify() . . . 429
7.47.2.17 AK_memoman_init() . . . 430
7.47.2.18 AK_memoman_test() . . . 430

7.47.2.19 AK_memoman_test2() . . . 430
7.47.2.20 AK_query_mem_AK_free() . . . 430
7.47.2.21 AK_query_mem_AK_malloc() . . . 431
7.47.2.22 AK_redo_log_AK_malloc() . . . 431
7.47.2.23 AK_refresh_cache() . . . 431
7.47.2.24 AK_release_oldest_cache_block() . . . 432
7.48 mm/memoman.h File Reference . . . 432
7.48.1 Detailed Description . . . 434
7.48.2 Function Documentation . . . 434
7.48.2.1 AK_cache_AK_malloc() . . . 434
7.48.2.2 AK_cache_block() . . . 434
7.48.2.3 AK_cache_result() . . . 435
7.48.2.4 AK_find_AK_free_space() . . . 435
7.48.2.5 AK_find_available_result_block() . . . 436
7.48.2.6 AK_flush_cache() . . . 436
7.48.2.7 AK_generate_result_id() . . . 436
7.48.2.8 AK_get_block() . . . 437
7.48.2.9 AK_get_index_addresses() . . . 437
7.48.2.10 AK_get_index_segment_addresses() . . . 438
7.48.2.11 AK_get_segment_addresses() . . . 438
7.48.2.12 AK_get_segment_addresses_internal() . . . 439
7.48.2.13 AK_get_table_addresses() . . . 439
7.48.2.14 AK_init_new_extent() . . . 440
7.48.2.15 AK_mem_block_modify() . . . 440
7.48.2.16 AK_memoman_init() . . . 441
7.48.2.17 AK_memoman_test() . . . 441
7.48.2.18 AK_memoman_test2() . . . 441
7.48.2.19 AK_query_mem_AK_free() . . . 441
7.48.2.20 AK_query_mem_AK_malloc() . . . 442
7.48.2.21 AK_redo_log_AK_malloc() . . . 442
7.48.2.22 AK_refresh_cache() . . . 442
7.48.2.23 AK_release_oldest_cache_block() . . . 443
7.48.3 Variable Documentation . . . 443
7.48.3.1 db_cache . . . 443
7.48.3.2 query_mem . . . 443
7.48.3.3 redo_log . . . 443
7.49 opti/query_optimization.c File Reference . . . 443
7.49.1 Detailed Description . . . 444
7.49.2 Function Documentation . . . 444
7.49.2.1 AK_execute_rel_eq() . . . 444
7.49.2.2 AK_print_optimized_query() . . . 445
7.49.2.3 AK_query_optimization() . . . 445

7.49.2.4 AK_query_optimization_test() . . . 446
7.49.3 Variable Documentation . . . 446
7.49.3.1 error_message . . . 446
7.50 opti/query_optimization.h File Reference . . . 446
7.50.1 Detailed Description . . . 447
7.50.2 Macro Definition Documentation . . . 447
7.50.2.1 MAX_PERMUTATION . . . 447
7.50.3 Function Documentation . . . 447
7.50.3.1 AK_execute_rel_eq() . . . 447
7.50.3.2 AK_print_optimized_query() . . . 448
7.50.3.3 AK_query_optimization() . . . 448
7.50.3.4 AK_query_optimization_test() . . . 449
7.51 opti/rel_eq_assoc.c File Reference . . . 449
7.51.1 Detailed Description . . . 450
7.51.2 Function Documentation . . . 450
7.51.2.1 AK_compare() . . . 450
7.51.2.2 AK_print_rel_eq_assoc() . . . 450
7.51.2.3 AK_rel_eq_assoc() . . . 451
7.51.2.4 AK_rel_eq_assoc_test() . . . 451
7.52 opti/rel_eq_assoc.h File Reference . . . 451
7.52.1 Detailed Description . . . 452
7.52.2 Typedef Documentation . . . 452
7.52.2.1 cost_eval . . . 452
7.52.3 Function Documentation . . . 452
7.52.3.1 AK_compare() . . . 452
7.52.3.2 AK_print_rel_eq_assoc() . . . 453
7.52.3.3 AK_rel_eq_assoc() . . . 453
7.52.3.4 AK_rel_eq_assoc_test() . . . 454
7.53 opti/rel_eq_comut.c File Reference . . . 454
7.53.1 Detailed Description . . . 454
7.53.2 Function Documentation . . . 454
7.53.2.1 AK_print_rel_eq_comut() . . . 454
7.53.2.2 AK_rel_eq_commute_with_theta_join() . . . 455
7.53.2.3 AK_rel_eq_comut() . . . 455
7.53.2.4 AK_rel_eq_comut_test() . . . 456
7.54 opti/rel_eq_comut.h File Reference . . . 456
7.54.1 Detailed Description . . . 456
7.54.2 Function Documentation . . . 457
7.54.2.1 AK_print_rel_eq_comut() . . . 457
7.54.2.2 AK_rel_eq_commute_with_theta_join() . . . 457
7.54.2.3 AK_rel_eq_comut() . . . 458
7.54.2.4 AK_rel_eq_comut_test() . . . 458

7.55 opti/rel_eq_projection.c File Reference . . . 458
7.55.1 Detailed Description . . . 459
7.55.2 Function Documentation . . . 459
7.55.2.1 AK_print_rel_eq_projection() . . . 459
7.55.2.2 AK_rel_eq_can_commute() . . . 460
7.55.2.3 AK_rel_eq_collect_cond_attributes() . . . 460
7.55.2.4 AK_rel_eq_get_attributes() . . . 461
7.55.2.5 AK_rel_eq_is_subset() . . . 461
7.55.2.6 AK_rel_eq_projection() . . . 462
7.55.2.7 AK_rel_eq_projection_attributes() . . . 463
7.55.2.8 AK_rel_eq_projection_test() . . . 463
7.55.2.9 AK_rel_eq_remove_duplicates() . . . 464
7.56 opti/rel_eq_projection.h File Reference . . . 464
7.56.1 Detailed Description . . . 465
7.56.2 Function Documentation . . . 465
7.56.2.1 AK_print_rel_eq_projection() . . . 465
7.56.2.2 AK_rel_eq_can_commute() . . . 465
7.56.2.3 AK_rel_eq_collect_cond_attributes() . . . 466
7.56.2.4 AK_rel_eq_get_attributes() . . . 466
7.56.2.5 AK_rel_eq_is_subset() . . . 467
7.56.2.6 AK_rel_eq_projection() . . . 468
7.56.2.7 AK_rel_eq_projection_attributes() . . . 469
7.56.2.8 AK_rel_eq_projection_test() . . . 469
7.56.2.9 AK_rel_eq_remove_duplicates() . . . 470
7.57 opti/rel_eq_selection.c File Reference . . . 470
7.57.1 Detailed Description . . . 471
7.57.2 Function Documentation . . . 471
7.57.2.1 AK_print_rel_eq_selection() . . . 471
7.57.2.2 AK_rel_eq_cond_attributes() . . . 471
7.57.2.3 AK_rel_eq_get_atrributes_char() . . . 472
7.57.2.4 AK_rel_eq_is_attr_subset() . . . 472
7.57.2.5 AK_rel_eq_selection() . . . 473
7.57.2.6 AK_rel_eq_selection_test() . . . 473
7.57.2.7 AK_rel_eq_share_attributes() . . . 474
7.57.2.8 AK_rel_eq_split_condition() . . . 474
7.58 opti/rel_eq_selection.h File Reference . . . 475
7.58.1 Detailed Description . . . 476
7.58.2 Function Documentation . . . 476
7.58.2.1 AK_print_rel_eq_selection() . . . 476
7.58.2.2 AK_rel_eq_cond_attributes() . . . 477
7.58.2.3 AK_rel_eq_get_atrributes_char() . . . 477
7.58.2.4 AK_rel_eq_is_attr_subset() . . . 479

7.58.2.5 AK_rel_eq_selection() . . . 480
7.58.2.6 AK_rel_eq_selection_test() . . . 480
7.58.2.7 AK_rel_eq_share_attributes() . . . 480
7.58.2.8 AK_rel_eq_split_condition() . . . 481
7.59 rec/archive_log.c File Reference . . . 482
7.59.1 Function Documentation . . . 483
7.59.1.1 AK_archive_log() . . . 483
7.59.1.2 AK_check_folder_archivelog() . . . 483
7.59.1.3 AK_get_timestamp() . . . 484
7.60 rec/archive_log.h File Reference . . . 484
7.60.1 Detailed Description . . . 484
7.60.2 Function Documentation . . . 484
7.60.2.1 AK_archive_log() . . . 485
7.60.2.2 AK_get_timestamp() . . . 485
7.61 rec/recovery.c File Reference . . . 486
7.61.1 Function Documentation . . . 486
7.61.1.1 AK_load_chosen_log() . . . 486
7.61.1.2 AK_load_latest_log() . . . 487
7.61.1.3 AK_recover_archive_log() . . . 487
7.61.1.4 AK_recover_operation() . . . 488
7.61.1.5 AK_recovery_insert_row() . . . 488
7.61.1.6 AK_recovery_test() . . . 489
7.61.1.7 AK_recovery_tokenize() . . . 489
7.61.1.8 recovery_insert_row() . . . 490
7.61.2 Variable Documentation . . . 490
7.61.2.1 grandfailure . . . 490
7.62 rec/recovery.h File Reference . . . 490
7.62.1 Function Documentation . . . 491
7.62.1.1 AK_load_chosen_log() . . . 491
7.62.1.2 AK_load_latest_log() . . . 491
7.62.1.3 AK_recover_archive_log() . . . 492
7.62.1.4 AK_recover_operation() . . . 492
7.62.1.5 AK_recovery_insert_row() . . . 493
7.62.1.6 AK_recovery_test() . . . 493
7.62.1.7 AK_recovery_tokenize() . . . 494
7.63 rec/redo_log.c File Reference . . . 494
7.63.1 Detailed Description . . . 494
7.63.2 Function Documentation . . . 495
7.63.2.1 AK_add_to_redolog() . . . 495
7.63.2.2 AK_add_to_redolog_select() . . . 495
7.63.2.3 AK_check_attributes() . . . 495
7.63.2.4 AK_check_redo_log_select() . . . 496

7.63.2.5 AK_printout_redolog() . . . 496
7.63.2.6 AK_redolog_commit() . . . 496
7.64 rec/redo_log.h File Reference . . . 496
7.64.1 Function Documentation . . . 497
7.64.1.1 AK_add_to_redolog() . . . 497
7.64.1.2 AK_add_to_redolog_select() . . . 497
7.64.1.3 AK_check_attributes() . . . 498
7.64.1.4 AK_check_redo_log_select() . . . 498
7.64.1.5 AK_printout_redolog() . . . 498
7.64.1.6 AK_redolog_commit() . . . 499
7.65 rel/aggregation.c File Reference . . . 499
7.65.1 Detailed Description . . . 499
7.65.2 Function Documentation . . . 499
7.65.2.1 AK_agg_input_add() . . . 500
7.65.2.2 AK_agg_input_add_to_beginning() . . . 500
7.65.2.3 AK_agg_input_fix() . . . 501
7.65.2.4 AK_agg_input_init() . . . 501
7.65.2.5 AK_aggregation() . . . 501
7.65.2.6 AK_aggregation_test() . . . 502
7.65.2.7 AK_header_size() . . . 503
7.65.2.8 AK_search_unsorted() . . . 503
7.66 rel/aggregation.h File Reference . . . 504
7.66.1 Detailed Description . . . 505
7.66.2 Macro Definition Documentation . . . 505
7.66.2.1 AGG_TASK_AVG . . . 505
7.66.2.2 AGG_TASK_AVG_COUNT . . . 505
7.66.2.3 AGG_TASK_AVG_SUM . . . 505
7.66.2.4 AGG_TASK_COUNT . . . 506
7.66.2.5 AGG_TASK_GROUP . . . 506
7.66.2.6 AGG_TASK_MAX . . . 506
7.66.2.7 AGG_TASK_MIN . . . 506
7.66.2.8 AGG_TASK_SUM . . . 506
7.66.3 Function Documentation . . . 506
7.66.3.1 AK_agg_input_add() . . . 506
7.66.3.2 AK_agg_input_add_to_beginning() . . . 507
7.66.3.3 AK_agg_input_fix() . . . 507
7.66.3.4 AK_agg_input_init() . . . 508
7.66.3.5 AK_aggregation() . . . 508
7.66.3.6 AK_aggregation_test() . . . 509
7.66.3.7 AK_header_size() . . . 509
7.67 rel/difference.c File Reference . . . 510
7.67.1 Detailed Description . . . 510

7.67.2 Function Documentation . . . 510
7.67.2.1 AK_difference() . . . 510
7.67.2.2 AK_op_difference_test() . . . 511
7.68 rel/difference.h File Reference . . . 511
7.68.1 Detailed Description . . . 511
7.68.2 Function Documentation . . . 512
7.68.2.1 AK_difference() . . . 512
7.68.2.2 AK_op_difference_test() . . . 512
7.69 rel/expression_check.c File Reference . . . 512
7.69.1 Detailed Description . . . 513
7.69.2 Function Documentation . . . 513
7.69.2.1 AK_check_arithmetic_statement() . . . 513
7.69.2.2 AK_check_if_row_satisfies_expression() . . . 514
7.69.2.3 AK_check_regex_expression() . . . 514
7.69.2.4 AK_check_regex_operator_expression() . . . 515
7.69.2.5 AK_expression_check_test() . . . 515
7.69.2.6 AK_replace_wild_card() . . . 515
7.70 rel/expression_check.h File Reference . . . 516
7.70.1 Detailed Description . . . 516
7.70.2 Function Documentation . . . 516
7.70.2.1 AK_check_arithmetic_statement() . . . 516
7.70.2.2 AK_check_if_row_satisfies_expression() . . . 517
7.70.2.3 AK_check_regex_expression() . . . 518
7.70.2.4 AK_check_regex_operator_expression() . . . 518
7.70.2.5 AK_expression_check_test() . . . 519
7.71 rel/intersect.c File Reference . . . 519
7.71.1 Detailed Description . . . 519
7.71.2 Function Documentation . . . 519
7.71.2.1 AK_intersect() . . . 519
7.71.2.2 AK_op_intersect_test() . . . 520
7.72 rel/intersect.h File Reference . . . 520
7.72.1 Detailed Description . . . 521
7.72.2 Function Documentation . . . 521
7.72.2.1 AK_intersect() . . . 521
7.72.2.2 AK_op_intersect_test() . . . 521
7.73 rel/nat_join.c File Reference . . . 522
7.73.1 Detailed Description . . . 522
7.73.2 Function Documentation . . . 522
7.73.2.1 AK_copy_blocks_join() . . . 522
7.73.2.2 AK_create_join_block_header() . . . 523
7.73.2.3 AK_join() . . . 523
7.73.2.4 AK_merge_block_join() . . . 524

7.73.2.5 AK_op_join_test() . . . 524
7.74 rel/nat_join.h File Reference . . . 525
7.74.1 Detailed Description . . . 525
7.74.2 Function Documentation . . . 525
7.74.2.1 AK_copy_blocks_join() . . . 526
7.74.2.2 AK_create_join_block_header() . . . 526
7.74.2.3 AK_join() . . . 527
7.74.2.4 AK_merge_block_join() . . . 527
7.74.2.5 AK_op_join_test() . . . 528
7.75 rel/product.c File Reference . . . 528
7.75.1 Detailed Description . . . 528
7.75.2 Function Documentation . . . 529
7.75.2.1 AK_op_product_test() . . . 529
7.75.2.2 AK_product() . . . 529
7.75.2.3 AK_product_procedure() . . . 530
7.76 rel/product.h File Reference . . . 530
7.76.1 Detailed Description . . . 531
7.76.2 Function Documentation . . . 531
7.76.2.1 AK_op_product_test() . . . 531
7.76.2.2 AK_product() . . . 531
7.76.2.3 AK_product_procedure() . . . 532
7.77 rel/projection.c File Reference . . . 533
7.77.1 Detailed Description . . . 533
7.77.2 Function Documentation . . . 533
7.77.2.1 AK_copy_block_projection() . . . 534
7.77.2.2 AK_create_block_header() . . . 534
7.77.2.3 AK_create_header_name() . . . 535
7.77.2.4 AK_determine_header_type() . . . 535
7.77.2.5 AK_get_operator() . . . 536
7.77.2.6 AK_op_projection_test() . . . 536
7.77.2.7 AK_perform_operation() . . . 537
7.77.2.8 AK_projection() . . . 537
7.77.2.9 AK_remove_substring() . . . 538
7.78 rel/projection.h File Reference . . . 538
7.78.1 Detailed Description . . . 539
7.78.2 Function Documentation . . . 539
7.78.2.1 AK_copy_block_projection() . . . 539
7.78.2.2 AK_create_block_header() . . . 540
7.78.2.3 AK_create_header_name() . . . 540
7.78.2.4 AK_determine_header_type() . . . 541
7.78.2.5 AK_get_operator() . . . 541
7.78.2.6 AK_op_projection_test() . . . 542

7.78.2.7 AK_perform_operation() . . . 542
7.78.2.8 AK_projection() . . . 543
7.78.2.9 AK_remove_substring() . . . 543
7.79 rel/selection.c File Reference . . . 544
7.79.1 Detailed Description . . . 544
7.79.2 Function Documentation . . . 544
7.79.2.1 AK_op_selection_test() . . . 544
7.79.2.2 AK_op_selection_test_pattern() . . . 545
7.79.2.3 AK_selection() . . . 545
7.79.2.4 AK_selection_op_rename() . . . 545
7.80 rel/selection.h File Reference . . . 546
7.80.1 Detailed Description . . . 546
7.80.2 Function Documentation . . . 546
7.80.2.1 AK_op_selection_test() . . . 546
7.80.2.2 AK_op_selection_test_pattern() . . . 547
7.80.2.3 AK_selection() . . . 547
7.81 rel/theta_join.c File Reference . . . 547
7.81.1 Detailed Description . . . 548
7.81.2 Function Documentation . . . 548
7.81.2.1 AK_check_constraints() . . . 548
7.81.2.2 AK_create_theta_join_header() . . . 549
7.81.2.3 AK_op_theta_join_test() . . . 549
7.81.2.4 AK_theta_join() . . . 550
7.82 rel/theta_join.h File Reference . . . 550
7.82.1 Detailed Description . . . 551
7.82.2 Function Documentation . . . 551
7.82.2.1 AK_check_constraints() . . . 551
7.82.2.2 AK_create_theta_join_header() . . . 551
7.82.2.3 AK_op_theta_join_test() . . . 552
7.82.2.4 AK_theta_join() . . . 552
7.83 rel/union.c File Reference . . . 553
7.83.1 Detailed Description . . . 553
7.83.2 Function Documentation . . . 554
7.83.2.1 AK_op_union_test() . . . 554
7.83.2.2 AK_union() . . . 554
7.84 rel/union.h File Reference . . . 554
7.84.1 Detailed Description . . . 555
7.84.2 Function Documentation . . . 555
7.84.2.1 AK_op_union_test() . . . 555
7.84.2.2 AK_union() . . . 555
7.85 sql/command.c File Reference . . . 556
7.85.1 Detailed Description . . . 556

7.85.2 Function Documentation . . . 556
7.85.2.1 AK_command() . . . 557
7.85.2.2 AK_test_command() . . . 557
7.86 sql/command.h File Reference . . . 557
7.86.1 Detailed Description . . . 558
7.86.2 Typedef Documentation . . . 558
7.86.2.1 command . . . 558
7.86.3 Function Documentation . . . 558
7.86.3.1 AK_command() . . . 558
7.86.3.2 AK_test_command() . . . 559
7.87 sql/cs/between.c File Reference . . . 559
7.87.1 Detailed Description . . . 559
7.87.2 Function Documentation . . . 559
7.87.2.1 AK_constraint_between_test() . . . 560
7.87.2.2 AK_delete_constraint_between() . . . 560
7.87.2.3 AK_find_table_address() . . . 560
7.87.2.4 AK_print_constraints() . . . 561
7.87.2.5 AK_read_constraint_between() . . . 561
7.87.2.6 AK_set_constraint_between() . . . 562
7.88 sql/cs/between.h File Reference . . . 562
7.88.1 Detailed Description . . . 563
7.88.2 Function Documentation . . . 563
7.88.2.1 AK_constraint_between_test() . . . 563
7.88.2.2 AK_delete_constraint_between() . . . 563
7.88.2.3 AK_find_table_address() . . . 564
7.88.2.4 AK_read_constraint_between() . . . 564
7.88.2.5 AK_set_constraint_between() . . . 565
7.89 sql/cs/check_constraint.c File Reference . . . 566
7.89.1 Detailed Description . . . 566
7.89.2 Function Documentation . . . 567
7.89.2.1 AK_check_constraint() . . . 567
7.89.2.2 AK_check_constraint_test() . . . 567
7.89.2.3 AK_delete_check_constraint() . . . 568
7.89.2.4 AK_set_check_constraint() . . . 568
7.89.2.5 condition_passed() . . . 569
7.90 sql/cs/check_constraint.h File Reference . . . 569
7.90.1 Detailed Description . . . 570
7.90.2 Function Documentation . . . 570
7.90.2.1 AK_check_constraint_test() . . . 570
7.90.2.2 AK_delete_check_constraint() . . . 570
7.90.2.3 AK_set_check_constraint() . . . 571
7.90.2.4 condition_passed() . . . 571

7.91 sql/cs/constraint_names.c File Reference . . . 572
7.91.1 Detailed Description . . . 572
7.91.2 Function Documentation . . . 572
7.91.2.1 AK_check_constraint_name() . . . 572
7.91.2.2 AK_constraint_names_test() . . . 573
7.92 sql/cs/constraint_names.h File Reference . . . 573
7.92.1 Detailed Description . . . 573
7.92.2 Function Documentation . . . 573
7.92.2.1 AK_check_constraint_name() . . . 574
7.92.2.2 AK_constraint_names_test() . . . 574
7.93 sql/cs/nnull.c File Reference . . . 575
7.93.1 Detailed Description . . . 575
7.93.2 Function Documentation . . . 575
7.93.2.1 AK_check_constraint_not_null() . . . 575
7.93.2.2 AK_delete_constraint_not_null() . . . 576
7.93.2.3 AK_nnull_constraint_test() . . . 576
7.93.2.4 AK_read_constraint_not_null() . . . 576
7.93.2.5 AK_set_constraint_not_null() . . . 577
7.94 sql/cs/nnull.h File Reference . . . 577
7.94.1 Detailed Description . . . 578
7.94.2 Function Documentation . . . 578
7.94.2.1 AK_check_constraint_not_null() . . . 578
7.94.2.2 AK_delete_constraint_not_null() . . . 578
7.94.2.3 AK_nnull_constraint_test() . . . 579
7.94.2.4 AK_read_constraint_not_null() . . . 579
7.94.2.5 AK_set_constraint_not_null() . . . 580
7.95 sql/cs/reference.c File Reference . . . 580
7.95.1 Detailed Description . . . 581
7.95.2 Function Documentation . . . 581
7.95.2.1 AK_add_reference() . . . 581
7.95.2.2 AK_get_reference() . . . 582
7.95.2.3 AK_reference_check_attribute() . . . 582
7.95.2.4 AK_reference_check_entry() . . . 583
7.95.2.5 AK_reference_check_if_update_needed() . . . 583
7.95.2.6 AK_reference_check_restricion() . . . 583
7.95.2.7 AK_reference_test() . . . 584
7.95.2.8 AK_reference_update() . . . 584
7.96 sql/cs/reference.h File Reference . . . 585
7.96.1 Detailed Description . . . 586
7.96.2 Macro Definition Documentation . . . 587
7.96.2.1 MAX_CHILD_CONSTRAINTS . . . 587
7.96.2.2 MAX_REFERENCE_ATTRIBUTES . . . 587

7.96.2.3 REF_TYPE_CASCADE . . . 587
7.96.2.4 REF_TYPE_NO_ACTION . . . 587
7.96.2.5 REF_TYPE_NONE . . . 587
7.96.2.6 REF_TYPE_RESTRICT . . . 587
7.96.2.7 REF_TYPE_SET_DEFAULT . . . 588
7.96.2.8 REF_TYPE_SET_NULL . . . 588
7.96.3 Function Documentation . . . 588
7.96.3.1 AK_add_reference() . . . 588
7.96.3.2 AK_delete_row() . . . 589
7.96.3.3 AK_get_reference() . . . 589
7.96.3.4 AK_initialize_new_segment() . . . 589
7.96.3.5 AK_Insert_New_Element() . . . 590
7.96.3.6 AK_Insert_New_Element_For_Update() . . . 591
7.96.3.7 AK_insert_row() . . . 591
7.96.3.8 AK_reference_check_attribute() . . . 592
7.96.3.9 AK_reference_check_entry() . . . 592
7.96.3.10 AK_reference_check_if_update_needed() . . . 593
7.96.3.11 AK_reference_check_restricion() . . . 593
7.96.3.12 AK_reference_test() . . . 594
7.96.3.13 AK_reference_update() . . . 594
7.96.3.14 AK_selection() . . . 595
7.96.3.15 AK_Update_Existing_Element() . . . 595
7.96.3.16 AK_update_row() . . . 596
7.97 sql/cs/unique.c File Reference . . . 596
7.97.1 Detailed Description . . . 596
7.97.2 Function Documentation . . . 597
7.97.2.1 AK_delete_constraint_unique() . . . 597
7.97.2.2 AK_read_constraint_unique() . . . 597
7.97.2.3 AK_set_constraint_unique() . . . 598
7.97.2.4 AK_unique_test() . . . 598
7.98 sql/cs/unique.h File Reference . . . 599
7.98.1 Detailed Description . . . 599
7.98.2 Function Documentation . . . 599
7.98.2.1 AK_delete_constraint_unique() . . . 599
7.98.2.2 AK_read_constraint_unique() . . . 600
7.98.2.3 AK_set_constraint_unique() . . . 601
7.98.2.4 AK_unique_test() . . . 601
7.99 sql/drop.c File Reference . . . 601
7.99.1 Detailed Description . . . 602
7.99.2 Function Documentation . . . 602
7.99.2.1 AK_drop() . . . 602
7.99.2.2 AK_drop_help_function() . . . 603

7.99.2.3 AK_drop_test() . . . 603
7.99.2.4 AK_if_exist() . . . 603
7.99.3 Variable Documentation . . . 604
7.99.3.1 system_catalog . . . 604
7.100 sql/drop.h File Reference . . . 604
7.100.1 Detailed Description . . . 605
7.100.2 Typedef Documentation . . . 605
7.100.2.1 AK_drop_arguments . . . 605
7.100.3 Function Documentation . . . 605
7.100.3.1 AK_drop() . . . 605
7.100.3.2 AK_drop_test() . . . 606
7.100.3.3 AK_if_exist() . . . 606
7.101 sql/function.c File Reference . . . 606
7.101.1 Detailed Description . . . 607
7.101.2 Function Documentation . . . 607
7.101.2.1 AK_check_function_arguments() . . . 607
7.101.2.2 AK_check_function_arguments_type() . . . 608
7.101.2.3 AK_function_add() . . . 608
7.101.2.4 AK_function_arguments_add() . . . 609
7.101.2.5 AK_function_arguments_remove_by_obj_id() . . . 609
7.101.2.6 AK_function_change_return_type() . . . 610
7.101.2.7 AK_function_remove_by_name() . . . 610
7.101.2.8 AK_function_remove_by_obj_id() . . . 611
7.101.2.9 AK_function_rename() . . . 611
7.101.2.10 AK_function_test() . . . 612
7.101.2.11 AK_get_function_obj_id() . . . 612
7.102 sql/function.h File Reference . . . 612
7.102.1 Detailed Description . . . 613
7.102.2 Function Documentation . . . 613
7.102.2.1 AK_check_function_arguments() . . . 613
7.102.2.2 AK_check_function_arguments_type() . . . 614
7.102.2.3 AK_function_add() . . . 614
7.102.2.4 AK_function_arguments_add() . . . 615
7.102.2.5 AK_function_arguments_remove_by_obj_id() . . . 615
7.102.2.6 AK_function_change_return_type() . . . 616
7.102.2.7 AK_function_remove_by_name() . . . 616
7.102.2.8 AK_function_remove_by_obj_id() . . . 617
7.102.2.9 AK_function_rename() . . . 617
7.102.2.10 AK_function_test() . . . 618
7.102.2.11 AK_get_function_obj_id() . . . 618
7.103 sql/insert.c File Reference . . . 618
7.103.1 Function Documentation . . . 619

7.103.1.1 AK_get_insert_header() . . . 619
7.103.1.2 AK_insert() . . . 619
7.103.1.3 AK_insert_test() . . . 620
7.104 sql/insert.h File Reference . . . 620
7.104.1 Detailed Description . . . 620
7.104.2 Function Documentation . . . 620
7.104.2.1 AK_get_insert_header() . . . 621
7.104.2.2 AK_insert() . . . 621
7.104.2.3 AK_insert_test() . . . 622
7.105 sql/privileges.c File Reference . . . 622
7.105.1 Detailed Description . . . 623
7.105.2 Function Documentation . . . 623
7.105.2.1 AK_add_user_to_group() . . . 623
7.105.2.2 AK_check_group_privilege() . . . 623
7.105.2.3 AK_check_privilege() . . . 624
7.105.2.4 AK_check_user_privilege() . . . 624
7.105.2.5 AK_grant_privilege_group() . . . 625
7.105.2.6 AK_grant_privilege_user() . . . 625
7.105.2.7 AK_group_add() . . . 626
7.105.2.8 AK_group_get_id() . . . 626
7.105.2.9 AK_group_remove_by_name() . . . 627
7.105.2.10 AK_group_rename() . . . 627
7.105.2.11 AK_privileges_test() . . . 628
7.105.2.12 AK_remove_all_users_from_group() . . . 628
7.105.2.13 AK_remove_user_from_all_groups() . . . 628
7.105.2.14 AK_revoke_all_privileges_group() . . . 629
7.105.2.15 AK_revoke_all_privileges_user() . . . 629
7.105.2.16 AK_revoke_privilege_group() . . . 630
7.105.2.17 AK_revoke_privilege_user() . . . 630
7.105.2.18 AK_user_add() . . . 631
7.105.2.19 AK_user_check_pass() . . . 631
7.105.2.20 AK_user_get_id() . . . 632
7.105.2.21 AK_user_remove_by_name() . . . 632
7.105.2.22 AK_user_rename() . . . 633
7.106 sql/privileges.h File Reference . . . 633
7.106.1 Detailed Description . . . 634
7.106.2 Function Documentation . . . 634
7.106.2.1 AK_add_user_to_group() . . . 635
7.106.2.2 AK_check_group_privilege() . . . 635
7.106.2.3 AK_check_privilege() . . . 635
7.106.2.4 AK_check_user_privilege() . . . 636
7.106.2.5 AK_grant_privilege_group() . . . 636

7.106.2.6 AK_grant_privilege_user() . . . 637
7.106.2.7 AK_group_add() . . . 637
7.106.2.8 AK_group_get_id() . . . 638
7.106.2.9 AK_group_remove_by_name() . . . 638
7.106.2.10 AK_group_rename() . . . 639
7.106.2.11 AK_privileges_test() . . . 639
7.106.2.12 AK_remove_all_users_from_group() . . . 640
7.106.2.13 AK_remove_user_from_all_groups() . . . 640
7.106.2.14 AK_revoke_all_privileges_group() . . . 640
7.106.2.15 AK_revoke_all_privileges_user() . . . 641
7.106.2.16 AK_revoke_privilege_group() . . . 641
7.106.2.17 AK_revoke_privilege_user() . . . 642
7.106.2.18 AK_user_add() . . . 643
7.106.2.19 AK_user_check_pass() . . . 644
7.106.2.20 AK_user_get_id() . . . 644
7.106.2.21 AK_user_rename() . . . 644
7.107 sql/select.c File Reference . . . 645
7.107.1 Detailed Description . . . 645
7.107.2 Function Documentation . . . 646
7.107.2.1 AK_select() . . . 646
7.107.2.2 AK_select_test() . . . 646
7.108 sql/select.h File Reference . . . 646
7.108.1 Detailed Description . . . 647
7.108.2 Function Documentation . . . 647
7.108.2.1 AK_select() . . . 647
7.108.2.2 AK_select_test() . . . 647
7.109 sql/trigger.c File Reference . . . 648
7.109.1 Detailed Description . . . 648
7.109.2 Function Documentation . . . 648
7.109.2.1 AK_trigger_add() . . . 649
7.109.2.2 AK_trigger_edit() . . . 649
7.109.2.3 AK_trigger_get_conditions() . . . 650
7.109.2.4 AK_trigger_get_id() . . . 650
7.109.2.5 AK_trigger_remove_by_name() . . . 651
7.109.2.6 AK_trigger_remove_by_obj_id() . . . 651
7.109.2.7 AK_trigger_rename() . . . 652
7.109.2.8 AK_trigger_save_conditions() . . . 652
7.109.2.9 AK_trigger_test() . . . 653
7.110 sql/trigger.h File Reference . . . 653
7.110.1 Detailed Description . . . 654
7.110.2 Function Documentation . . . 654
7.110.2.1 AK_trigger_add() . . . 654

7.110.2.2 AK_trigger_edit() . . . 654
7.110.2.3 AK_trigger_get_conditions() . . . 655
7.110.2.4 AK_trigger_get_id() . . . 656
7.110.2.5 AK_trigger_remove_by_name() . . . 656
7.110.2.6 AK_trigger_remove_by_obj_id() . . . 657
7.110.2.7 AK_trigger_rename() . . . 657
7.110.2.8 AK_trigger_save_conditions() . . . 658
7.110.2.9 AK_trigger_test() . . . 658
7.111 sql/view.c File Reference . . . 658
7.111.1 Detailed Description . . . 659
7.111.2 Function Documentation . . . 659
7.111.2.1 AK_check_view_name() . . . 659
7.111.2.2 AK_get_relation_expression() . . . 660
7.111.2.3 AK_get_view_object_id() . . . 660
7.111.2.4 AK_get_view_query() . . . 661
7.111.2.5 AK_test_get_view_data() . . . 661
7.111.2.6 AK_view_add() . . . 661
7.111.2.7 AK_view_change_query() . . . 662
7.111.2.8 AK_view_remove_by_name() . . . 662
7.111.2.9 AK_view_remove_by_object_id() . . . 663
7.111.2.10 AK_view_rename() . . . 663
7.111.2.11 AK_view_test() . . . 664
7.112 sql/view.h File Reference . . . 664
7.112.1 Function Documentation . . . 665
7.112.1.1 AK_check_view_name() . . . 665
7.112.1.2 AK_get_view_query() . . . 665
7.112.1.3 AK_view_add() . . . 666
7.112.1.4 AK_view_change_query() . . . 666
7.112.1.5 AK_view_remove_by_name() . . . 667
7.112.1.6 AK_view_rename() . . . 667
7.112.1.7 AK_view_test() . . . 668
7.113 tools/comments.py File Reference . . . 668
7.113.1 Detailed Description . . . 669
7.114 tools/getFiles.sh File Reference . . . 669
7.114.1 Detailed Description . . . 669
7.115 tools/parseC.sh File Reference . . . 669
7.115.1 Detailed Description . . . 669
7.116 tools/parsePy.sh File Reference . . . 669
7.116.1 Detailed Description . . . 669
7.117 tools/updateVersion.sh File Reference . . . 669
7.117.1 Detailed Description . . . 669
7.118 trans/transaction.c File Reference . . . 669

7.118.1 Detailed Description . . . 671
7.118.2 Function Documentation . . . 671
7.118.2.1 AK_acquire_lock() . . . 672
7.118.2.2 AK_add_hash_entry_list() . . . 672
7.118.2.3 AK_add_lock() . . . 673
7.118.2.4 AK_all_transactions_finished() . . . 673
7.118.2.5 AK_create_lock() . . . 673
7.118.2.6 AK_create_new_transaction_thread() . . . 674
7.118.2.7 AK_delete_hash_entry_list() . . . 674
7.118.2.8 AK_delete_lock_entry_list() . . . 675
7.118.2.9 AK_execute_commands() . . . 675
7.118.2.10 AK_execute_transaction() . . . 676
7.118.2.11 AK_get_memory_blocks() . . . 676
7.118.2.12 AK_handle_observable_transaction_action() . . . 677
7.118.2.13 AK_init_observable_transaction() . . . 677
7.118.2.14 AK_init_observer_lock() . . . 677
7.118.2.15 AK_isLock_waiting() . . . 678
7.118.2.16 AK_lock_released() . . . 678
7.118.2.17 AK_memory_block_hash() . . . 678
7.118.2.18 AK_on_all_transactions_end() . . . 679
7.118.2.19 AK_on_lock_release() . . . 679
7.118.2.20 AK_on_observable_notify() . . . 679
7.118.2.21 AK_on_transaction_end() . . . 680
7.118.2.22 AK_release_locks() . . . 680
7.118.2.23 AK_remove_transaction_thread() . . . 681
7.118.2.24 AK_search_empty_link_for_hook() . . . 681
7.118.2.25 AK_search_existing_link_for_hook() . . . 681
7.118.2.26 AK_search_lock_entry_list_by_key() . . . 682
7.118.2.27 AK_test_Transaction() . . . 682
7.118.2.28 AK_transaction_finished() . . . 682
7.118.2.29 AK_transaction_manager() . . . 683
7.118.2.30 AK_transaction_register_observer() . . . 683
7.118.2.31 AK_transaction_unregister_observer() . . . 683
7.118.2.32 handle_transaction_notify() . . . 684
7.118.3 Variable Documentation . . . 684
7.118.3.1 accessLockMutex . . . 684
7.118.3.2 acquireLockMutex . . . 684
7.118.3.3 activeThreads . . . 685
7.118.3.4 activeTransactionsCount . . . 685
7.118.3.5 cond_lock . . . 685
7.118.3.6 endTransationTestLockMutex . . . 685
7.118.3.7 LockTable . . . 685

7.118.3.8 newTransactionLockMutex . . . 685
7.118.3.9 observable_transaction . . . 685
7.118.3.10 transactionsCount . . . 685
7.119 trans/transaction.h File Reference . . . 686
7.119.1 Detailed Description . . . 688
7.119.2 Typedef Documentation . . . 688
7.119.2.1 AK_memoryAddresses . . . 688
7.119.2.2 AK_memoryAddresses_link . . . 688
7.119.2.3 AK_observable_transaction . . . 689
7.119.2.4 AK_observer_lock . . . 689
7.119.2.5 AK_thread_Container . . . 689
7.119.2.6 AK_thread_elem . . . 689
7.119.2.7 AK_transaction_data . . . 689
7.119.2.8 AK_transaction_elem . . . 689
7.119.2.9 AK_transaction_elem_P . . . 689
7.119.2.10 AK_transaction_list . . . 689
7.119.2.11 AK_transaction_lock_elem . . . 690
7.119.2.12 AK_transaction_lock_elem_P . . . 690
7.119.3 Enumeration Type Documentation . . . 690
7.119.3.1 NoticeType . . . 690
7.119.4 Function Documentation . . . 690
7.119.4.1 AK_acquire_lock() . . . 690
7.119.4.2 AK_add_hash_entry_list() . . . 691
7.119.4.3 AK_add_lock() . . . 692
7.119.4.4 AK_all_transactions_finished() . . . 692
7.119.4.5 AK_create_lock() . . . 693
7.119.4.6 AK_create_new_transaction_thread() . . . 693
7.119.4.7 AK_delete_hash_entry_list() . . . 694
7.119.4.8 AK_delete_lock_entry_list() . . . 694
7.119.4.9 AK_execute_commands() . . . 694
7.119.4.10 AK_execute_transaction() . . . 695
7.119.4.11 AK_get_memory_blocks() . . . 696
7.119.4.12 AK_handle_observable_transaction_action() . . . 696
7.119.4.13 AK_init_observable_transaction() . . . 697
7.119.4.14 AK_init_observer_lock() . . . 697
7.119.4.15 AK_isLock_waiting() . . . 697
7.119.4.16 AK_lock_released() . . . 698
7.119.4.17 AK_memory_block_hash() . . . 698
7.119.4.18 AK_on_all_transactions_end() . . . 699
7.119.4.19 AK_on_lock_release() . . . 699
7.119.4.20 AK_on_observable_notify() . . . 699
7.119.4.21 AK_on_transaction_end() . . . 700

7.119.4.22 AK_release_locks() . . . . . 700
7.119.4.23 AK_remove_transaction_thread() . . . . . 701
7.119.4.24 AK_search_empty_link_for_hook() . . . . . 701
7.119.4.25 AK_search_existing_link_for_hook() . . . . . 701
7.119.4.26 AK_search_lock_entry_list_by_key() . . . . . 702
7.119.4.27 AK_test_Transaction() . . . . . 702
7.119.4.28 AK_transaction_finished() . . . . . 702
7.119.4.29 AK_transaction_manager() . . . . . 703
7.119.4.30 AK_transaction_register_observer() . . . . . 703
7.119.4.31 AK_transaction_unregister_observer() . . . . . 703
7.119.4.32 handle_transaction_notify() . . . . . 704

**Index** **705**

# 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.

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.

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

Check multithreading, check if it's working correctly

**Member AK_execute_commands (command ∗, int)**

Check multithreading, check if it's working correctly

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.

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.

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_sort_segment (char *srcTable, char *destTable, struct list_node *attributes)**

Make it to suport multiple sort atributes and ASC|DESC ordering

Make it to suport multiple sort atributes and ASC|DESC ordering

# Chapter 2

# Namespace Index

## 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

comments . . . . . 13

# Chapter 3

# Class Index

## 3.1 Class List

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


_dictionary_
: Dictionary object . . . 15

_file_metadata . . . 16

_notifyDetails . . . 17

AK_agg_input
: Structure that contains attributes from table header, tasks for this table and counter value . . . 18

AK_agg_value
: Structure that contains atribute name, date and aggregation task associated . . . 19

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 . . . 20

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 uninteruptable, 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 . . . 22

AK_blocktable . . . 24

AK_command_recovery_struct
: Recovery structure used to recover commands from binary file . . . 25

AK_command_struct . . . 26

AK_create_table_struct . . . 27

AK_db_cache
: Structure that defines global cache memory . . . 28

AK_debmod_state
: Global structure that holds all relevant information for the debug mode and related functionality 29

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 . . . 32

AK_mem_block
Structure that defines a block of data in memory . . . 33
AK_operand . . . 35
AK_query_mem
Structure that defines global query memory . . . 35
AK_query_mem_dict
Structure that defines global query memory for data dictionaries . . . 37
AK_query_mem_lib
Structure that defines global query memory for libraries . . . 38
AK_query_mem_result
Structure that defines global query memory for results . . . 39
AK_redo_log
Structure that defines global redo log . . . 40
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 . . . 41
AK_results
Structure used for in-memory result caching . . . 42
AK_synchronization_info
Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex) . . . 44
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 . . . 45
blocktable
Structure that defines bit status of blocks, last initialized and last allocated index . . . 46
btree_node . . . 46
bucket_elem
Structure for defining a single bucket element . . . 47
cost_eval_t
Stucture for cost estimation on relations. It contains value (number of rows in table) and data (used to store table name) . . . 48
DEBUG_LEVEL
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 . . . 49
DEBUG_TYPE
Structure for setting debug type. Divide debug information according to their type (e.g. DB modules). More modules can be aditional added to the enum. Each debug type can be easly excluded from output by setting corresponding enum element to 0 . . . 49
drop_arguments . . . 50
hash_bucket
Structure for hash bucket for table hashing . . . 50
hash_info
Structure for defining a hash info element . . . 51
intersect_attr
Structure defines intersect attribute . . . 52
list_node
Structure defines a list node . . . 53
list_structure_ad . . . 55
list_structure_add
Structure that defines linked list node for index . . . 56
main_bucket
Structure for defining main bucket for table hashing . . . 57
memoryAddresses
Structure that represents a linked list of locked addresses . . . 57
Observable
Structure that defines the functions for observable object . . . 58

observable_transaction
Structure which defines transaction observable type . . . 60
observable_transaction_struct . . . 61
Observer
Structure that defines the functions for observer object . . . 62
observer_lock
Structure which defines transaction lock observer type . . . 63
projection_att_struct
Structure that defines projection_att which is a new list_node . . . 64
root_info . . . 65
rowroot_struct
Structure that defines a new row in table using list_node . . . 65
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 . . . 66
search_result
Structure which represents search result of AK_equisearch_unsorted and AK_rangesearch_↩ unsorted . . . 67
Stack
Structure defines a Stack element. Every Stack has its Vertex pointer and pointer to next Stack in the linked list . . . 69
struct_add
Structure defining node address . . . 70
Succesor
Structure defines a Succesor element. Every Succesor has its Vertex pointer and pointer to next Succesor in the linked list . . . 71
table_addresses
Structure that defines start and end address of extent . . . 72
TestResult
Used so tests can report the amount of successful tests . . . 73
threadContainer
Structure that represents a linked list of threads.
74
transaction_list_elem
Structure that represents LockTable entry about transaction lock holder.Element indexed by Hash table . . . 75
transaction_list_head
Structure that represents LockTable entry about doubly linked list of collision in Hash table . . . 76
transaction_locks_list_elem
Structure that represents LockTable entry about transaction resource lock . . . 77
transactionData
Structure used to transport transaction data to the thread . . . 78
TypeObservable . . . 79
TypeObserver . . . 81
Vertex
Structure defines a Vertex node element. Every Vertex has its VertexId, index, lowLink and pointer to next edge and vertex . . . 81

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:


auxi/auxiliary.c . . . 83
auxi/auxiliary.h . . . 83
auxi/configuration.h . . . 105
auxi/constants.h . . . 108
auxi/debug.c . . . 130
auxi/debug.h . . . 131
auxi/dictionary.c
    Implements a dictionary for string variables . . . 134
auxi/dictionary.h
    Implements a dictionary for string variables . . . 138
auxi/iniparser.c
    Parser for ini files . . . 142
auxi/iniparser.h
    Parser for ini files . . . 153
auxi/mempro.c . . . 162
auxi/mempro.h . . . 177
auxi/observable.c . . . 194
auxi/observable.h . . . 198
auxi/test.c . . . 201
auxi/test.h . . . 208
dm/dbman.c . . . 217
dm/dbman.h . . . 235
file/blobs.c . . . 258
file/blobs.h . . . 263
file/fileio.c . . . 269
file/fileio.h . . . 275
file/files.c . . . 281
file/files.h . . . 283
file/filesearch.c . . . 285
file/filesearch.h . . . 288
file/filesort.c . . . 291
file/filesort.h . . . 294
file/id.c . . . 298
file/id.h . . . 299
file/sequence.c . . . 360

file/sequence.h . . . 364
file/table.c . . . 369
file/table.h . . . 382
file/tableOld.c . . . 396
file/tableOld.h . . . 409
file/test.c . . . 202
file/test.h . . . 213
file/idx/bitmap.c . . . 301
file/idx/bitmap.h . . . 307
file/idx/btree.c . . . 315
file/idx/btree.h . . . 322
file/idx/hash.c . . . 329
file/idx/hash.h . . . 336
file/idx/index.c . . . 343
file/idx/index.h . . . 351
mm/memoman.c . . . 422
mm/memoman.h . . . 432
opti/query_optimization.c . . . 443
opti/query_optimization.h . . . 446
opti/rel_eq_assoc.c . . . 449
opti/rel_eq_assoc.h . . . 451
opti/rel_eq_comut.c . . . 454
opti/rel_eq_comut.h . . . 456
opti/rel_eq_projection.c . . . 458
opti/rel_eq_projection.h . . . 464
opti/rel_eq_selection.c . . . 470
opti/rel_eq_selection.h . . . 475
rec/archive_log.c . . . 482
rec/archive_log.h . . . 484
rec/recovery.c . . . 486
rec/recovery.h . . . 490
rec/redo_log.c . . . 494
rec/redo_log.h . . . 496
rel/aggregation.c . . . 499
rel/aggregation.h . . . 504
rel/difference.c . . . 510
rel/difference.h . . . 511
rel/expression_check.c . . . 512
rel/expression_check.h . . . 516
rel/intersect.c . . . 519
rel/intersect.h . . . 520
rel/nat_join.c . . . 522
rel/nat_join.h . . . 525
rel/product.c . . . 528
rel/product.h . . . 530
rel/projection.c . . . 533
rel/projection.h . . . 538
rel/selection.c . . . 544
rel/selection.h . . . 546
rel/theta_join.c . . . 547
rel/theta_join.h . . . 550
rel/union.c . . . 553
rel/union.h . . . 554
sql/command.c . . . 556
sql/command.h . . . 557
sql/drop.c . . . 601
sql/drop.h . . . 604
sql/function.c . . . 606

sql/function.h . . . 612
sql/insert.c . . . 618
sql/insert.h . . . 620
sql/privileges.c . . . 622
sql/privileges.h . . . 633
sql/select.c . . . 645
sql/select.h . . . 646
sql/trigger.c . . . 648
sql/trigger.h . . . 653
sql/view.c . . . 658
sql/view.h . . . 664
sql/cs/between.c . . . 559
sql/cs/between.h . . . 562
sql/cs/check_constraint.c . . . 566
sql/cs/check_constraint.h . . . 569
sql/cs/constraint_names.c . . . 572
sql/cs/constraint_names.h . . . 573
sql/cs/nnull.c . . . 575
sql/cs/nnull.h . . . 577
sql/cs/reference.c . . . 580
sql/cs/reference.h . . . 585
sql/cs/unique.c . . . 596
sql/cs/unique.h . . . 599
tools/comments.py . . . 668
tools/getFiles.sh . . . 669
tools/parseC.sh . . . 669
tools/parsePy.sh . . . 669
tools/updateVersion.sh . . . 669
trans/transaction.c . . . 669
trans/transaction.h . . . 686

# Chapter 5

# Namespace Documentation

## 5.1 comments Namespace Reference

### Functions

- def 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 detectLanguage ()

  *Function is detecting language (is it croatian or alike) of a newly created commentsFile.*
- def makeCommentsFile ()

  *Function is parsing comments from file with .c extension and .py extension.*

### Variables

- string commentsFile = "all_comments.tmp"
- list cFiles = [ ]
- list pyFiles = [ ]

### 5.1.1 Function Documentation

#### 5.1.1.1 detectLanguage()

```
def comments.detectLanguage ( )
```

Function is detecting language (is it croatian or alike) of a newly created commentsFile.

#### 5.1.1.2 getcommentsFiles()

```
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.

#### 5.1.1.3 makeCommentsFile()

```
def comments.makeCommentsFile ( )
```

Function is parsing comments from file with .c extension and .py extension.

### 5.1.2 Variable Documentation

#### 5.1.2.1 cFiles

```
list comments.cFiles = []
```

#### 5.1.2.2 commentsFile

```
string comments.commentsFile = "all_comments.tmp"
```

#### 5.1.2.3 pyFiles

```
list comments.pyFiles = []
```

# Chapter 6

# Class Documentation

## 6.1 _dictionary_ Struct Reference

Dictionary object.

`#include <dictionary.h>`

### Public Attributes

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

### 6.1.1 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.

### 6.1.2 Member Data Documentation

#### 6.1.2.1 hash

`unsigned* _dictionary_::hash`

List of string keys

#### 6.1.2.2 key

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

List of string values

#### 6.1.2.3 n

```
int _dictionary_::n
```

#### 6.1.2.4 size

```
int _dictionary_::size
```

Number of entries in dictionary

#### 6.1.2.5 val

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

Storage size

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

- auxi/dictionary.h

## 6.2 _file_metadata Struct Reference

```
#include <blobs.h>
```

### Public Attributes

- char * new_path
- char * new_name
- char * old_path
- char * old_name
- char * checksum

### 6.2.1 Member Data Documentation

#### 6.2.1.1 checksum

```
char* _file_metadata::checksum
```

#### 6.2.1.2 new_name

```
char* _file_metadata::new_name
```

#### 6.2.1.3 new_path

```
char* _file_metadata::new_path
```

#### 6.2.1.4 old_name

```
char* _file_metadata::old_name
```

#### 6.2.1.5 old_path

```
char* _file_metadata::old_path
```

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

- file/blobs.h

## 6.3 _notifyDetails Struct Reference

### Public Attributes

- char * message
- NotifyType type

### 6.3.1 Member Data Documentation

#### 6.3.1.1 message

```
char* _notifyDetails::message
```

#### 6.3.1.2 type

```
NotifyType _notifyDetails::type
```

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

- auxi/observable.c

## 6.4 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

### 6.4.1 Detailed Description

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

**Author**

Unknown

### 6.4.2 Member Data Documentation

#### 6.4.2.1 attributes

```
AK_header AK_agg_input::attributes[MAX_ATTRIBUTES]
```

#### 6.4.2.2 counter

```
int AK_agg_input::counter
```

#### 6.4.2.3 tasks

```
int AK_agg_input::tasks[MAX_ATTRIBUTES]
```

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

- rel/aggregation.h

## 6.5 AK_agg_value Struct Reference

Structure that contains atribute 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

### 6.5.1 Detailed Description

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

**Author**

Unknown

### 6.5.2 Member Data Documentation

#### 6.5.2.1 agg_task

```
int AK_agg_value::agg_task
```

#### 6.5.2.2 att_name

```
char AK_agg_value::att_name[MAX_ATT_NAME]
```

#### 6.5.2.3 data

```
char AK_agg_value::data[MAX_VARCHAR_LENGTH]
```

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

- rel/aggregation.h

## 6.6 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*

### 6.6.1 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

## 6.6.2 Member Data Documentation

### 6.6.2.1 address

```
int AK_block::address
```

block number (address) in DB file

### 6.6.2.2 AK_free_space

```
int AK_block::AK_free_space
```

AK_free space in block.

### 6.6.2.3 chained_with

```
int AK_block::chained_with
```

address of chained block; NOT_CHAINED otherwise

### 6.6.2.4 data

```
unsigned char AK_block::data[DATA_BLOCK_SIZE *DATA_ENTRY_SIZE]
```

actual data entries

### 6.6.2.5 header

```
AK_header AK_block::header[MAX_ATTRIBUTES]
```

attribute definitions

### 6.6.2.6 last_tuple_dict_id

```
int AK_block::last_tuple_dict_id
```

#### 6.6.2.7 tuple_dict

```
AK_tuple_dict AK_block::tuple_dict[DATA_BLOCK_SIZE]
```

dictionary of data entries

#### 6.6.2.8 type

```
int AK_block::type
```

block type (can be BLOCK_TYPE_FREE, BLOCK_TYPE_NORMAL or BLOCK_TYPE_CHAINED)

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

- dm/dbman.h

## 6.7 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 uninteruptable, 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

### 6.7.1 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 uninteruptable, 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

### 6.7.2 Member Data Documentation

#### 6.7.2.1 block_lock

```
pthread_mutex_t AK_block_activity::block_lock
```

#### 6.7.2.2 locked_for_reading

```
short AK_block_activity::locked_for_reading
```

#### 6.7.2.3 locked_for_writing

```
short AK_block_activity::locked_for_writing
```

#### 6.7.2.4 reading_done

```
pthread_cond_t AK_block_activity::reading_done
```

#### 6.7.2.5 thread_holding_lock

```
int* AK_block_activity::thread_holding_lock
```

#### 6.7.2.6 writing_done

```
pthread_cond_t AK_block_activity::writing_done
```

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

- dm/dbman.h

## 6.8 AK_blocktable Struct Reference

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

### 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

### 6.8.1 Member Data Documentation

#### 6.8.1.1 allocationtable

```
unsigned int AK_blocktable::allocationtable[DB_FILE_BLOCKS_NUM_EX]
```

#### 6.8.1.2 bittable

```
unsigned char AK_blocktable::bittable[BITNSLOTS(DB_FILE_BLOCKS_NUM_EX)]
```

#### 6.8.1.3 last_allocated

```
int AK_blocktable::last_allocated
```

#### 6.8.1.4 last_initialized

```
int AK_blocktable::last_initialized
```

#### 6.8.1.5 ltime

```
time_t AK_blocktable::ltime
```

#### 6.8.1.6 prepared

```
int AK_blocktable::prepared
```

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

- dm/dbman.h

## 6.9 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]
- char condition [MAX_ATTRIBUTES][MAX_VARCHAR_LENGTH]
- int finished

### 6.9.1 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. Updated can be used to save select operation

**Author**

Tomislav Turek updated by Danko Bukovac

### 6.9.2 Member Data Documentation

#### 6.9.2.1 arguments

```
char AK_command_recovery_struct::arguments[MAX_ATTRIBUTES][MAX_VARCHAR_LENGTH]
```

#### 6.9.2.2 condition

```
char AK_command_recovery_struct::condition[MAX_ATTRIBUTES][MAX_VARCHAR_LENGTH]
```

#### 6.9.2.3 finished

```
int AK_command_recovery_struct::finished
```

#### 6.9.2.4 operation

```
int AK_command_recovery_struct::operation
```

#### 6.9.2.5 table_name

```
char AK_command_recovery_struct::table_name[MAX_VARCHAR_LENGTH]
```

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

- mm/memoman.h

## 6.10 AK_command_struct Struct Reference

```
#include <command.h>
```

### Public Attributes

- int id_command
- char * tblName
- void * parameters

### 6.10.1 Member Data Documentation

#### 6.10.1.1 id_command

```
int AK_command_struct::id_command
```

#### 6.10.1.2 parameters

```
void* AK_command_struct::parameters
```

#### 6.10.1.3 tblName

```
char* AK_command_struct::tblName
```

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

- sql/command.h

## 6.11 AK_create_table_struct Struct Reference

```
#include <table.h>
```

### Public Attributes

- char name [MAX_ATT_NAME]
- int type

### 6.11.1 Member Data Documentation

#### 6.11.1.1 name

```
char AK_create_table_struct::name
```

#### 6.11.1.2 type

```
int AK_create_table_struct::type
```

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

- file/table.h
- file/tableOld.h

## 6.12 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*

### 6.12.1 Detailed Description

Structure that defines global cache memory.

**Author**

Unknown

### 6.12.2 Member Data Documentation

#### 6.12.2.1 cache

```
AK_mem_block* AK_db_cache::cache[MAX_CACHE_MEMORY]
```

last recently read blocks

#### 6.12.2.2 next_replace

```
int AK_db_cache::next_replace
```

next cached block to be replaced (0 - MAX_CACHE_MEMORY-1); depends on caching algorithm

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

- mm/memoman.h

## 6.13 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]

### 6.13.1 Detailed Description

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

**Author**

Marin Rukavina, Mislav Bozicevic

### 6.13.2 Member Data Documentation

#### 6.13.2.1 alloc_owner

```
int32_t AK_debmod_state::alloc_owner[AK_DEBMOD_PAGES_NUM]
```

#### 6.13.2.2 dirty

```
uint8_t AK_debmod_state::dirty[AK_DEBMOD_PAGES_NUM]
```

#### 6.13.2.3 free_owner

```
int32_t AK_debmod_state::free_owner[AK_DEBMOD_PAGES_NUM]
```

#### 6.13.2.4 fstack_items

```
int32_t AK_debmod_state::fstack_items[AK_DEBMOD_STACKSIZE]
```

#### 6.13.2.5 fstack_size

```
int32_t AK_debmod_state::fstack_size
```

#### 6.13.2.6 func_used_by

```
int8_t AK_debmod_state::func_used_by[AK_DEBMOD_MAX_FUNCTIONS][AK_DEBMOD_MAX_FUNCTIONS]
```

#### 6.13.2.7 function

```
char AK_debmod_state::function[AK_DEBMOD_MAX_FUNCTIONS][AK_DEBMOD_MAX_FUNC_NAME]
```

#### 6.13.2.8 init

```
uint8_t AK_debmod_state::init
```

#### 6.13.2.9 last_function_id

```
int32_t AK_debmod_state::last_function_id
```

#### 6.13.2.10 nomi

```
uint32_t AK_debmod_state::nomi[AK_DEBMOD_PAGES_NUM]
```

#### 6.13.2.11 page

```
void* AK_debmod_state::page[AK_DEBMOD_PAGES_NUM]
```

#### 6.13.2.12 page_size

```
uint32_t AK_debmod_state::page_size
```

#### 6.13.2.13 print

```
uint8_t AK_debmod_state::print
```

#### 6.13.2.14 ready

```
uint8_t AK_debmod_state::ready
```

#### 6.13.2.15 real

```
uint32_t AK_debmod_state::real[AK_DEBMOD_PAGES_NUM]
```

#### 6.13.2.16 used

```
uint8_t AK_debmod_state::used[AK_DEBMOD_PAGES_NUM]
```

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

- auxi/mempro.h

## 6.14 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 costraints*
- char constr_name [MAX_CONSTRAINTS][MAX_CONSTR_NAME]
  *extra integrity constraint names*
- char constr_code [MAX_CONSTRAINTS][MAX_CONSTR_CODE]
  *extra integrity costraint codes*

### 6.14.1 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

### 6.14.2 Member Data Documentation

#### 6.14.2.1 att_name

```
char AK_header::att_name[MAX_ATT_NAME]
```

attribute name

#### 6.14.2.2 constr_code

```
char AK_header::constr_code[MAX_CONSTRAINTS][MAX_CONSTR_CODE]
```

extra integrity costraint codes

#### 6.14.2.3 constr_name

```
char AK_header::constr_name[MAX_CONSTRAINTS][MAX_CONSTR_NAME]
```

extra integrity constraint names

#### 6.14.2.4 integrity

```
int AK_header::integrity[MAX_CONSTRAINTS]
```

standard integrity costraints

#### 6.14.2.5 type

```
int AK_header::type
```

type of attribute

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

- dm/dbman.h

## 6.15 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*

### 6.15.1 Detailed Description

Structure that defines a block of data in memory.

**Author**

Unknown

### 6.15.2 Member Data Documentation

#### 6.15.2.1 block

```
AK_block* AK_mem_block::block
```

pointer to block from DB file

#### 6.15.2.2 dirty

```
int AK_mem_block::dirty
```

dirty bit (BLOCK_CLEAN if unchanged; BLOCK_DIRTY if changed but not yet written to file)

#### 6.15.2.3 timestamp_last_change

```
unsigned long AK_mem_block::timestamp_last_change
```

timestamp when the block has lastly been changed

#### 6.15.2.4 timestamp_read

```
unsigned long AK_mem_block::timestamp_read
```

timestamp when the block has lastly been read

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

- mm/memoman.h

## 6.16 AK_operand Struct Reference

```
#include <projection.h>
```

### Public Attributes

- char value [MAX_VARCHAR_LENGTH]
- int type

### 6.16.1 Member Data Documentation

#### 6.16.1.1 type

```
int AK_operand::type
```

#### 6.16.1.2 value

```
char AK_operand::value[MAX_VARCHAR_LENGTH]
```

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

- rel/projection.h

## 6.17 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*

## 6.17.1 Detailed Description

Structure that defines global query memory.

**Author**

Unknown

## 6.17.2 Member Data Documentation

### 6.17.2.1 dictionary

```
AK_query_mem_dict* AK_query_mem::dictionary
```

obtained data dictionaries

### 6.17.2.2 parsed

```
AK_query_mem_lib* AK_query_mem::parsed
```

parsed queries

### 6.17.2.3 result

```
AK_query_mem_result* AK_query_mem::result
```

obtained query results

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

- mm/memoman.h

## 6.18 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)*

### 6.18.1 Detailed Description

Structure that defines global query memory for data dictionaries.

**Author**

Unkown

### 6.18.2 Member Data Documentation

#### 6.18.2.1 dictionary

```
AK_tuple_dict* AK_query_mem_dict::dictionary[MAX_QUERY_DICT_MEMORY]
```

last used data dictionaries

#### 6.18.2.2 next_replace

```
int AK_query_mem_dict::next_replace
```

next dictionary to be replaced (0 - MAX_QUERY_DICT_MEMORY-1); field pointer (LIFO)

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

- mm/memoman.h

## 6.19 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)*

### 6.19.1 Detailed Description

Structure that defines global query memory for libraries.

**Author**

Unkown

### 6.19.2 Member Data Documentation

#### 6.19.2.1 next_replace

`int AK_query_mem_lib::next_replace`

next query to be replaced (0 - MAX_QUERY_LIB_MEMORY-1); field pointer (LIFO)

#### 6.19.2.2 parsed

`char AK_query_mem_lib::parsed[MAX_QUERY_LIB_MEMORY]`

last parsed queries; to be changed to more adequate data structure

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

- mm/memoman.h

## 6.20 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)*

### 6.20.1 Detailed Description

Structure that defines global query memory for results.

**Author**

Unknown

### 6.20.2 Member Data Documentation

#### 6.20.2.1 next_replace

```
int AK_query_mem_result::next_replace
```

next result to be replaced (0 - MAX_QUERY_RESULT_MEMORY-1); field pointer (LIFO)

#### 6.20.2.2 results

```
AK_results* AK_query_mem_result::results
```

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

- mm/memoman.h

## 6.21 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

### 6.21.1 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

### 6.21.2 Member Data Documentation

#### 6.21.2.1 command_recovery

```
AK_command_recovery_struct AK_redo_log::command_recovery[MAX_REDO_LOG_ENTRIES]
```

#### 6.21.2.2 number

```
int AK_redo_log::number
```

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

- mm/memoman.h

## 6.22 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

### 6.22.1 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

### 6.22.2 Member Data Documentation

#### 6.22.2.1 attributes

```
char AK_ref_item::attributes[MAX_REFERENCE_ATTRIBUTES][MAX_ATT_NAME]
```

#### 6.22.2.2 attributes_number

```
int AK_ref_item::attributes_number
```

#### 6.22.2.3 constraint

```
char AK_ref_item::constraint[MAX_VARCHAR_LENGTH]
```

#### 6.22.2.4 parent

```
char AK_ref_item::parent[MAX_ATT_NAME]
```

#### 6.22.2.5 parent_attributes

```
char AK_ref_item::parent_attributes[MAX_REFERENCE_ATTRIBUTES][MAX_ATT_NAME]
```

#### 6.22.2.6 table

```
char AK_ref_item::table[MAX_ATT_NAME]
```

#### 6.22.2.7 type

```
int AK_ref_item::type
```

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

- sql/cs/reference.h

## 6.23 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]

### 6.23.1 Detailed Description

Structure used for in-memory result caching.

**Author**

Mario Novoselec

### 6.23.2 Member Data Documentation

#### 6.23.2.1 date_created

```
char AK_results::date_created[80]
```

#### 6.23.2.2 free

```
short AK_results::free
```

#### 6.23.2.3 header

```
AK_header AK_results::header[MAX_ATTRIBUTES]
```

#### 6.23.2.4 result_block

```
AK_block* AK_results::result_block
```

#### 6.23.2.5 result_id

```
unsigned long AK_results::result_id
```

#### 6.23.2.6 result_size

```
int AK_results::result_size
```

#### 6.23.2.7 source_table

```
char* AK_results::source_table
```

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

- mm/memoman.h

## 6.24 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

### 6.24.1 Detailed Description

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

**Author**

Marko Sinko

### 6.24.2 Member Data Documentation

#### 6.24.2.1 init

```
int AK_synchronization_info::init
```

#### 6.24.2.2 ready

```
int AK_synchronization_info::ready
```

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

- auxi/auxiliary.h

## 6.25 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( *** ) )*

### 6.25.1 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

### 6.25.2 Member Data Documentation

#### 6.25.2.1 address

`int AK_tuple_dict::address`

data entry address (in AK_block->data)

#### 6.25.2.2 size

`int AK_tuple_dict::size`

data entry size (using sizeof( *** ) )

#### 6.25.2.3 type

```
int AK_tuple_dict::type
```

data entry type

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

- dm/dbman.h

## 6.26 blocktable Struct Reference

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

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

### 6.26.1 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

## 6.27 btree_node Struct Reference

```
#include <btree.h>
```

Collaboration diagram for btree_node:

### Public Attributes

- int values [B]
- struct_add pointers [B+1]

### 6.27.1 Member Data Documentation

#### 6.27.1.1 pointers

```
struct_add btree_node::pointers[B+1]
```

#### 6.27.1.2 values

```
int btree_node::values[B]
```

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

- file/idx/btree.h

## 6.28 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*

### 6.28.1 Detailed Description

Structure for defining a single bucket element.

**Author**

Unknown

### 6.28.2 Member Data Documentation

#### 6.28.2.1 add

```
struct_add bucket_elem::add
```

bucket element address values

#### 6.28.2.2 value

```
unsigned int bucket_elem::value
```

bucket element hash value

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

- file/idx/hash.h

## 6.29 cost_eval_t Struct Reference

Stucture 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]

### 6.29.1 Detailed Description

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

**Author**

Dino Laktašić

### 6.29.2 Member Data Documentation

#### 6.29.2.1 data

```
char cost_eval_t::data[MAX_VARCHAR_LENGTH]
```

#### 6.29.2.2 value

```
int cost_eval_t::value
```

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

- opti/rel_eq_assoc.h

## 6.30 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>`

### 6.30.1 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

## 6.31 DEBUG_TYPE Struct Reference

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

`#include <debug.h>`

### 6.31.1 Detailed Description

Structure for setting debug type. Divide debug information according to their type (e.g. DB modules). More modules can be aditional added to the enum. Each debug type can be easly 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

## 6.32 drop_arguments Struct Reference

`#include <drop.h>`

Collaboration diagram for drop_arguments:

### Public Attributes

- void * value
- struct drop_arguments * next

### 6.32.1 Member Data Documentation

#### 6.32.1.1 next

```
struct drop_arguments* drop_arguments::next
```

#### 6.32.1.2 value

```
void* drop_arguments::value
```

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

- sql/drop.h

## 6.33 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*

### 6.33.1 Detailed Description

Structure for hash bucket for table hashing.

**Author**

Unknown

### 6.33.2 Member Data Documentation

#### 6.33.2.1 bucket_level

```
int hash_bucket::bucket_level
```

hash bucket level

#### 6.33.2.2 element

```
bucket_elem hash_bucket::element[HASH_BUCKET_SIZE]
```

hash bucket array of bucket_elem elements

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

- file/idx/hash.h

## 6.34 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*

### 6.34.1 Detailed Description

Structure for defining a hash info element.

**Author**

Unknown

### 6.34.2 Member Data Documentation

#### 6.34.2.1 hash_bucket_num

```
int hash_info::hash_bucket_num
```

hash bucket number

#### 6.34.2.2 main_bucket_num

```
int hash_info::main_bucket_num
```

bucket number

#### 6.34.2.3 modulo

```
int hash_info::modulo
```

modulo value for hash function

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

- file/idx/hash.h

## 6.35 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*

### 6.35.1 Detailed Description

Structure defines intersect attribute.

**Author**

Dino Laktašić

### 6.35.2 Member Data Documentation

#### 6.35.2.1 att_name

```
char intersect_attr::att_name[MAX_ATT_NAME]
```

attribute name

#### 6.35.2.2 type

```
int intersect_attr::type
```

type of attribute

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

- rel/intersect.h

## 6.36 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*

### 6.36.1 Detailed Description

Structure defines a list node.

**Author**

Ljiljana Pintarić

### 6.36.2 Member Data Documentation

#### 6.36.2.1 attribute_name

```
char list_node::attribute_name[MAX_ATT_NAME]
```

#### 6.36.2.2 constraint

```
int list_node::constraint
```

#### 6.36.2.3 data

```
char list_node::data[MAX_VARCHAR_LENGTH]
```

loaded data

#### 6.36.2.4 next

```
struct list_node* list_node::next
```

pointer to next element

#### 6.36.2.5 size

```
int list_node::size
```

#### 6.36.2.6 table

```
char list_node::table[MAX_ATT_NAME]
```

#### 6.36.2.7 type

```
int list_node::type
```

TODO - type, attribute name, table staviti na početak polja data data type.

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

- auxi/auxiliary.h

## 6.37 list_structure_ad Struct Reference

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

Collaboration diagram for list_structure_ad:

### Public Attributes

- char * attName
  *attribute name*
- struct_add add
  *addresses*
- struct list_structure_ad * next
  *next node pointer*

### 6.37.1 Member Data Documentation

#### 6.37.1.1 add

```
struct_add list_structure_ad::add
```

addresses

#### 6.37.1.2 attName

```
char* list_structure_ad::attName
```

attribute name

#### 6.37.1.3 next

```
struct list_structure_ad* list_structure_ad::next
```

next node pointer

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

- file/idx/index.h

## 6.38 list_structure_add Struct Reference

Structure that defines linked list node for index.

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

### 6.38.1 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

## 6.39 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*

### 6.39.1 Detailed Description

Structure for defining main bucket for table hashing.

**Author**

Unknown

### 6.39.2 Member Data Documentation

#### 6.39.2.1 element

```
bucket_elem main_bucket::element[MAIN_BUCKET_SIZE]
```

main bucket array of bucket_elem elements

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

- file/idx/hash.h

## 6.40 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

### 6.40.1 Detailed Description

Structure that represents a linked list of locked addresses.

**Author**

Frane Jakelić

### 6.40.2 Member Data Documentation

#### 6.40.2.1 adresa

```
int memoryAddresses::adresa
```

#### 6.40.2.2 nextElement

```
struct memoryAddresses* memoryAddresses::nextElement
```

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

- trans/transaction.h

## 6.41 Observable Struct Reference

Structure that defines the 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)

### 6.41.1 Detailed Description

Structure that defines the functions for observable object.

**Author**

Ivan Pusic

### 6.41.2 Member Data Documentation

#### 6.41.2.1 AK_destroy_observable

```
int(* Observable::AK_destroy_observable) (struct Observable *)
```

#### 6.41.2.2 AK_get_observer_by_id

```
AK_observer*(* Observable::AK_get_observer_by_id) (struct Observable *, int id)
```

#### 6.41.2.3 AK_notify_observer

```
int(* Observable::AK_notify_observer) (struct Observable *, AK_observer *)
```

#### 6.41.2.4 AK_notify_observers

```
int(* Observable::AK_notify_observers) (struct Observable *)
```

#### 6.41.2.5 AK_observable_type

```
void* Observable::AK_observable_type
```

#### 6.41.2.6 AK_ObservableType_Def

```
int Observable::AK_ObservableType_Def
```

#### 6.41.2.7 AK_register_observer

```
int(* Observable::AK_register_observer) (struct Observable *, AK_observer *)
```

#### 6.41.2.8 AK_run_custom_action

```
int(* Observable::AK_run_custom_action) (void *)
```

#### 6.41.2.9 AK_unregister_observer

```
int(* Observable::AK_unregister_observer) (struct Observable *, AK_observer *)
```

#### 6.41.2.10 observer_id_counter

```
int Observable::observer_id_counter
```

#### 6.41.2.11 observers

```
AK_observer* Observable::observers[MAX_OBSERVABLE_OBSERVERS]
```

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

- auxi/observable.h

## 6.42 observable_transaction Struct Reference

Structure which defines transaction observable type.

`#include <transaction.h>`

### 6.42.1 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

# 6.43 observable_transaction_struct Struct Reference

`#include <transaction.h>`

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

## 6.43.1 Member Data Documentation

### 6.43.1.1 AK_all_transactions_finished

```
void(* observable_transaction_struct::AK_all_transactions_finished) ()
```

### 6.43.1.2 AK_lock_released

```
void(* observable_transaction_struct::AK_lock_released) ()
```

### 6.43.1.3 AK_transaction_finished

```
void(* observable_transaction_struct::AK_transaction_finished) ()
```

### 6.43.1.4 AK_transaction_register_observer

```
int(* observable_transaction_struct::AK_transaction_register_observer) (struct observable_transaction_struct
*, AK_observer *)
```

#### 6.43.1.5 AK_transaction_unregister_observer

```
int(* observable_transaction_struct::AK_transaction_unregister_observer) (struct observable_transaction_struct
*, AK_observer *)
```

#### 6.43.1.6 observable

```
AK_observable* observable_transaction_struct::observable
```

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

- trans/transaction.h

## 6.44 Observer Struct Reference

Structure that defines the 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 *)

### 6.44.1 Detailed Description

Structure that defines the functions for observer object.

**Author**

Ivan Pusic

### 6.44.2 Member Data Documentation

#### 6.44.2.1 AK_destroy_observer

```
int(* Observer::AK_destroy_observer) (struct Observer *)
```

#### 6.44.2.2 AK_notify

```
int(* Observer::AK_notify) (struct Observer *, void *observable_type, AK_ObservableType_Enum)
```

#### 6.44.2.3 AK_observer_type

```
void* Observer::AK_observer_type
```

#### 6.44.2.4 AK_observer_type_event_handler

```
void(* Observer::AK_observer_type_event_handler) (void *, void *, AK_ObservableType_Enum)
```

#### 6.44.2.5 observer_id

```
int Observer::observer_id
```

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

- auxi/observable.h

## 6.45 observer_lock Struct Reference

Structure which defines transaction lock observer type.

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

Collaboration diagram for observer_lock:

### Public Attributes

- AK_observer * observer

### 6.45.1 Detailed Description

Structure which defines transaction lock observer type.

**Author**

Ivan Pusic

### 6.45.2 Member Data Documentation

#### 6.45.2.1 observer

```
AK_observer* observer_lock::observer
```

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

- trans/transaction.h

## 6.46 projection_att_struct Struct Reference

Structure that defines projection_att which is a new list_node.

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

Collaboration diagram for projection_att_struct:

### Public Attributes

- struct list_node * projection_att

### 6.46.1 Detailed Description

Structure that defines projection_att which is a new list_node.

**Author**

Ena Dujak

### 6.46.2 Member Data Documentation

#### 6.46.2.1 projection_att

```
struct list_node* projection_att_struct::projection_att
```

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

- rel/aggregation.h

## 6.47 root_info Struct Reference

`#include <btree.h>`

### Public Attributes

- int root
- int level [ORDER]

### 6.47.1 Member Data Documentation

#### 6.47.1.1 level

`int root_info::level[ORDER]`

#### 6.47.1.2 root

`int root_info::root`

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

- file/idx/btree.h

## 6.48 rowroot_struct Struct Reference

Structure that defines a new row in table using list_node.

`#include <aggregation.h>`

Collaboration diagram for rowroot_struct:

### Public Attributes

- struct list_node * row_root

### 6.48.1 Detailed Description

Structure that defines a new row in table using list_node.

**Author**

Ena Dujak

### 6.48.2 Member Data Documentation

#### 6.48.2.1 row_root

```
struct list_node* rowroot_struct::row_root
```

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

- rel/aggregation.h

## 6.49 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_rangesearch_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*

### 6.49.1 Detailed Description

Structure that contains attribute name, lower and upper data value, special(NULL or ∗) which is input for AK_↩ equisearch_unsorted and AK_rangesearch_unsorted.

**Author**

Unknown

### 6.49.2 Member Data Documentation

#### 6.49.2.1 iSearchType

```
int search_params::iSearchType
```

if searching for NULL values, set to SEARCH_NULL, all values -> SEARCH_ALL, particular value -> SEARCH↩ _PARTICULAR, range of values -> SEARCH_RANGE

#### 6.49.2.2 pData_lower

```
void* search_params::pData_lower
```

pointer to lower value of search range

#### 6.49.2.3 pData_upper

```
void* search_params::pData_upper
```

pointer to upper value of search range

#### 6.49.2.4 szAttribute

```
char* search_params::szAttribute
```

name of attribute

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

- file/filesearch.h

## 6.50 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*

### 6.50.1 Detailed Description

Structure which represents search result of AK_equisearch_unsorted and AK_rangesearch_unsorted.

**Author**

Unknown

### 6.50.2 Member Data Documentation

#### 6.50.2.1 aiBlocks

```
int* search_result::aiBlocks
```

array of blocks to which the tuple addresses are relative

#### 6.50.2.2 aiSearch_attributes

```
int* search_result::aiSearch_attributes
```

array of indexes of searched-for attributes

#### 6.50.2.3 aiTuple_addresses

```
int* search_result::aiTuple_addresses
```

array of tuple addresses

#### 6.50.2.4 iNum_search_attributes

```
int search_result::iNum_search_attributes
```

number of searched-for attributes in array

#### 6.50.2.5 iNum_tuple_addresses

```
int search_result::iNum_tuple_addresses
```

number of tuple addresses/blocks in corresponding arrays

#### 6.50.2.6 iNum_tuple_attributes

```
int search_result::iNum_tuple_attributes
```

number of attributes in tuple

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

- file/filesearch.h

## 6.51 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

### 6.51.1 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

### 6.51.2 Member Data Documentation

#### 6.51.2.1 link

```
struct Vertex* Stack::link
```

#### 6.51.2.2 nextElement

```
struct Stack* Stack::nextElement
```

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

- auxi/auxiliary.h

## 6.52 struct_add Struct Reference

Structure defining node address.

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

### Public Attributes

- int addBlock
  *block address*
- int indexTd
  *index table destination*

### 6.52.1 Detailed Description

Structure defining node address.

**Author**

Unknown

### 6.52.2 Member Data Documentation

#### 6.52.2.1 addBlock

```
int struct_add::addBlock
```

block address

#### 6.52.2.2 indexTd

```
int struct_add::indexTd
```

index table destination

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

- file/idx/index.h

## 6.53 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

### 6.53.1 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

### 6.53.2 Member Data Documentation

#### 6.53.2.1 link

```
struct Vertex* Succesor::link
```

#### 6.53.2.2 nextSuccesor

```
struct Succesor* Succesor::nextSuccesor
```

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

- auxi/auxiliary.h

## 6.54 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]

### 6.54.1 Detailed Description

Structure that defines start and end address of extent.

**Author**

Matija Novak

### 6.54.2 Member Data Documentation

#### 6.54.2.1 address_from

```
int table_addresses::address_from[MAX_EXTENTS_IN_SEGMENT]
```

sturcture for extents start end stop adresses

#### 6.54.2.2 address_to

```
int table_addresses::address_to[MAX_EXTENTS_IN_SEGMENT]
```

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

- dm/dbman.h

## 6.55 TestResult Struct Reference

Used so tests can report the amount of successful tests.

```
#include <test.h>
```

### Public Attributes

- int testSucceded
- int testFailed
- char implemented

### 6.55.1 Detailed Description

Used so tests can report the amount of successful tests.

This structure is used so tests can report the amount of successful tests.

**Author**

Igor Rinkovec

### 6.55.2 Member Data Documentation

#### 6.55.2.1 implemented

```
char TestResult::implemented
```

#### 6.55.2.2 testFailed

```
int TestResult::testFailed
```

#### 6.55.2.3 testSucceded

```
int TestResult::testSucceded
```

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

- auxi/test.h

## 6.56 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

### 6.56.1 Detailed Description

Structure that represents a linked list of threads.

**Author**

Frane Jakelić

### 6.56.2 Member Data Documentation

#### 6.56.2.1 nextThread

```
struct threadContainer* threadContainer::nextThread
```

#### 6.56.2.2 thread

```
pthread_t threadContainer::thread
```

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

- trans/transaction.h

## 6.57 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

### 6.57.1 Detailed Description

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

**Author**

Frane Jakelić

### 6.57.2 Member Data Documentation

#### 6.57.2.1 address

```
int transaction_list_elem::address
```

#### 6.57.2.2 DLLLocksHead

```
struct transaction_locks_list_elem* transaction_list_elem::DLLLocksHead
```

#### 6.57.2.3 isWaiting

```
int transaction_list_elem::isWaiting
```

#### 6.57.2.4 lock_type

```
int transaction_list_elem::lock_type
```

#### 6.57.2.5 nextBucket

```
struct transaction_list_elem* transaction_list_elem::nextBucket
```

#### 6.57.2.6 observer_lock

```
AK_observer_lock* transaction_list_elem::observer_lock
```

#### 6.57.2.7 prevBucket

```
struct transaction_list_elem* transaction_list_elem::prevBucket
```

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

- trans/transaction.h

## 6.58 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

### 6.58.1 Detailed Description

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

**Author**

Frane Jakelić

### 6.58.2 Member Data Documentation

#### 6.58.2.1 DLLHead

```
struct transaction_list_elem* transaction_list_head::DLLHead
```

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

- trans/transaction.h

## 6.59 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

### 6.59.1 Detailed Description

Structure that represents LockTable entry about transaction resource lock.

**Author**

Frane Jakelić

### 6.59.2 Member Data Documentation

#### 6.59.2.1 isWaiting

```
int transaction_locks_list_elem::isWaiting
```

#### 6.59.2.2 lock_type

```
int transaction_locks_list_elem::lock_type
```

#### 6.59.2.3 nextLock

```
struct transaction_locks_list_elem* transaction_locks_list_elem::nextLock
```

#### 6.59.2.4 prevLock

```
struct transaction_locks_list_elem* transaction_locks_list_elem::prevLock
```

#### 6.59.2.5 TransactionId

```
pthread_t transaction_locks_list_elem::TransactionId
```

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

- trans/transaction.h

## 6.60 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

### 6.60.1 Detailed Description

Structure used to transport transaction data to the thread.

**Author**

Frane Jakelić

### 6.60.2 Member Data Documentation

#### 6.60.2.1 array

```
command* transactionData::array
```

#### 6.60.2.2 lengthOfArray

```
int transactionData::lengthOfArray
```

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

- trans/transaction.h

## 6.61 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

### 6.61.1 Member Data Documentation

#### 6.61.1.1 AK_custom_register_observer

```
int(* TypeObservable::AK_custom_register_observer) (struct TypeObservable *, AK_observer *)
```

#### 6.61.1.2 AK_custom_unregister_observer

```
int(* TypeObservable::AK_custom_unregister_observer) (struct TypeObservable *, AK_observer *)
```

#### 6.61.1.3 AK_get_message

```
char*(* TypeObservable::AK_get_message) (struct TypeObservable *)
```

#### 6.61.1.4 AK_set_notify_info_details

```
void(* TypeObservable::AK_set_notify_info_details) (struct TypeObservable *, NotifyType type,
char *message)
```

#### 6.61.1.5 notifyDetails

```
NotifyDetails* TypeObservable::notifyDetails
```

#### 6.61.1.6 observable

```
AK_observable* TypeObservable::observable
```

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

- auxi/observable.c

## 6.62 TypeObserver Struct Reference

Collaboration diagram for TypeObserver:

### Public Attributes

- AK_TypeObservable * observable
- AK_observer * observer

### 6.62.1 Member Data Documentation

#### 6.62.1.1 observable

```
AK_TypeObservable* TypeObserver::observable
```

#### 6.62.1.2 observer

```
AK_observer* TypeObserver::observer
```

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

- auxi/observable.c

## 6.63 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

### 6.63.1 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

### 6.63.2 Member Data Documentation

#### 6.63.2.1 index

```
int Vertex::index
```

#### 6.63.2.2 lowLink

```
int Vertex::lowLink
```

#### 6.63.2.3 nextSuccesor

```
struct Succesor* Vertex::nextSuccesor
```

#### 6.63.2.4 nextVertex

```
struct Vertex* Vertex::nextVertex
```

#### 6.63.2.5 vertexId

```
int Vertex::vertexId
```

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

- auxi/auxiliary.h

# Chapter 7

# File Documentation

## 7.1 auxi/auxiliary.c File Reference

```
#include "auxiliary.h"
```
Include dependency graph for auxiliary.c:

## 7.2 auxi/auxiliary.h File Reference

```
#include "constants.h"
#include "configuration.h"
#include "test.h"
#include "assert.h"
#include "time.h"
#include "string.h"
#include "ctype.h"
#include "debug.h"
#include "mempro.h"
```
Include dependency graph for auxiliary.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct list_node

  *Structure defines a list node.*
- struct Vertex

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

  *Structure defines a Succesor element. Every Succesor has its Vertex pointer and pointer to next Succesor 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 Succesor * AK_succesor
- typedef struct Vertex * AK_vertex
- typedef struct Stack * AK_stack
- typedef struct Stack AK_stackHead

## Functions

- char * AK_convert_type (char *arg_type)

  *Function that change type of argument from string to integer.*
- int AK_strcmp (const void *a, const void *b)

  *Function compares two Strings.*
- void AK_define_tarjan_graph ()

  *Function for creating graph for testing tarjan algorithm.*
- int AK_chars_num_from_number (int number, int base)

  *Function that gets the number of digits for any given number.*
- size_t AK_type_size (int iDB_type, char *szVarchar)

  *Function returns the size in bytes for the provided database type.*
- void AK_Init_L3 (struct list_node **L)

  *Function that initializes an empty list.*
- struct list_node * AK_First_L2 (struct list_node *L)

  *Function that fetches the first element of the list.*
- struct list_node * AK_End_L2 (struct list_node *L)

  *Function that fetches the last element of the list.*
- struct list_node * AK_Next_L2 (struct list_node *current)

  *Function that fetches the next element of the list.*
- struct list_node * AK_Previous_L2 (struct list_node *current, struct list_node *L)

  *Function that fetches the previous element of the list.*
- unsigned int AK_IsEmpty_L2 (struct list_node *L)

  *Function that tests if the list is empty.*
- void AK_InsertBefore_L2 (int type, char *data, int size, struct list_node **current, struct list_node **L)

  *Function that inserts a new element before the current element of the list.*
- void AK_InsertAfter_L2 (int type, char *data, int size, struct list_node **current, struct list_node **L)

  *Function that inserts a new element after the current element of the list.*
- void AK_InsertAtBegin_L3 (int type, char *data, int size, struct list_node *L)

  *Function that inserts a new element at the beginning of the list. It uses function called: AK_InsertBefore_L.*
- void AK_InsertAtEnd_L3 (int type, char *data, int size, struct list_node *L)

  *Function that inserts a new element at the end of the list. It uses a function called: AK_InsertAfter_L2.*
- void AK_Delete_L3 (struct list_node **current, struct list_node **L)

  *Function that deletes the current element of the list.*
- void AK_DeleteAll_L3 (struct list_node **L)

*Function that empties the list.*

- int AK_Size_L2 (struct list_node *L)

  *Function that fetches the number of the elements in the list.*
- char * AK_Retrieve_L2 (struct list_node *current, struct list_node *L)

  *Function that retrieves the data from the current element of the list.*
- struct list_node * AK_GetNth_L2 (int pos, struct list_node *row)

  *Function that fetches the nth element in a row.*
- char * AK_get_array_perms (char *arr)

  *Get all permutations without repetition (currently not used, but it can be helpful)*
- AK_vertex AK_search_vertex (int id)

  *Function that searches for a specific graph node by its ID.*
- AK_vertex AK_search_empty_link ()

  *Looks for empty link for a new graph node.*
- AK_vertex AK_add_vertex (int id)

  *Function that adds a new graph node.*
- AK_succesor AK_add_succesor (int succesorId, int succesorOf)

  *Creates an 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.*
- TestResult AK_tarjan_test ()

  *Function for testing Tarjan's algorithm.*
- AK_synchronization_info * AK_init_critical_section ()

  *Initializes an AK_synchronization_info structure and returns an owned pointer that must later be passed on 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)!*

### 7.2.1 Detailed Description

Header file that provides a data structure for the auxiliary functions

## 7.2.2 Macro Definition Documentation

### 7.2.2.1 MAX_LOOP_ITERATIONS

```
#define MAX_LOOP_ITERATIONS 1000
```

### 7.2.2.2 TBL_BOX_OFFSET

```
#define TBL_BOX_OFFSET 1
```

## 7.2.3 Typedef Documentation

### 7.2.3.1 AK_graph

```
typedef struct Vertex AK_graph
```

### 7.2.3.2 AK_list

```
typedef struct list_node AK_list
```

### 7.2.3.3 AK_list_elem

```
typedef struct list_node* AK_list_elem
```

### 7.2.3.4 AK_stack

```
typedef struct Stack* AK_stack
```

#### 7.2.3.5 AK_stackHead

```
typedef struct Stack AK_stackHead
```

#### 7.2.3.6 AK_succesor

```
typedef struct Succesor* AK_succesor
```

#### 7.2.3.7 AK_vertex

```
typedef struct Vertex* AK_vertex
```

### 7.2.4 Function Documentation

#### 7.2.4.1 AK_add_succesor()

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

Creates an edge between two nodes.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *succesorId* | id of a newly created edge |
| *succesorOf* | source of the newly created edge |

**Returns**

pointer to the newly created edge

#### 7.2.4.2 AK_add_vertex()

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

Function that adds a new graph node.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *id* | of the vertex that needs to be added |
| *graphRoot* | root node of the graph structure |

**Returns**

pointer to the newly created node

#### 7.2.4.3 AK_chars_num_from_number()

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

Function that gets the number of digits for any given number.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *number* | number to evaluate |
| *int* | base mathematic base (e.g. 2, 10 etc.) |

**Returns**

the number of digits for the given number

#### 7.2.4.4 AK_convert_type()

```
char* AK_convert_type (
            char * arg_type )
```

Function that change type of argument from string to integer.

**Author**

Aleksandra Polak

**Parameters**

| | |
|---|---|
| *∗arg_type* | type of an argument |

**Returns**

EXIT_SUCCESS of the function (return type of argument in value of integer) or EXIT_ERROR

Function that change type of argument from string to integer.

**Author**

Aleksandra Polak

**Parameters**

| | |
|---|---|
| *∗arg_type* | type of argument |

**Returns**

EXIT_SUCCESS of the function (return type of argument as a value of the integer) or EXIT_ERROR

#### 7.2.4.5 AK_define_tarjan_graph()

```
void AK_define_tarjan_graph ( )
```

Function for creating graph for testing tarjan algorithm.

**Author**

Blaž Rajič

**Parameters**

| | |
|---|---|
| *graph* | AK_graph where graph will be created |

**Author**

Blaž Rajič

#### 7.2.4.6 AK_Delete_L3()

```
void AK_Delete_L3 (
            struct list_node ** current,
            struct list_node ** L )
```

Function that deletes the current element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *current* | current element of the list |
| *L* | root of the list @retrun No return value |

#### 7.2.4.7 AK_DeleteAll_L3()

```
void AK_DeleteAll_L3 (
            struct list_node ** L )
```

Function that empties the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *L* | root of the list |

**Returns**

No return value

#### 7.2.4.8 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**

| | |
|---|---|
| *info* | Synchronization info structure |

**Returns**

void

#### 7.2.4.9 AK_End_L2()

```
struct list_node* AK_End_L2 (
            struct list_node * L )
```

Function that fetches the last element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *L* | root of the list |

**Returns**

last element of the list

#### 7.2.4.10 AK_enter_critical_section()

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

Enters a critical section.

**Author**

Marko Sinko

**Parameters**

| | |
|---|---|
| *info* | Synchronization info structure |

**Returns**

void

#### 7.2.4.11 AK_First_L2()

```
struct list_node* AK_First_L2 (
            struct list_node * L )
```

Function that fetches the first element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *L* | root of the list |

**Returns**

first element of the list

#### 7.2.4.12 AK_get_array_perms()

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

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

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *arr* | array of chars to perform permutation on |

**Returns**

char pointer to an array of pointers pointing to permuted char arrays

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

**Author**

Matija Novak

**Parameters**

| | |
|---|---|
| *SearchElement* | element whose posititon we search for |
| *L* | 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**

| | |
|---|---|
| *arr* | array of chars to perform permutation on |

**Returns**

char pointer to an array of pointers pointing to permuted char arrays

#### 7.2.4.13 AK_GetNth_L2()

```
struct list_node* AK_GetNth_L2 (
            int pos,
            struct list_node * row )
```

Function that fetches the nth element in a row.

**Author**

Ljiljana Pintarić

**Parameters**

| | |
|---|---|
| *pos* | position of element in a row |
| *row* | list of elements of a row in the table |

**Returns**

element of list of elements of a row in the table

Function that fetches the nth element in a row.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *current* | current list element |
| *L* | root of the list |

**Returns**

data type of the current list element

**Author**

Matija Šestak.

Function that fetches the data size of the element

**Parameters**

| | |
|---|---|
| *current* | current list element |
| *L* | - root of the list |

**Returns**

data size of the current list element

**Author**

Ljiljana Pintarić

Function that fetches the nth element in a row

**Parameters**

| | |
|---|---|
| *pos* | position of element in a row |
| *row* | list of elements of a row in the table |

**Returns**

element of list of elements of a row in the table

#### 7.2.4.14 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 on to AK_destroy_critical_section.

**Author**

Marko Sinko

**Returns**

Initialized synchronization object

#### 7.2.4.15 AK_Init_L3()

```
void AK_Init_L3 (
            struct list_node ** L )
```

Function that initializes an empty list.

**Author**

Ljiljana Pintarić

**Parameters**

| | |
|---|---|
| *L* | root of the list |

**Returns**

NO return value

#### 7.2.4.16 AK_InsertAfter_L2()

```
void AK_InsertAfter_L2 (
            int type,
            char * data,
            int size,
            struct list_node ** current,
            struct list_node ** L )
```

Function that inserts a new element after the current element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *data* | new data |
| *current* | current element of the list |
| *L* | root of the list |

**Returns**

No return value.

#### 7.2.4.17 AK_InsertAtBegin_L3()

```
void AK_InsertAtBegin_L3 (
            int type,
            char * data,
            int size,
            struct list_node * L )
```

Function that inserts a new element at the beginning of the list. It uses function called: AK_InsertBefore_L.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *data* | new data |
| *L* | root of the list |

**Returns**

No return value

#### 7.2.4.18 AK_InsertAtEnd_L3()

```
void AK_InsertAtEnd_L3 (
            int type,
            char * data,
            int size,
            struct list_node * L )
```

Function that inserts a new element at the end of the list. It uses a function called: AK_InsertAfter_L2.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *data* | new data |
| *L* | root of the list |

**Returns**

No return value.

#### 7.2.4.19 AK_InsertBefore_L2()

```
void AK_InsertBefore_L2 (
            int type,
            char * data,
            int size,
            struct list_node ** current,
            struct list_node ** L )
```

Function that inserts a new element before the current element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *data* | new data |
| *current* | current element of the list |
| *L* | root of the list |

**Returns**

No return value

#### 7.2.4.20 AK_IsEmpty_L2()

```
unsigned int AK_IsEmpty_L2 (
            struct list_node * L )
```

Function that tests if the list is empty.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *L* | root of the list |

**Returns**

1 if the list is empty, otherwise returns 0

#### 7.2.4.21 AK_leave_critical_section()

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

Leaves a critical section.

**Author**

Marko Sinko

**Parameters**

| | |
|---|---|
| *info* | Synchronization info structure |

**Returns**

void

#### 7.2.4.22 AK_Next_L2()

```
struct list_node* AK_Next_L2 (
            struct list_node * current )
```

Function that fetches the next element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *current* | current element of the list |

**Returns**

next element of the list

#### 7.2.4.23 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

#### 7.2.4.24 AK_Previous_L2()

```
struct list_node* AK_Previous_L2 (
            struct list_node * current,
            struct list_node * L )
```

Function that fetches the previous element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *current* | current element of the list |
| *L* | root of the list |

**Returns**

previous element of the list

#### 7.2.4.25 AK_push_to_stack()

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

Adds a entry to the stack.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *id* | of the element that is being added to the stack |

**Returns**

pointer to the newly added stack node

#### 7.2.4.26 AK_Retrieve_L2()

```
char* AK_Retrieve_L2 (
            struct list_node * current,
            struct list_node * L )
```

Function that retrieves the data from the current element of the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *current* | current element of the list |
| *L* | root of the list |

**Returns**

data from the list element

#### 7.2.4.27 AK_search_empty_link()

```
AK_vertex AK_search_empty_link ( )
```

Looks for empty link for a new graph node.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *graphRoot* | oot node of the graph structure |

**Returns**

empty link for a new graph node

#### 7.2.4.28 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**

| | |
|---|---|
| *stackRoot* | root node of the selected stack |

**Returns**

pointer to the empty link

#### 7.2.4.29 AK_search_in_stack()

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

Finds an element in the stack.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *id* | of the node that needs to be found in the stack |

**Returns**

pointer to the found stack node

#### 7.2.4.30 AK_search_vertex()

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

Function that searches for a specific graph node by its ID.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *id* | of the vertex that needs to be found |
| *graphRoot* | root node of the graph structure |

**Returns**

found graph nod or null

#### 7.2.4.31 AK_Size_L2()

```
int AK_Size_L2 (
            struct list_node * L )
```

Function that fetches the number of the elements in the list.

**Author**

Ljiljana Pintarić.

**Parameters**

| | |
|---|---|
| *L* | root of the list |

**Returns**

Size of the list

#### 7.2.4.32 AK_strcmp()

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

Function compares two Strings.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *∗a* | pointer of a value to compare |
| *∗b* | pointer of a value to compare |

**Returns**

result of the comparison in line with strcmp function

#### 7.2.4.33 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**

| | |
|---|---|
| *id* | of the element on which the algorithm looks for an id of a strongly connected component |

**Author**

Frane Jakelić, updated by Blaž Rajič

**Parameters**

| | |
|---|---|
| *id* | of the element on which the algorithm looks for an id of a strongly connected component |

#### 7.2.4.34 AK_tarjan_test()

```
TestResult AK_tarjan_test ( )
```

Function for testing Tarjan's algorithm.

**Author**

Blaž Rajič

**Returns**

No return value

#### 7.2.4.35 AK_type_size()

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

Function returns the size in bytes for the provided database type.

**Author**

Miroslav Policki

**Parameters**

| | |
|---|---|
| *iDB_type* | database data type (defined in constants.h) |
| *szVarchar* | if iDB_type == TYPE_VARCHAR, pointer to the string, otherwise unused |

**Returns**

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

#### 7.2.4.36 MIN()

```
int MIN (
            int X,
            int Y )
```

### 7.2.5 Variable Documentation

#### 7.2.5.1 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

## 7.3 auxi/configuration.h File Reference

`#include "iniparser.h"`
Include dependency graph for configuration.h: This graph shows which files directly or indirectly include this file:

### Macros

- #define AK_BLOBS_PATH (iniparser_getstring(AK_config, "general:blobs_folder", "./blobs"))

  *Constant declaring the path of blobs folder (note: if changed keep in mind for make clean in makefile). Path declared in config.ini has to be absolute (tied up with installation package), but for debugging purpose we are going to keep it relative.*
- #define DB_FILE (iniparser_getstring(AK_config,"general:db_file","kalashnikov.db"))
- #define MAX_NUM_OF_BLOCKS (iniparser_getint(AK_config,"segments:max_num_of_blocks",200))

  *Constant declaring the maximum number of blocks in a segment.*
- #define MAX_EXTENTS_IN_SEGMENT 200

  *Constant declaring the maximum number of extents in segment.*
- #define MAX_FREE_SPACE_SIZE (iniparser_getint(AK_config,"blocks:max_AK_free_space_size",4000))

  *Constant declaring the maximum free space in block.*
- #define MAX_LAST_TUPLE_DICT_SIZE_TO_USE (iniparser_getint(AK_config,"dictionary:max_last_↩ tuple_dict_size_to_use",470))

  *Constant declaring the maximum size od last tuple in a dictionary.*
- #define DB_FILE_SIZE (iniparser_getint(AK_config,"general:db_file_size",40))

  *Constant declaring size of DB file in MB.*
- #define DB_FILE_BLOCKS_NUM (1024 ∗ 1024 ∗ DB_FILE_SIZE / sizeof(AK_block))
- #define INITIAL_EXTENT_SIZE (iniparser_getint(AK_config,"extents:initial_extent_size",15))

  *Constant declaring initial extent size in blocks.*
- #define EXTENT_GROWTH_TABLE (iniparser_getdouble(AK_config,"extents:extent_growth_table",0.5))

  *Constant declaring extent growth factor for tables.*
- #define EXTENT_GROWTH_INDEX (iniparser_getdouble(AK_config,"extents:extent_growth_index",0.2))

  *Constant declaring extent growth factor for indices.*
- #define EXTENT_GROWTH_TRANSACTION (iniparser_getdouble(AK_config,"extents:extent_growth_↩ transaction",0.2))

  *Constant declaring extent growth factor for transaction segments.*
- #define EXTENT_GROWTH_TEMP (iniparser_getdouble(AK_config,"extents:extent_growth_temp",0.5))

  *Constant declaring extent growth factor for temporary segments.*
- #define ARCHIVELOG_PATH (iniparser_getstring(AK_config, "redolog:archivelog_folder", "./archivelog"))

  *Constant declaring the path of archivelog folder.*
- #define MAX_REDO_LOG_MEMORY 4096

  *The maximum size of REDO log memory.*
- #define MAX_REDO_LOG_ENTRIES 100

  *The maximum size of REDO log entries.*
- #define NUMBER_OF_THREADS (iniparser_getint(AK_config,"general:number_of_threads",42))

  *Constant declaring maximum number of threads that an application can acquire.*

### 7.3.1 Macro Definition Documentation

#### 7.3.1.1 AK_BLOBS_PATH

```
#define AK_BLOBS_PATH (iniparser_getstring(AK_config, "general:blobs_folder", "./blobs"))
```

Constant declaring the path of blobs folder (note: if changed keep in mind for make clean in makefile). Path declared in config.ini has to be absolute (tied up with installation package), but for debugging purpose we are going to keep it relative.

#### 7.3.1.2 ARCHIVELOG_PATH

```
#define ARCHIVELOG_PATH (iniparser_getstring(AK_config, "redolog:archivelog_folder", "./archivelog"))
```

Constant declaring the path of archivelog folder.

#### 7.3.1.3 DB_FILE

```
#define DB_FILE (iniparser_getstring(AK_config,"general:db_file","kalashnikov.db"))
```

#### 7.3.1.4 DB_FILE_BLOCKS_NUM

```
#define DB_FILE_BLOCKS_NUM (1024 * 1024 * DB_FILE_SIZE / sizeof(AK_block))
```

#### 7.3.1.5 DB_FILE_SIZE

```
#define DB_FILE_SIZE (iniparser_getint(AK_config,"general:db_file_size",40))
```

Constant declaring size of DB file in MB.

#### 7.3.1.6 EXTENT_GROWTH_INDEX

```
#define EXTENT_GROWTH_INDEX (iniparser_getdouble(AK_config,"extents:extent_growth_index",0.2))
```

Constant declaring extent growth factor for indices.

#### 7.3.1.7 EXTENT_GROWTH_TABLE

```
#define EXTENT_GROWTH_TABLE (iniparser_getdouble(AK_config,"extents:extent_growth_table",0.5))
```

Constant declaring extent growth factor for tables.

#### 7.3.1.8 EXTENT_GROWTH_TEMP

```
#define EXTENT_GROWTH_TEMP (iniparser_getdouble(AK_config,"extents:extent_growth_temp",0.5))
```

Constant declaring extent growth factor for temporary segments.

#### 7.3.1.9 EXTENT_GROWTH_TRANSACTION

```
#define EXTENT_GROWTH_TRANSACTION (iniparser_getdouble(AK_config,"extents:extent_growth_↩
transaction",0.2))
```

Constant declaring extent growth factor for transaction segments.

#### 7.3.1.10 INITIAL_EXTENT_SIZE

```
#define INITIAL_EXTENT_SIZE (iniparser_getint(AK_config,"extents:initial_extent_size",15))
```

Constant declaring initial extent size in blocks.

#### 7.3.1.11 MAX_EXTENTS_IN_SEGMENT

```
#define MAX_EXTENTS_IN_SEGMENT 200
```

Constant declaring the maximum number of extents in segment.

#### 7.3.1.12 MAX_FREE_SPACE_SIZE

```
#define MAX_FREE_SPACE_SIZE (iniparser_getint(AK_config,"blocks:max_AK_free_space_size",4000))
```

Constant declaring the maximum free space in block.

#### 7.3.1.13 MAX_LAST_TUPLE_DICT_SIZE_TO_USE

```
#define MAX_LAST_TUPLE_DICT_SIZE_TO_USE (iniparser_getint(AK_config,"dictionary:max_last_↩
tuple_dict_size_to_use",470))
```

Constant declaring the maximum size od last tuple in a dictionary.

#### 7.3.1.14 MAX_NUM_OF_BLOCKS

```
#define MAX_NUM_OF_BLOCKS (iniparser_getint(AK_config,"segments:max_num_of_blocks",200))
```

Constant declaring the maximum number of blocks in a segment.

#### 7.3.1.15 MAX_REDO_LOG_ENTRIES

```
#define MAX_REDO_LOG_ENTRIES 100
```

The maximum size of REDO log entries.

#### 7.3.1.16 MAX_REDO_LOG_MEMORY

```
#define MAX_REDO_LOG_MEMORY 4096
```

The maximum size of REDO log memory.

#### 7.3.1.17 NUMBER_OF_THREADS

```
#define NUMBER_OF_THREADS (iniparser_getint(AK_config,"general:number_of_threads",42))
```

Constant declaring maximum number of threads that an application can acquire.

## 7.4 auxi/constants.h File Reference

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

## Macros

- #define MAX_VARCHAR_LENGTH 200

  *Constant declaring the maximum length of varchar data value.*
- #define MAX_ATTRIBUTES 10

  *Constant declaring the maximum number of attributes per block.*
- #define MAX_ATT_NAME 255

  *Constant declaring the maximum length of attribute name string (used in AK_header->att_name)*
- #define MAX_CONSTRAINTS 5

  *Constant declaring the maximum number of constraints per attribute.*
- #define MAX_CONSTR_NAME 255

  *Constant declaring the maximum length of constraint name string (used in AK_header->constr_name)*
- #define MAX_CONSTR_CODE 255

  *Constant declaring the maximum lenght of constraint code string.*
- #define MAX_OBSERVABLE_OBSERVERS 4096

  *Constant for declaring the maximum number of observers objects for some observable type.*
- #define MAX_ACTIVE_TRANSACTIONS_COUNT 100

  *Constant for declaring the maximum number of active trasactions 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 the maximum size of query lib memory.*
- #define MAX_CACHE_MEMORY 255

  *Constant declaring the maximum size of DB cache memory.*
- #define MAX_QUERY_DICT_MEMORY 255

  *Constant declaring the maximum size of query dictionary memory.*
- #define MAX_QUERY_RESULT_MEMORY 255

  *Constant declaring the maximum size of query result cache memory.*
- #define MAX_TOKENS 255

  *Constant declaring the 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 the number of buckets in hash table.*
- #define EXIT_SUCCESS 0

  *Constant declaring a successful exit.*
- #define EXIT_ERROR -1

  *Constant declaring unsuccessful 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 the data is a new value.*
- #define SEARCH_CONSTRAINT 1

  *Constant indicating that the data is constraint to search for.*
- #define UPDATE 0

  *Constant indicating that the operation to be performed is 'update'.*
- #define DELETE 1

  *Constant indicating that the operation to be performed is 'delete'.*
- #define INSERT 2

  *Constant indicating that the operation to be performed is 'insert'.*
- #define SELECT 3

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

  *Constant indicating that the operation to be performed is 'search'.*
- #define INFO_BUCKET 0

  *Constant declaring the type of bucket as "info bucket" when inserting bucket to block.*
- #define MAIN_BUCKET 1

  *Constant declaring the type of bucket as "main bucket" when inserting bucket to block.*
- #define HASH_BUCKET 2

  *Constant declaring the type of bucket as "hash bucket" when inserting bucket to block.*
- #define SHARED_LOCK 0

  *Constant declaring the type of lock as SHARED LOCK.*
- #define EXCLUSIVE_LOCK 1

  *Constant declaring the 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 successfuly.*
- #define NOT_OK 0

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

  *Constant declaring that the transaction is completed successfuly.*
- #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 the 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_CONSTRAINTS_INDEX "AK_constraints_index"

  *Defines system table name for storing check constraints.*
- #define AK_CONSTRAINTS_PRIMARY_KEY "AK_constraints_primary_key"

  *Defines system table name for storing check constraints.*
- #define AK_CONSTRAINTS_FOREIGN_KEY "AK_constraints_foreign_key"

  *Defines system table name for storing check constraints.*
- #define AK_CONSTRAINTS_DEFAULT "AK_constraints_default"

  *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 the number of drop statement.*
- #define DROP_INDEX 1

  *Constant which defines the number of drop statement.*
- #define DROP_VIEW 2

  *Constant which defines the number of drop statement.*
- #define DROP_SEQUENCE 3

  *Constant which defines the number of drop statement.*
- #define DROP_TRIGGER 4

  *Constant which defines the number of drop statement.*
- #define DROP_FUNCTION 5

  *Constant which defines the number of drop statement.*
- #define DROP_USER 6

  *Constant which defines the number of drop statement.*
- #define DROP_GROUP 7

  *Constant which defines the number of drop statement.*
- #define DROP_CONSTRAINT 8

  *Constant which defines thenumber of drop statement.*
- #define NUM_SYS_TABLES 20

  *Constant which defines the length of system_catalog.*

### 7.4.1 Detailed Description

Header file that provides global macros, constants and variables

## 7.4.2 Macro Definition Documentation

### 7.4.2.1 ABORT

```
#define ABORT 0
```

Constant declaring if the transaction is being aborted.

### 7.4.2.2 AK_CONSTRAINTS_BEWTEEN

```
#define AK_CONSTRAINTS_BEWTEEN "AK_constraints_between"
```

Defines system table name for storing between constraints.

### 7.4.2.3 AK_CONSTRAINTS_CHECK_CONSTRAINT

```
#define AK_CONSTRAINTS_CHECK_CONSTRAINT "AK_constraints_check_constraint"
```

Defines system table name for storing check constraints.

### 7.4.2.4 AK_CONSTRAINTS_DEFAULT

```
#define AK_CONSTRAINTS_DEFAULT "AK_constraints_default"
```

Defines system table name for storing check constraints.

- –

### 7.4.2.5 AK_CONSTRAINTS_FOREIGN_KEY

```
#define AK_CONSTRAINTS_FOREIGN_KEY "AK_constraints_foreign_key"
```

Defines system table name for storing check constraints.

- –

#### 7.4.2.6 AK_CONSTRAINTS_INDEX

```
#define AK_CONSTRAINTS_INDEX "AK_constraints_index"
```

Defines system table name for storing check constraints.

- –

#### 7.4.2.7 AK_CONSTRAINTS_NOT_NULL

```
#define AK_CONSTRAINTS_NOT_NULL "AK_constraints_not_null"
```

Defines system table name for storing check constraints.

#### 7.4.2.8 AK_CONSTRAINTS_PRIMARY_KEY

```
#define AK_CONSTRAINTS_PRIMARY_KEY "AK_constraints_primary_key"
```

Defines system table name for storing check constraints.

- –

#### 7.4.2.9 AK_CONSTRAINTS_UNIQUE

```
#define AK_CONSTRAINTS_UNIQUE "AK_constraints_unique"
```

Defines system table name for storing check constraints.

#### 7.4.2.10 AK_REFERENCE

```
#define AK_REFERENCE "AK_reference"
```

Defines system table name for storing check constraints.

#### 7.4.2.11 ATTR_DELIMITER

```
#define ATTR_DELIMITER ";"
```

Constant declaring attributes delimiter.

#### 7.4.2.12 ATTR_ESCAPE

```
#define ATTR_ESCAPE '`'
```

Constant indicating attributes escape section.

#### 7.4.2.13 BLOCK_CLEAN

```
#define BLOCK_CLEAN 0
```

Constant indicating block cleaning (not changed since read from disk)

#### 7.4.2.14 BLOCK_DIRTY

```
#define BLOCK_DIRTY 1
```

Constant indicating dirty block (changed since read from disk, has to be written)

#### 7.4.2.15 BLOCK_TYPE_CHAINED

```
#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)

#### 7.4.2.16 BLOCK_TYPE_FREE

```
#define BLOCK_TYPE_FREE -1
```

Constant declaring AK_free block type (used in AK_block->type)

#### 7.4.2.17 BLOCK_TYPE_NORMAL

```
#define BLOCK_TYPE_NORMAL 0
```

Constant declaring normal block type e.g. used by some extent (used in AK_block->type)

#### 7.4.2.18 COMMIT

```
#define COMMIT 1
```

Constant declaring that the transaction is completed successfuly.

#### 7.4.2.19 DATA_BLOCK_SIZE

```
#define DATA_BLOCK_SIZE 500
```

Constant declaring length of data block size (used in AK_block->data)

#### 7.4.2.20 DATA_ENTRY_SIZE

```
#define DATA_ENTRY_SIZE 10
```

Constant declaring lenght of data entry in sizeof( int )

#### 7.4.2.21 DELETE

```
#define DELETE 1
```

Constant indicating that the operation to be performed is 'delete'.

#### 7.4.2.22 DROP_CONSTRAINT

```
#define DROP_CONSTRAINT 8
```

Constant which defines thenumber of drop statement.

#### 7.4.2.23 DROP_FUNCTION

```
#define DROP_FUNCTION 5
```

Constant which defines the number of drop statement.

#### 7.4.2.24 DROP_GROUP

```
#define DROP_GROUP 7
```

Constant which defines the number of drop statement.

#### 7.4.2.25 DROP_INDEX

```
#define DROP_INDEX 1
```

Constant which defines the number of drop statement.

#### 7.4.2.26 DROP_SEQUENCE

```
#define DROP_SEQUENCE 3
```

Constant which defines the number of drop statement.

#### 7.4.2.27 DROP_TABLE

```
#define DROP_TABLE 0
```

Constant which defines the number of drop statement.

#### 7.4.2.28 DROP_TRIGGER

```
#define DROP_TRIGGER 4
```

Constant which defines the number of drop statement.

#### 7.4.2.29 DROP_USER

```
#define DROP_USER 6
```

Constant which defines the number of drop statement.

#### 7.4.2.30 DROP_VIEW

```
#define DROP_VIEW 2
```

Constant which defines the number of drop statement.

#### 7.4.2.31 EXCLUSIVE_LOCK

```
#define EXCLUSIVE_LOCK 1
```

Constant declaring the type of lock as EXCLUSIVE LOCK.

#### 7.4.2.32 EXIT_ERROR

```
#define EXIT_ERROR -1
```

Constant declaring unsuccessful exit.

#### 7.4.2.33 EXIT_SUCCESS

```
#define EXIT_SUCCESS 0
```

Constant declaring a successful exit.

#### 7.4.2.34 EXIT_WARNING

```
#define EXIT_WARNING -2
```

#### 7.4.2.35 FIND

```
#define FIND 2
```

Constant indicating that the operation to be performed is 'search'.

#### 7.4.2.36 FREE_CHAR

```
#define FREE_CHAR '\0'
```

Constant declaring dummy data for empty chars.

#### 7.4.2.37 FREE_INT

```
#define FREE_INT -10
```

Constant declaring dummy data for empty integers.

#### 7.4.2.38 HASH_BUCKET

```
#define HASH_BUCKET 2
```

Constant declaring the type of bucket as "hash bucket" when inserting bucket to block.

#### 7.4.2.39 HASH_BUCKET_SIZE

```
#define HASH_BUCKET_SIZE 4
```

Constant declaring the size of hash buckets.

#### 7.4.2.40 INFO_BUCKET

```
#define INFO_BUCKET 0
```

Constant declaring the type of bucket as "info bucket" when inserting bucket to block.

#### 7.4.2.41 INSERT

```
#define INSERT 2
```

Constant indicating that the operation to be performed is 'insert'.

#### 7.4.2.42 MAIN_BUCKET

```
#define MAIN_BUCKET 1
```

Constant declaring the type of bucket as "main bucket" when inserting bucket to block.

#### 7.4.2.43 MAIN_BUCKET_SIZE

```
#define MAIN_BUCKET_SIZE 4
```

Constant declaring the size of main buckets.

#### 7.4.2.44 MAX_ACTIVE_TRANSACTIONS_COUNT

```
#define MAX_ACTIVE_TRANSACTIONS_COUNT 100
```

Constant for declaring the maximum number of active trasactions in DBMS.

#### 7.4.2.45 MAX_ATT_NAME

```
#define MAX_ATT_NAME 255
```

Constant declaring the maximum length of attribute name string (used in AK_header->att_name)

#### 7.4.2.46 MAX_ATTRIBUTES

```
#define MAX_ATTRIBUTES 10
```

Constant declaring the maximum number of attributes per block.

#### 7.4.2.47 MAX_BLOCKS_CURRENTLY_ACCESSED

```
#define MAX_BLOCKS_CURRENTLY_ACCESSED 32
```

Indicates the maximum number of threads that can access (read or write) database at the same time.

#### 7.4.2.48 MAX_CACHE_MEMORY

```
#define MAX_CACHE_MEMORY 255
```

Constant declaring the maximum size of DB cache memory.

#### 7.4.2.49 MAX_CONSTR_CODE

```
#define MAX_CONSTR_CODE 255
```

Constant declaring the maximum lenght of constraint code string.

#### 7.4.2.50 MAX_CONSTR_NAME

```
#define MAX_CONSTR_NAME 255
```

Constant declaring the maximum length of constraint name string (used in AK_header->constr_name)

#### 7.4.2.51 MAX_CONSTRAINTS

```
#define MAX_CONSTRAINTS 5
```

Constant declaring the maximum number of constraints per attribute.

#### 7.4.2.52 MAX_MAIN_BUCKETS

```
#define MAX_MAIN_BUCKETS 512
```

Constant declaring the maximum number of main buckets.

#### 7.4.2.53 MAX_OBSERVABLE_OBSERVERS

```
#define MAX_OBSERVABLE_OBSERVERS 4096
```

Constant for declaring the maximum number of observers objects for some observable type.

#### 7.4.2.54 MAX_QUERY_DICT_MEMORY

```
#define MAX_QUERY_DICT_MEMORY 255
```

Constant declaring the maximum size of query dictionary memory.

#### 7.4.2.55 MAX_QUERY_LIB_MEMORY

```
#define MAX_QUERY_LIB_MEMORY 255
```

Constant declaring the maximum size of query lib memory.

#### 7.4.2.56 MAX_QUERY_RESULT_MEMORY

```
#define MAX_QUERY_RESULT_MEMORY 255
```

Constant declaring the maximum size of query result cache memory.

#### 7.4.2.57 MAX_TOKENS

```
#define MAX_TOKENS 255
```

Constant declaring the maximum number of attributes to handle in relation equivalence function.

#### 7.4.2.58 MAX_VARCHAR_LENGTH

```
#define MAX_VARCHAR_LENGTH 200
```

Constant declaring the maximum length of varchar data value.

#### 7.4.2.59 NEW_ID

```
#define NEW_ID 0
```

Constant declaring if new obj_id should be created.

#### 7.4.2.60 NEW_VALUE

```
#define NEW_VALUE 0
```

Constant indicating that the data is a new value.

#### 7.4.2.61 NOT_CHAINED

```
#define NOT_CHAINED -1
```

Constant used in AK_block->chained_with if the block isn't chained.

#### 7.4.2.62 NOT_OK

```
#define NOT_OK 0
```

Constant declaring that the method isn't completed successfuly.

#### 7.4.2.63 NULLL

```
#define NULLL "asdfgXYZ"
```

Constant declaring null value for tables.

#### 7.4.2.64 NUM_SYS_TABLES

```
#define NUM_SYS_TABLES 20
```

Constant which defines the length of system_catalog.

#### 7.4.2.65 NUMBER_OF_KEYS

```
#define NUMBER_OF_KEYS 4096
```

Constant declaring the number of buckets in hash table.

#### 7.4.2.66 OK

```
#define OK 1
```

Constant declaring that the method is completed successfuly.

#### 7.4.2.67 PASS_LOCK_QUEUE

```
#define PASS_LOCK_QUEUE 1
```

Constant declaring that a lock can acquire the resource AK_freely.

#### 7.4.2.68 RO_EXCEPT

```
#define RO_EXCEPT 'e'
```

#### 7.4.2.69 RO_INTERSECT

```
#define RO_INTERSECT 'i'
```

#### 7.4.2.70 RO_NAT_JOIN

```
#define RO_NAT_JOIN 'n'
```

#### 7.4.2.71 RO_PROJECTION

```
#define RO_PROJECTION 'p'
```

#### 7.4.2.72 RO_RENAME

```
#define RO_RENAME 'r'
```

#### 7.4.2.73 RO_SELECTION

```
#define RO_SELECTION 's'
```

#### 7.4.2.74 RO_THETA_JOIN

```
#define RO_THETA_JOIN 't'
```

#### 7.4.2.75 RO_UNION

```
#define RO_UNION 'u'
```

#### 7.4.2.76 SEARCH_CONSTRAINT

```
#define SEARCH_CONSTRAINT 1
```

Constant indicating that the data is constraint to search for.

#### 7.4.2.77 SEGMENT_TYPE_INDEX

```
#define SEGMENT_TYPE_INDEX 2
```

Constant declaring index segment type (used in system catalog)

#### 7.4.2.78 SEGMENT_TYPE_SYSTEM_TABLE

```
#define SEGMENT_TYPE_SYSTEM_TABLE 0
```

Constant declaring system table segment type (used in system catalog)

#### 7.4.2.79 SEGMENT_TYPE_TABLE

```
#define SEGMENT_TYPE_TABLE 1
```

Constant declaring table segment type (used in system catalog)

#### 7.4.2.80 SEGMENT_TYPE_TEMP

```
#define SEGMENT_TYPE_TEMP 4
```

Constant declaring temporary segment type (used in system catalog)

#### 7.4.2.81 SEGMENT_TYPE_TRANSACTION

```
#define SEGMENT_TYPE_TRANSACTION 3
```

Constant declaring transaction segment type (used in system catalog)

#### 7.4.2.82 SELECT

```
#define SELECT 3
```

Constant indicating 'select' operation.

#### 7.4.2.83 SEPARATOR

```
#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.

#### 7.4.2.84 SHARED_LOCK

```
#define SHARED_LOCK 0
```

Constant declaring the type of lock as SHARED LOCK.

#### 7.4.2.85 TEST_MODE_OFF

```
#define TEST_MODE_OFF 0
```

This constant is used to turn testMode (auxi/auxillary.h) OFF.

#### 7.4.2.86 TEST_MODE_ON

```
#define TEST_MODE_ON 1
```

This constant is used to turn testMode (auxi/auxillary.h) ON.

#### 7.4.2.87 TYPE_ATTRIBS

```
#define TYPE_ATTRIBS 12
```

Constant indicating attribute/s in AK_list.

#### 7.4.2.88 TYPE_BLOB

```
#define TYPE_BLOB 8
```

Blob data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.89 TYPE_BOOL

```
#define TYPE_BOOL 9
```

Constant declaring boolean data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.90 TYPE_CONDITION

```
#define TYPE_CONDITION 13
```

Constant indicating condition in AK_list.

#### 7.4.2.91 TYPE_DATE

```
#define TYPE_DATE 5
```

Constant declaring date data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.92 TYPE_DATETIME

```
#define TYPE_DATETIME 6
```

Datetime data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.93 TYPE_FLOAT

```
#define TYPE_FLOAT 2
```

Constant declaring float data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.94 TYPE_INT

```
#define TYPE_INT 1
```

integer data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.95 TYPE_INTERNAL

```
#define TYPE_INTERNAL 0
```

Constant declaring internal data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.96 TYPE_NUMBER

```
#define TYPE_NUMBER 3
```

Constant declaring number data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.97 TYPE_OPERAND

```
#define TYPE_OPERAND 10
```

Constant indicating operand in AK_list.

#### 7.4.2.98 TYPE_OPERATOR

```
#define TYPE_OPERATOR 11
```

indicates operator in AK_list

#### 7.4.2.99 TYPE_TIME

```
#define TYPE_TIME 7
```

Constant declaring time data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.100 TYPE_VARCHAR

```
#define TYPE_VARCHAR 4
```

Constant declaring varchar data type (used in AK_header->type and AK_tuple_dict->type)

#### 7.4.2.101 UPDATE

```
#define UPDATE 0
```

Constant indicating that the operation to be performed is 'update'.

#### 7.4.2.102 WAIT_FOR_UNLOCK

```
#define WAIT_FOR_UNLOCK 0
```

Constant declaring that a lock has to wait until other locks release the resource.

## 7.5 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 that prints the debug message. Provides debug level, debug type and message with corresponding variables for the output.*

### 7.5.1 Detailed Description

Provides a function for debuging

### 7.5.2 Function Documentation

#### 7.5.2.1 AK_dbg_messg()

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

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

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *level* | level of debug information for a given DB module |
| *type* | the name of DB module for which to print debug information |
| *format* | 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

## 7.6 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 a 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 that prints the debug message. Provides debug level, debug type and message with corresponding variables for the output.*

### 7.6.1 Detailed Description

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

### 7.6.2 Macro Definition Documentation

#### 7.6.2.1 DEBUG_ALL

```
#define DEBUG_ALL 0
```

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

**Author**

Dino Laktašić

### 7.6.3 Typedef Documentation

#### 7.6.3.1 DEBUG_LEVEL

```
typedef enum debug_level DEBUG_LEVEL
```

#### 7.6.3.2 DEBUG_TYPE

```
typedef enum debug_type DEBUG_TYPE
```

### 7.6.4 Enumeration Type Documentation

#### 7.6.4.1 debug_level

```
enum debug_level
```

**Enumerator**

| | |
|---|---|
| LOW | |
| MIDDLE | |
| HIGH | |

#### 7.6.4.2 debug_type

```
enum debug_type
```

**Enumerator**

| | |
|---|---|
| GLOBAL | |
| DB_MAN | |
| FILE_MAN | |
| MEMO_MAN | |
| INDICES | |
| TABLES | |
| REL_OP | |
| REL_EQ | |
| CONSTRAINTS | |
| FUNCTIONS | |
| SEQUENCES | |
| TRIGGERS | |
| REDO | |

## 7.6.5 Function Documentation

### 7.6.5.1 AK_dbg_messg()

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

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

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *level* | level of debug information for a given DB module |
| *type* | the name of DB module for which to print debug information |
| *format* | 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

## 7.7 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.*

### 7.7.1 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).

## 7.7.2 Macro Definition Documentation

### 7.7.2.1 DICT_INVALID_KEY

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

Invalid key token

### 7.7.2.2 DICTMINSZ

```
#define DICTMINSZ 128
```

Minimal allocated number of entries in a dictionary

### 7.7.2.3 MAXVALSZ

```
#define MAXVALSZ 1024
```

Maximum value size for integers and doubles.

## 7.7.3 Function Documentation

### 7.7.3.1 dictionary_del()

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

Delete a dictionary object.

**Parameters**

| | |
|---|---|
| *d* | dictionary object to deallocate. |

**Returns**

void

Deallocate a dictionary object and all memory associated to it.

#### 7.7.3.2 dictionary_dump()

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

Dump a dictionary to an opened file pointer.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to dump |
| *f* | 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.

#### 7.7.3.3 dictionary_get()

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

Get a value from a dictionary.

**Parameters**

| | |
|---|---|
| *d* | dictionary object to search. |
| *key* | Key to look for in the dictionary. |
| *def* | 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.

#### 7.7.3.4 dictionary_hash()

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

Compute the hash key for a string.

**Parameters**

| | |
|---|---|
| *key* | 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.

#### 7.7.3.5 dictionary_new()

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

Create a new dictionary object.

**Parameters**

| | |
|---|---|
| *size* | 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.

#### 7.7.3.6 dictionary_set()

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

Set a value in a dictionary.

**Parameters**

| | |
|---|---|
| *d* | dictionary object to modify. |
| *key* | Key to modify or add. |
| *val* | 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.

#### 7.7.3.7 dictionary_unset()

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

Delete a key in a dictionary.

**Parameters**

| | |
|---|---|
| *d* | dictionary object to modify. |
| *key* | Key to remove. |

**Returns**

void

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

## 7.8 auxi/dictionary.h File Reference

Implements a dictionary for string variables.

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

Include dependency graph for dictionary.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct _dictionary_

  *Dictionary object.*

## Typedefs

- typedef struct _dictionary_ dictionary
  *Dictionary object.*

## 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 *vd)
  *Delete a dictionary object.*
- char * dictionary_get (dictionary *d, const char *key, char *def)
  *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.*

### 7.8.1 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).

### 7.8.2 Typedef Documentation

#### 7.8.2.1 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.

### 7.8.3 Function Documentation

#### 7.8.3.1 dictionary_del()

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

Delete a dictionary object.

**Parameters**

| | |
|---|---|
| *d* | dictionary object to deallocate. |

**Returns**

void

Deallocate a dictionary object and all memory associated to it.

#### 7.8.3.2 dictionary_dump()

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

Dump a dictionary to an opened file pointer.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to dump |
| *f* | 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.

#### 7.8.3.3 dictionary_get()

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

Get a value from a dictionary.

**Parameters**

| | |
|---|---|
| *d* | dictionary object to search. |
| *key* | Key to look for in the dictionary. |
| *def* | 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.

#### 7.8.3.4 dictionary_hash()

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

Compute the hash key for a string.

**Parameters**

| | |
|---|---|
| *key* | 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.

#### 7.8.3.5 dictionary_new()

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

Create a new dictionary object.

**Parameters**

| | |
|---|---|
| *size* | 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.

#### 7.8.3.6 dictionary_set()

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

Set a value in a dictionary.

**Parameters**

| | |
|---|---|
| *d* | dictionary object to modify. |
| *key* | Key to modify or add. |
| *val* | 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.

#### 7.8.3.7 dictionary_unset()

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

Delete a key in a dictionary.

**Parameters**

| | |
|---|---|
| *d* | dictionary object to modify. |
| *key* | Key to remove. |

**Returns**

void

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

## 7.9 auxi/iniparser.c File Reference

Parser for ini files.

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

Include dependency graph for iniparser.c:

## Macros

- #define ASCIILINESZ (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

### 7.9.1 Detailed Description

Parser for ini files.

**Author**

N. Devillard

### 7.9.2 Macro Definition Documentation

#### 7.9.2.1 ASCIILINESZ

```
#define ASCIILINESZ (1024)
```

#### 7.9.2.2 INI_INVALID_KEY

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

### 7.9.3 Typedef Documentation

#### 7.9.3.1 line_status

```
typedef enum _line_status_ line_status
```

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

### 7.9.4 Enumeration Type Documentation

#### 7.9.4.1 _line_status_

```
enum _line_status_
```

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

**Enumerator**

| | |
|---|---|
| LINE_UNPROCESSED | |
| LINE_ERROR | |
| LINE_EMPTY | |
| LINE_COMMENT | |
| LINE_SECTION | |
| LINE_VALUE | |

## 7.9.5 Function Documentation

### 7.9.5.1 AK_inflate_config()

```
void AK_inflate_config ( )
```

### 7.9.5.2 iniparser_AK_freedict()

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

Free all memory associated to an ini dictionary.

**Parameters**

| | |
|---|---|
| *d* | 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.

### 7.9.5.3 iniparser_dump()

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

Dump a dictionary to an opened file pointer.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to dump. |
| *f* | 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.

#### 7.9.5.4 iniparser_dump_ini()

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

Save a dictionary to a loadable ini file.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to dump |
| *f* | 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.

#### 7.9.5.5 iniparser_dumpsection_ini()

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

Save a dictionary section to a loadable ini file.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to dump |
| *s* | Section name of dictionary to dump |
| *f* | 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.

#### 7.9.5.6 iniparser_find_entry()

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

Finds out if a given entry exists in a dictionary.

**Parameters**

| | |
|---|---|
| *ini* | Dictionary to search |
| *entry* | 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.

#### 7.9.5.7 iniparser_getboolean()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to search |
| *key* | Key string to look for |
| *notfound* | 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.

#### 7.9.5.8 iniparser_getdouble()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to search |
| *key* | Key string to look for |
| *notfound* | 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.

#### 7.9.5.9 iniparser_getint()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to search |
| *key* | Key string to look for |
| *notfound* | 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()

#### 7.9.5.10 iniparser_getnsec()

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

Get number of sections in a dictionary.

**Parameters**

| | |
|---|---|
| *d* | 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.

#### 7.9.5.11 iniparser_getseckeys()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to examine |
| *s* | 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.

#### 7.9.5.12 iniparser_getsecname()

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

Get name for section n in a dictionary.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to examine |
| *n* | 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.

#### 7.9.5.13 iniparser_getsecnkeys()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to examine |
| *s* | Section name of dictionary to examine |

**Returns**

Number of keys in section

#### 7.9.5.14 iniparser_getstring()

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

Get the string associated to a key.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to search |
| *key* | Key string to look for |
| *def* | 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.

#### 7.9.5.15 iniparser_load()

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

Parse an ini file and return an allocated dictionary object.

**Parameters**

| | |
|---|---|
| *ininame* | 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().

#### 7.9.5.16 iniparser_set()

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

Set an entry in a dictionary.

**Parameters**

| | |
|---|---|
| *ini* | Dictionary to modify. |
| *entry* | Entry to modify (entry name) |
| *val* | 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.

#### 7.9.5.17 iniparser_unset()

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

Delete an entry in a dictionary.

**Parameters**

| | |
|---|---|
| *ini* | Dictionary to modify |
| *entry* | Entry to delete (entry name) |

**Returns**

void

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

### 7.9.6 Variable Documentation

#### 7.9.6.1 AK_config

```
dictionary* AK_config
```

#### 7.9.6.2 iniParserMutex

```
pthread_mutex_t iniParserMutex = PTHREAD_MUTEX_INITIALIZER
```

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

### 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_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.*
- void iniparser_dump (dictionary *d, FILE *f)
  *Dump a dictionary to an opened file pointer.*
- 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_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.*
- int iniparser_find_entry (dictionary *ini, const char *entry)
  *Finds out if a given entry exists 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

- dictionary * AK_config

### 7.10.1 Detailed Description

Parser for ini files.

**Author**

N. Devillard

### 7.10.2 Function Documentation

#### 7.10.2.1 AK_inflate_config()

```
void AK_inflate_config ( )
```

#### 7.10.2.2 iniparser_AK_freedict()

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

Free all memory associated to an ini dictionary.

**Parameters**

| | |
|---|---|
| *d* | 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.

#### 7.10.2.3 iniparser_dump()

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

Dump a dictionary to an opened file pointer.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to dump. |
| *f* | 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.

#### 7.10.2.4 iniparser_dump_ini()

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

Save a dictionary to a loadable ini file.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to dump |
| *f* | 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.

#### 7.10.2.5 iniparser_dumpsection_ini()

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

Save a dictionary section to a loadable ini file.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to dump |
| *s* | Section name of dictionary to dump |
| *f* | 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.

#### 7.10.2.6 iniparser_find_entry()

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

Finds out if a given entry exists in a dictionary.

**Parameters**

| | |
|---|---|
| *ini* | Dictionary to search |
| *entry* | 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.

#### 7.10.2.7 iniparser_getboolean()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to search |
| *key* | Key string to look for |
| *notfound* | 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.

#### 7.10.2.8 iniparser_getdouble()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to search |
| *key* | Key string to look for |
| *notfound* | 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.

#### 7.10.2.9 iniparser_getint()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to search |
| *key* | Key string to look for |
| *notfound* | 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**

| | |
|---|---|
| *d* | Dictionary to search |
| *key* | Key string to look for |
| *notfound* | 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()

#### 7.10.2.10 iniparser_getnsec()

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

Get number of sections in a dictionary.

**Parameters**

| | |
|---|---|
| *d* | 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.

#### 7.10.2.11 iniparser_getseckeys()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to examine |
| *s* | 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.

#### 7.10.2.12 iniparser_getsecname()

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

Get name for section n in a dictionary.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to examine |
| *n* | 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.

#### 7.10.2.13 iniparser_getsecnkeys()

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

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

**Parameters**

| | |
|---|---|
| *d* | Dictionary to examine |
| *s* | Section name of dictionary to examine |

**Returns**

Number of keys in section

#### 7.10.2.14 iniparser_getstring()

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

Get the string associated to a key.

**Parameters**

| | |
|---|---|
| *d* | Dictionary to search |
| *key* | Key string to look for |
| *def* | 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.

#### 7.10.2.15 iniparser_load()

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

Parse an ini file and return an allocated dictionary object.

**Parameters**

| | |
|---|---|
| *ininame* | 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().

#### 7.10.2.16 iniparser_set()

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

Set an entry in a dictionary.

**Parameters**

| | |
|---|---|
| *ini* | Dictionary to modify. |
| *entry* | Entry to modify (entry name) |
| *val* | 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.

#### 7.10.2.17 iniparser_unset()

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

Delete an entry in a dictionary.

**Parameters**

| | |
|---|---|
| *ini* | Dictionary to modify |
| *entry* | Entry to delete (entry name) |

**Returns**

void

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

### 7.10.3 Variable Documentation

#### 7.10.3.1 AK_config

```
dictionary* AK_config
```

## 7.11 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.*

### 7.11.1 Detailed Description

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

## 7.11.2 Function Documentation

### 7.11.2.1 AK_calloc()

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

Allocates memory (see calloc) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *num* | number of elements |
| *size* | of element in bytes |

**Returns**

allocated memory or NULL

### 7.11.2.2 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

### 7.11.2.3 AK_debmod_calloc()

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

Allocates memory [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *ds* | debug mode state |
| *size* | in bytes to allocate |

**Returns**

pointer to allocated memory or NULL

#### 7.11.2.4 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**

| | |
|---|---|
| *ds* | debug mode state |
| *message* | string to print |

**Returns**

void

#### 7.11.2.5 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**

| | |
|---|---|
| *ds* | debug mode state |

**Returns**

void

#### 7.11.2.6 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**

| | |
|---|---|
| *ds* | debug mode state |
| *format* | format string like printf |

**Returns**

void

#### 7.11.2.7 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**

| | |
|---|---|
| *ds* | debug mode state |

**Returns**

void

#### 7.11.2.8 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**

| | |
|---|---|
| *ds* | debug mode state |
| *memory* | |

**Returns**

void

#### 7.11.2.9 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**

| | |
|---|---|
| *ds* | debug mode state |

**Returns**

function id popped

#### 7.11.2.10 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**

| | |
|---|---|
| *ds* | debug mode state |
| *func_id* | function id |

**Returns**

void

#### 7.11.2.11 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**

| | |
|---|---|
| *ds* | debug mode state |
| *func_name* | |

**Returns**

id for added function name

#### 7.11.2.12 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**

| | |
|---|---|
| *ds* | debug mode state |
| *function_id* | |

**Returns**

function name for given function_id

### 7.11.2.13 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**

| | |
|---|---|
| *ds* | debug mode state |
| *func_name* | function name [private function] |

**Returns**

function id

### 7.11.2.14 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**

| | |
|---|---|
| *ds* | debug mode state |
| *new_function↩ _id* | |

**Returns**

void

#### 7.11.2.15 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**

| | |
|---|---|
| *func_name* | function name as in source |
| *source_file* | file name where function is defined |
| *source_line* | line from which this function is called |

**Returns**

void

#### 7.11.2.16 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**

| | |
|---|---|
| *func_name* | function name as in source |
| *source_file* | file name where function is defined |
| *source_line* | line from which this function is called |

**Returns**

void

#### 7.11.2.17 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

#### 7.11.2.18 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**

| | |
|---|---|
| *ds* | debug mode state |

**Returns**

void

#### 7.11.2.19 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**

| | |
|---|---|
| *func_id* | calling function id |

**Returns**

void

#### 7.11.2.20 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**

| | |
|---|---|
| *func_name* | function name |
| *in_recur* | called in recursion |

**Returns**

void

#### 7.11.2.21 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

#### 7.11.2.22 AK_free()

```
void AK_free (
            void * ptr )
```

Free memory at ptr (see free) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *ptr* | pointer to memory |

**Returns**

void

#### 7.11.2.23 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

#### 7.11.2.24 AK_malloc()

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

Allocate memory (see malloc) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *size* | of memory to allocate in bytes |

**Returns**

allocated memory or NULL

#### 7.11.2.25 AK_mempro_test()

```
void AK_mempro_test ( )
```

Test function.

**Author**

Ivan Kristo

#### 7.11.2.26 AK_print_active_functions()

```
void AK_print_active_functions ( )
```

Print all detected functions.

**Author**

Marin Rukavina, Mislav Bozicevic

**Returns**

void

#### 7.11.2.27 AK_print_function_use()

```
void AK_print_function_use (
            const char * func_name )
```

Print function dependency [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *func_name* | function name |

**Returns**

void

#### 7.11.2.28 AK_print_function_uses()

```
void AK_print_function_uses ( )
```

Print function dependency for all functions [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Returns**

void

#### 7.11.2.29 AK_realloc()

```
void* AK_realloc (
            void * ptr,
            size_t size )
```

Reallocates memory (see realloc) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *ptr* | old memory |
| *size* | new size |

**Returns**

reallocated memory or NULL

#### 7.11.2.30 AK_write_protect()

```
void AK_write_protect (
            void * memory )
```

Function write-protects memory [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *memory* | |

**Returns**

void

#### 7.11.2.31 AK_write_unprotect()

```
void AK_write_unprotect (
            void * memory )
```

Function write-unprotects memory [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *memory* | |

**Returns**

void

## 7.12 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 the debug mode messages are going to be printed.*
- #define AK_DEBMOD_PAGES_NUM 8192

  *Defines the total 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 the 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

### 7.12.1 Detailed Description

Data structures, includes, macros and declarations for the memory wrappers and debug mode of Kalashnikov DB.

### 7.12.2 Macro Definition Documentation

#### 7.12.2.1 AK_DEBMOD_MAX_FUNC_NAME

```
#define AK_DEBMOD_MAX_FUNC_NAME 80
```

Defines the maximum function name length possible.

#### 7.12.2.2 AK_DEBMOD_MAX_FUNCTIONS

```
#define AK_DEBMOD_MAX_FUNCTIONS 500
```

Defines the maximum number of function names in the application.

#### 7.12.2.3 AK_DEBMOD_MAX_WRITE_DETECTIONS

```
#define AK_DEBMOD_MAX_WRITE_DETECTIONS (AK_DEBMOD_PAGES_NUM * 10)
```

Defines the maximum number of memory write detections.

#### 7.12.2.4 AK_DEBMOD_ON

```
#define AK_DEBMOD_ON 0
```

Zero to switch memory protection and debug mode off.

#### 7.12.2.5 AK_DEBMOD_PAGES_NUM

```
#define AK_DEBMOD_PAGES_NUM 8192
```

Defines the total available memory pages for allocation.

#### 7.12.2.6 AK_DEBMOD_PRINT

```
#define AK_DEBMOD_PRINT 0
```

Defines if the debug mode messages are going to be printed.

#### 7.12.2.7 AK_DEBMOD_STACKSIZE

```
#define AK_DEBMOD_STACKSIZE AK_DEBMOD_PAGES_NUM
```

Defines the monitored functions stack.

#### 7.12.2.8 AK_EPI

```
#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.

#### 7.12.2.9 AK_INLINE

```
#define AK_INLINE __inline__
```

#### 7.12.2.10 AK_PRO

```
#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.

#### 7.12.2.11 NEW

```
#define NEW(
              type,
              type_size ) (calloc(type_size, sizeof(type)))
```

### 7.12.3 Function Documentation

#### 7.12.3.1 AK_calloc()

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

Allocates memory (see calloc) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *num* | number of elements |
| *size* | of element in bytes |

**Returns**

allocated memory or NULL

#### 7.12.3.2 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

#### 7.12.3.3 AK_debmod_calloc()

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

Allocates memory [private function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *ds* | debug mode state |
| *size* | in bytes to allocate |

**Returns**

pointer to allocated memory or NULL

#### 7.12.3.4 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**

| | |
|---|---|
| *ds* | debug mode state |
| *message* | string to print |

**Returns**

void

#### 7.12.3.5 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**

| | |
|---|---|
| *ds* | debug mode state |

**Returns**

void

#### 7.12.3.6 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**

| | |
|---|---|
| *ds* | debug mode state |
| *format* | format string like printf |

**Returns**

void

#### 7.12.3.7 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**

| | |
|---|---|
| *ds* | debug mode state |

**Returns**

void

#### 7.12.3.8 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**

| | |
|---|---|
| *ds* | debug mode state |
| *memory* | |

**Returns**

void

#### 7.12.3.9 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**

| | |
|---|---|
| *ds* | debug mode state |

**Returns**

function id popped

#### 7.12.3.10 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**

| | |
|---|---|
| *ds* | debug mode state |
| *func_id* | function id |

**Returns**

void

#### 7.12.3.11 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**

| | |
|---|---|
| *ds* | debug mode state |
| *func_name* | |

**Returns**

id for added function name

#### 7.12.3.12 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**

| | |
|---|---|
| *ds* | debug mode state |
| *function_id* | |

**Returns**

function name for given function_id

#### 7.12.3.13 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**

| | |
|---|---|
| *ds* | debug mode state |
| *func_name* | function name [private function] |

**Returns**

function id

#### 7.12.3.14 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**

| | |
|---|---|
| *ds* | debug mode state |
| *new_function↩ _id* | |

**Returns**

void

#### 7.12.3.15 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**

| | |
|---|---|
| *func_name* | function name as in source |
| *source_file* | file name where function is defined |
| *source_line* | line from which this function is called |

**Returns**

void

#### 7.12.3.16 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**

| | |
|---|---|
| *func_name* | function name as in source |
| *source_file* | file name where function is defined |
| *source_line* | line from which this function is called |

**Returns**

void

#### 7.12.3.17 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

#### 7.12.3.18 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**

| | |
|---|---|
| *ds* | debug mode state |

**Returns**

void

#### 7.12.3.19 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**

| | |
|---|---|
| *func↩ _id* | calling function id |

**Returns**

void

#### 7.12.3.20 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**

| | |
|---|---|
| *func_name* | function name |
| *in_recur* | called in recursion |

**Returns**

void

#### 7.12.3.21 AK_free()

```
void AK_free (
            void * ptr )
```

Free memory at ptr (see free) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *ptr* | pointer to memory |

**Returns**

void

#### 7.12.3.22 AK_malloc()

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

Allocate memory (see malloc) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *size* | of memory to allocate in bytes |

**Returns**

allocated memory or NULL

#### 7.12.3.23 AK_mempro_test()

```
void AK_mempro_test ( )
```

Test function.

**Author**

Ivan Kristo

#### 7.12.3.24 AK_print_active_functions()

```
void AK_print_active_functions ( )
```

Print all detected functions.

**Author**

Marin Rukavina, Mislav Bozicevic

**Returns**

void

#### 7.12.3.25 AK_print_function_use()

```
void AK_print_function_use (
            const char * func_name )
```

Print function dependency [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *func_name* | function name |

**Returns**

void

#### 7.12.3.26 AK_print_function_uses()

```
void AK_print_function_uses ( )
```

Print function dependency for all functions [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Returns**

void

#### 7.12.3.27 AK_realloc()

```
void* AK_realloc (
            void * ptr,
            size_t size )
```

Reallocates memory (see realloc) [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *ptr* | old memory |
| *size* | new size |

**Returns**

reallocated memory or NULL

#### 7.12.3.28 AK_write_protect()

```
void AK_write_protect (
            void * memory )
```

Function write-protects memory [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *memory* | |

**Returns**

void

#### 7.12.3.29 AK_write_unprotect()

```
void AK_write_unprotect (
            void * memory )
```

Function write-unprotects memory [public function].

**Author**

Marin Rukavina, Mislav Bozicevic

**Parameters**

| | |
|---|---|
| *memory* | |

**Returns**

void

### 7.12.4 Variable Documentation

#### 7.12.4.1 AK_DEBMOD_STATE

```
AK_debmod_state* AK_DEBMOD_STATE
```

## 7.13 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_↩ ObservableType_Def, void *AK_custom_action)

  *Function that initializes a observable object.*
- AK_observer * AK_init_observer (void *observer_type, void(*observer_type_event_handler)(void *, void *, AK_ObservableType_Enum))

  *Function that initializes the 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 ()
- TestResult AK_observable_test ()

  *Function that runs tests for observable pattern.*
- TestResult AK_observable_pattern ()

### 7.13.1 Detailed Description

File that provides the implementations of functions for observable pattern

## 7.13.2 Typedef Documentation

### 7.13.2.1 AK_TypeObservable

```
typedef struct TypeObservable AK_TypeObservable
```

### 7.13.2.2 AK_TypeObserver

```
typedef struct TypeObserver AK_TypeObserver
```

### 7.13.2.3 AK_TypeObserver_Second

```
typedef struct TypeObserver AK_TypeObserver_Second
```

### 7.13.2.4 NotifyDetails

```
typedef struct _notifyDetails NotifyDetails
```

## 7.13.3 Enumeration Type Documentation

### 7.13.3.1 NotifyType

```
enum NotifyType
```

**Enumerator**

| | |
|---|---|
| ERROR | |
| INFO | |
| WARMING | |

## 7.13.4 Function Documentation

#### 7.13.4.1 AK_custom_action()

```
int AK_custom_action (
             void * data )
```

#### 7.13.4.2 AK_custom_register_observer()

```
int AK_custom_register_observer (
             AK_TypeObservable * self,
             AK_observer * observer )
```

#### 7.13.4.3 AK_custom_unregister_observer()

```
int AK_custom_unregister_observer (
             AK_TypeObservable * self,
             AK_observer * observer )
```

#### 7.13.4.4 AK_get_message()

```
char* AK_get_message (
             AK_TypeObservable * self )
```

#### 7.13.4.5 AK_init_observable()

```
AK_observable* AK_init_observable (
             void * AK_observable_type,
             AK_ObservableType_Enum AK_ObservableType_Def,
             void * AK_custom_action )
```

Function that initializes a observable object.

**Author**

Ivan Pusic

**Returns**

Pointer to new observable object

#### 7.13.4.6 AK_init_observer()

```
AK_observer* AK_init_observer (
            void * observer_type,
            void(*)(void *, void *, AK_ObservableType_Enum) observer_type_event_handler )
```

Function that initializes the observer object.

**Author**

Ivan Pusic

**Returns**

Pointer to new observer object

#### 7.13.4.7 AK_observable_pattern()

```
TestResult AK_observable_pattern ( )
```

#### 7.13.4.8 AK_observable_test()

```
TestResult AK_observable_test ( )
```

Function that runs tests for observable pattern.

**Author**

Ivan Pusic

#### 7.13.4.9 AK_set_notify_info_details()

```
void AK_set_notify_info_details (
            AK_TypeObservable * self,
            NotifyType type,
            char * message )
```

#### 7.13.4.10 custom_observer_event_handler()

```
void custom_observer_event_handler (
            void * observer,
            void * observable,
            AK_ObservableType_Enum AK_ObservableType_Def )
```

#### 7.13.4.11 handle_AK_custom_type()

```
void handle_AK_custom_type (
            AK_TypeObserver * observer,
            AK_TypeObservable * observable )
```

#### 7.13.4.12 init_observable_type()

```
AK_TypeObservable* init_observable_type ( )
```

#### 7.13.4.13 init_observer_type()

```
AK_TypeObserver* init_observer_type (
            void * observable )
```

#### 7.13.4.14 init_observer_type_second()

```
AK_TypeObserver* init_observer_type_second ( )
```

## 7.14 auxi/observable.h File Reference

```
#include "test.h"
#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 that defines the functions for observer object.*
- struct Observable

  *Structure that defines the 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 that initializes the observer object.*
- AK_observable * AK_init_observable (void *AK_observable_type, AK_ObservableType_Enum AK_ObservableType_Def, void *AK_custom_action)

  *Function that initializes a observable object.*
- TestResult AK_observable_test ()

  *Function that runs tests for observable pattern.*
- TestResult AK_observable_pattern ()

### 7.14.1 Detailed Description

Header file that provides data structures and declarations of functions for observable pattern

### 7.14.2 Typedef Documentation

#### 7.14.2.1 AK_observable

```
typedef struct Observable AK_observable
```

#### 7.14.2.2 AK_observer

```
typedef struct Observer AK_observer
```

### 7.14.3 Enumeration Type Documentation

#### 7.14.3.1 AK_ObservableType_Enum

```
enum AK_ObservableType_Enum
```

**Enumerator**

| | |
|---|---|
| AK_TRANSACTION | |
| AK_TRIGGER | |
| AK_CUSTOM_FIRST | |
| AK_CUSTOM_SECOND | |

## 7.14.4 Function Documentation

### 7.14.4.1 AK_init_observable()

```
AK_observable* AK_init_observable (
            void * AK_observable_type,
            AK_ObservableType_Enum AK_ObservableType_Def,
            void * AK_custom_action )
```

Function that initializes a observable object.

**Author**

Ivan Pusic

**Returns**

Pointer to new observable object

### 7.14.4.2 AK_init_observer()

```
AK_observer* AK_init_observer (
            void * observer_type,
            void(*)(void *, void *, AK_ObservableType_Enum) observer_type_event_handler )
```

Function that initializes the observer object.

**Author**

Ivan Pusic

**Returns**

Pointer to new observer object

#### 7.14.4.3 AK_observable_pattern()

```
TestResult AK_observable_pattern ( )
```

#### 7.14.4.4 AK_observable_test()

```
TestResult AK_observable_test ( )
```

Function that runs tests for observable pattern.

**Author**

Ivan Pusic

## 7.15 auxi/test.c File Reference

```
#include "test.h"
```

Include dependency graph for test.c:

### Functions

- TestResult TEST_result (int successfulAmount, int failedAmount)

  *Returns the amount of successful and failed tests.*
- void TEST_output_results (TestResult result)

  *Prints a beautiful string informing the user of test results in the terminal.*

### 7.15.1 Detailed Description

Provides functions for reporting test results for modules.

### 7.15.2 Function Documentation

#### 7.15.2.1 TEST_output_results()

```
void TEST_output_results (
            TestResult result )
```

Prints a beautiful string informing the user of test results in the terminal.

**Author**

Igor Rinkovec

**Returns**

void

#### 7.15.2.2 TEST_result()

```
TestResult TEST_result (
            int successfulAmount,
            int failedAmount )
```

Returns the amount of successful and failed tests.

**Author**

Igor Rinkovec

**Parameters**

| | |
|---|---|
| *successfulAmount* | amount of successful tests |
| *failedAmount* | amount of failed tests |

**Returns**

TestResult

## 7.16 file/test.c File Reference

```
#include <pthread.h>
#include <stdio.h>
#include "test.h"
#include "../trans/transaction.h"
#include "../file/table.h"
#include "../auxi/auxiliary.h"
#include "../opti/rel_eq_comut.h"
```

Include dependency graph for test.c:

### Functions

- char * AK_get_table_atribute_types (char *tblName)

  *returns a string containing attribute types for the 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 the 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)

  *Function that prints the requested column.*
- int get_row_test (int num, char *tbl)

  *Function that prints the requested row.*
- void AK_create_test_tables ()

  *Function that calls all functions for creating test tables in this file.*
- void AK_create_test_table_student ()

*Creates table "student" and fills it with arbitrary data, for testing purposes.*

- void AK_create_test_table_professor ()

  *Creates table "professor" and fills it with arbitrary data, for testing purposes.*
- void AK_create_test_table_professor2 ()

  *Creates table "professor2" and fills it with arbitrary data, for testing purposes.*
- void AK_create_test_table_assistant ()

  *Creates table "assistant" and fills it with arbitrary data, for testing purposes.*
- void AK_create_test_table_employee ()

  *Creates table "employee" and fills it with arbitrary data, for testing purposes.*
- void AK_create_test_table_department ()

  *Creates table "department" and fills it with arbitrary data, for testing purposes.*
- void AK_create_test_table_course ()

  *Creates table "Course" and fills it with arbitrary data, for testing purposes.*

## 7.16.1 Detailed Description

Provides functions for testing purposes

## 7.16.2 Function Documentation

### 7.16.2.1 AK_create_test_table_assistant()

```
void AK_create_test_table_assistant ( )
```

Creates table "assistant" and fills it with arbitrary data, for testing purposes.

**Author**

Žan Žlender

**Returns**

No return value

### 7.16.2.2 AK_create_test_table_course()

```
void AK_create_test_table_course ( )
```

Creates table "Course" and fills it with arbitrary data, for testing purposes.

**Author**

Žan Žlender

**Returns**

No return value

#### 7.16.2.3 AK_create_test_table_department()

```
void AK_create_test_table_department ( )
```

Creates table "department" and fills it with arbitrary data, for testing purposes.

**Author**

Žan Žlender

**Returns**

No return value

#### 7.16.2.4 AK_create_test_table_employee()

```
void AK_create_test_table_employee ( )
```

Creates table "employee" and fills it with arbitrary data, for testing purposes.

**Author**

Žan Žlender

**Returns**

No return value

#### 7.16.2.5 AK_create_test_table_professor()

```
void AK_create_test_table_professor ( )
```

Creates table "professor" and fills it with arbitrary data, for testing purposes.

**Author**

Žan Žlender

**Returns**

No return value

#### 7.16.2.6 AK_create_test_table_professor2()

```
void AK_create_test_table_professor2 ( )
```

Creates table "professor2" and fills it with arbitrary data, for testing purposes.

**Author**

Žan Žlender

**Returns**

No return value

#### 7.16.2.7 AK_create_test_table_student()

```
void AK_create_test_table_student ( )
```

Creates table "student" and fills it with arbitrary data, for testing purposes.

**Author**

Žan Žlender

**Returns**

No return value

#### 7.16.2.8 AK_create_test_tables()

```
void AK_create_test_tables ( )
```

Function that calls all functions for creating test tables in this file.

Function for creating test tables.

**Author**

Dino Laktašić edited by Žan Žlender @2022

**Returns**

No return value

#### 7.16.2.9 AK_get_table_atribute_types()

```
char* AK_get_table_atribute_types (
            char * tblName )
```

returns a string containing attribute types for the supplied table name, seperated by ATTR_DELIMITER

**Author**

Goran Štrok

**Parameters**

| | |
|---|---|
| *tblName* | name of the table for which the attribute types will be returned |

#### 7.16.2.10 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**

| | |
|---|---|
| *tbl_name* | - name of the table for which the header will be created |
| *attr_name* | - array of attribute names |
| *_num* | - number of attributes |
| *_type* | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

**Returns**

1 if ok, 0 otherwise

#### 7.16.2.11 get_column_test()

```
int get_column_test (
            int num,
            char * tbl )
```

Function that prints the requested column.

**Author**

Luka Rajcevic

**Returns**

1 if column is found, 0 otherwise

**Parameters**

| | |
|---|---|
| *num* | - 0 based index of column |
| *tbl* | - name of the table |

#### 7.16.2.12 get_row_test()

```
int get_row_test (
             int num,
             char * tbl )
```

Function that prints the requested row.

**Author**

Luka Rajcevic

**Returns**

1 if row is found, 0 otherwise

**Parameters**

| | |
|---|---|
| *num* | - 0 based index of row |
| *tbl* | - name of the table |

#### 7.16.2.13 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 the table (needed for python testing)

**Author**

Luka Rajcevic

**Parameters**

| | |
|---|---|
| *tbl_name* | - name of the table for which the header will be created |
| *attr_name* | - array of attribute names |
| *attr_value* | - values of attributes to be inserted |
| *_num* | - number of attributes |
| *_type* | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

**Returns**

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

#### 7.16.2.14 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**

| | |
|---|---|
| *src_table* | - name of the source table<br>• |
| *dest_table* | - table in which selection will be stored |
| *sel_query* | - array of operators, operands and attributes (postfix query) |
| *_num* | - number of attributes |
| *_type* | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

**Returns**

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

## 7.17 auxi/test.h File Reference

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

Include dependency graph for test.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct TestResult

  *Used so tests can report the amount of successful tests.*

## Macros

- #define RESET "\033[0m"
- #define BLACK "\033[30m" /* Black */
- #define RED "\033[31m" /* Red */
- #define GREEN "\033[32m" /* Green */
- #define YELLOW "\033[33m" /* Yellow */
- #define BLUE "\033[34m" /* Blue */
- #define MAGENTA "\033[35m" /* Magenta */
- #define CYAN "\033[36m" /* Cyan */
- #define WHITE "\033[37m" /* White */
- #define BOLDBLACK "\033[1m\033[30m" /* Bold Black */
- #define BOLDRED "\033[1m\033[31m" /* Bold Red */
- #define BOLDGREEN "\033[1m\033[32m" /* Bold Green */
- #define BOLDYELLOW "\033[1m\033[33m" /* Bold Yellow */
- #define BOLDBLUE "\033[1m\033[34m" /* Bold Blue */
- #define BOLDMAGENTA "\033[1m\033[35m" /* Bold Magenta */
- #define BOLDCYAN "\033[1m\033[36m" /* Bold Cyan */
- #define BOLDWHITE "\033[1m\033[37m" /* Bold White */

## Typedefs

- typedef struct TestResult TestResult

## Functions

- TestResult TEST_result (int successfulAmount, int failedAmount)

  *Returns the amount of successful and failed tests.*
- void TEST_output_results (TestResult result)

  *Prints a beautiful string informing the user of test results in the terminal.*

### 7.17.1 Macro Definition Documentation

#### 7.17.1.1 BLACK

```
#define BLACK "\033[30m" /* Black */
```

#### 7.17.1.2 BLUE

```
#define BLUE "\033[34m" /* Blue */
```

#### 7.17.1.3 BOLDBLACK

```
#define BOLDBLACK "\033[1m\033[30m" /* Bold Black */
```

#### 7.17.1.4 BOLDBLUE

```
#define BOLDBLUE "\033[1m\033[34m" /* Bold Blue */
```

#### 7.17.1.5 BOLDCYAN

```
#define BOLDCYAN "\033[1m\033[36m" /* Bold Cyan */
```

#### 7.17.1.6 BOLDGREEN

```
#define BOLDGREEN "\033[1m\033[32m" /* Bold Green */
```

#### 7.17.1.7 BOLDMAGENTA

```
#define BOLDMAGENTA "\033[1m\033[35m" /* Bold Magenta */
```

#### 7.17.1.8 BOLDRED

```
#define BOLDRED "\033[1m\033[31m" /* Bold Red */
```

#### 7.17.1.9 BOLDWHITE

```
#define BOLDWHITE "\033[1m\033[37m" /* Bold White */
```

#### 7.17.1.10 BOLDYELLOW

```
#define BOLDYELLOW "\033[1m\033[33m" /* Bold Yellow */
```

#### 7.17.1.11 CYAN

```
#define CYAN "\033[36m" /* Cyan */
```

#### 7.17.1.12 GREEN

```
#define GREEN "\033[32m" /* Green */
```

#### 7.17.1.13 MAGENTA

```
#define MAGENTA "\033[35m" /* Magenta */
```

#### 7.17.1.14 RED

```
#define RED "\033[31m" /* Red */
```

#### 7.17.1.15 RESET

```
#define RESET "\033[0m"
```

Provides services for reporting test results.

#### 7.17.1.16 WHITE

```
#define WHITE "\033[37m" /* White */
```

#### 7.17.1.17 YELLOW

```
#define YELLOW "\033[33m" /* Yellow */
```

### 7.17.2 Typedef Documentation

#### 7.17.2.1 TestResult

```
typedef struct TestResult TestResult
```

### 7.17.3 Function Documentation

#### 7.17.3.1 TEST_output_results()

```
void TEST_output_results (
            TestResult result )
```

Prints a beautiful string informing the user of test results in the terminal.

**Author**

Igor Rinkovec

**Returns**

void

#### 7.17.3.2 TEST_result()

```
TestResult TEST_result (
            int successfulAmount,
            int failedAmount )
```

Returns the amount of successful and failed tests.

**Author**

Igor Rinkovec

**Parameters**

| | |
|---|---|
| *successfulAmount* | amount of successful tests |
| *failedAmount* | amount of failed tests |

**Returns**

TestResult

## 7.18 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_atribute_types (char *tblName)
  *returns a string containing attribute types for the 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 the 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)
  *Function that prints the requested column.*
- int get_row_test (int num, char *tbl)
  *Function that prints the requested row.*
- void AK_create_test_tables ()
  *Function for creating test tables.*

### 7.18.1 Detailed Description

Header file that provides functions and defines for testing purposes

### 7.18.2 Function Documentation

#### 7.18.2.1 AK_create_test_tables()

```
void AK_create_test_tables ( )
```

Function for creating test tables.

**Author**

Dino Laktašić

**Returns**

No return value

Function for creating test tables.

**Author**

Dino Laktašić edited by Žan Žlender @2022

**Returns**

No return value

#### 7.18.2.2 AK_get_table_atribute_types()

```
char* AK_get_table_atribute_types (
            char * tblName )
```

returns a string containing attribute types for the supplied table name, seperated by ATTR_DELIMITER

**Author**

Goran Štrok

**Parameters**

| | |
|---|---|
| *tblName* | name of the table for which the attribute types will be returned |

#### 7.18.2.3 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**

| | |
|---|---|
| *tbl_name* | - name of the table for which the header will be created |
| *attr_name* | - array of attribute names |
| *_num* | - number of attributes |
| *_type* | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

**Returns**

1 if ok, 0 otherwise

#### 7.18.2.4 get_column_test()

```
int get_column_test (
            int num,
            char * tbl )
```

Function that prints the requested column.

**Author**

Luka Rajcevic

**Returns**

1 if column is found, 0 otherwise

**Parameters**

| | |
|---|---|
| *num* | - 0 based index of column |
| *tbl* | - name of the table |

#### 7.18.2.5 get_row_test()

```
int get_row_test (
            int num,
            char * tbl )
```

Function that prints the requested row.

**Author**

Luka Rajcevic

**Returns**

1 if row is found, 0 otherwise

**Parameters**

| | |
|---|---|
| *num* | - 0 based index of row |
| *tbl* | - name of the table |

#### 7.18.2.6 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 the table (needed for python testing)

**Author**

Luka Rajcevic

**Parameters**

| | |
|---|---|
| *tbl_name* | - name of the table for which the header will be created |
| *attr_name* | - array of attribute names |
| *attr_value* | - values of attributes to be inserted |
| *_num* | - number of attributes |
| *_type* | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

**Returns**

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

#### 7.18.2.7 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**

| | |
|---|---|
| *src_table* | - name of the source table<br>• |
| *dest_table* | - table in which selection will be stored |
| *sel_query* | - array of operators, operands and attributes (postfix query) |
| *_num* | - number of attributes |
| *_type* | - array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.) |

**Returns**

EXIT_SUCCESS if ok, EXIT_ERROR otherwise

## 7.19 dm/dbman.c File Reference

```
#include "dbman.h"
#include "../mm/memoman.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.*
- int 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 the disk.*
- void AK_allocate_block_activity_modes ()

  *Allocation of an 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 the disk.*
- int fsize (FILE *fp)

  *Helper function to determine file size.*
- int AK_init_allocation_table ()

  *Function that initializes the allocation table, writes it to the disk and caches it 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 a 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 updates the 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 that writes a block to the 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 that 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 that allocates a new blocks for increasing extent size.*

- int AK_new_extent (int start_address, int old_size, int extent_type, AK_header *header)

  *Function that 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_E↩ XTENT_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 *contr_code)

  *Function that creates header and initalize 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 that inserts an 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 that 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 that 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 that 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 that deletes an extent between the first and the last block.*
- int AK_delete_segment (char *name, int type)
- int AK_init_disk_manager ()
- TestResult AK_allocationbit_test ()
- TestResult AK_allocationtable_test ()
- TestResult 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 containted in this wariables. If they don't match, then the error occured! 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 suceeded?" 0 means NO, 1 means YES.*

### 7.19.1 Detailed Description

Defines functions for the disk manager

### 7.19.2 Function Documentation

#### 7.19.2.1 AK_allocate_block_activity_modes()

```
void AK_allocate_block_activity_modes ( )
```

Allocation of an 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

#### 7.19.2.2 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 updates the last initialized index.

**Author**

Markus Schatten , rearranged by dv

**Returns**

EXIT_SUCCESS if the file has been written to disk, EXIT_ERROR otherwise

#### 7.19.2.3 AK_allocationbit_test()

```
TestResult AK_allocationbit_test ( )
```

#### 7.19.2.4 AK_allocationtable_dump()

```
int AK_allocationtable_dump (
            int verbosity )
```

Dumps the allocation table from the global allocation bit-vector onto standard output.

**Author**

dv

**Parameters**

| | |
|---|---|
| *verbosity* | level of verbosity (1 - minimal, 0 - no output) |

#### 7.19.2.5 AK_allocationtable_test()

```
TestResult AK_allocationtable_test ( )
```

#### 7.19.2.6 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**

| | |
|---|---|
| *verbosity* | level of verbosity (1 - verbose, 0 - minimal) |

#### 7.19.2.7 AK_blocktable_flush()

```
int AK_blocktable_flush ( )
```

Function flushes bitmask table to the disk.

**Author**

dv

**Returns**

EXIT_SUCCESS if the file has been written to the disk, EXIT_ERROR otherwise

#### 7.19.2.8 AK_blocktable_get()

```
int AK_blocktable_get ( )
```

Function gets allocation table from the disk.

**Author**

dv

**Returns**

EXIT_SUCCESS if the file has been taken from disk, EXIT_ERROR otherwise

#### 7.19.2.9 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**

| | |
|---|---|
| *header* | Pointer to header which will be copied into each block in blockSet |
| *blockSet* | Pointer to array of block addresses into which to copy header |
| *blockSetSize* | Number of blocks in blockSet |

**Returns**

number of performed header copy

#### 7.19.2.10 AK_create_header()

```
AK_header* AK_create_header (
            char * attribute_name,
            int type,
            int integrity,
            char * constr_name,
            char * contr_code )
```

Function that creates header and initalize integrity, constraint name and constraint code with parameter values of function.

**Author**

Matija Novak

**Parameters**

| | |
|---|---|
| *name* | name of the atribute |
| *type* | type of the atribute |
| *integrity* | standard integrity costraint |
| *constr_name* | extra integrity constraint name |
| *contr_code* | extra integrity costraint code |

**Returns**

AK_header

#### 7.19.2.11 AK_delete_block()

```
int AK_delete_block (
            int address )
```

Function that 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**

| | |
|---|---|
| *address* | address of the block to be deleted |

**Returns**

returns EXIT_SUCCESS if deletion successful, else EXIT_ERROR

#### 7.19.2.12 AK_delete_extent()

```
int AK_delete_extent (
            int begin,
            int end )
```

Function that deletes an extent between the first and the last block.

**Author**

Dejan Sambolić

**Parameters**

| | |
|---|---|
| *begin* | address of extent's first block |
| *end* | address of extent's last block |

**Returns**

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

#### 7.19.2.13 AK_delete_segment()

```
int AK_delete_segment (
            char * name,
            int type )
```

**Author**

Mislav Èakariæ, fixed by Josip Susnjara

**Parameters**

| | |
|---|---|
| *name* | name of the segment |
| *type* | type of the segment |

**Returns**

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

#### 7.19.2.14 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**

| | |
|---|---|
| *allocationSet* | Pointer to array which will be filled and represent the allocation set |
| *fromWhere* | Has meaning only if mode is SEQUENCE. It describes from which address searching starts. |
| *gaplength* | Tells how many used blocks can be tolerated in allocation set |
| *numRequestedBlocks* | Tells how many AK_free blocks have been requested |
| *mode* | Defines how to obtain set of indexes to AK_free addresses |
| *target* | 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

### 7.19.2.15 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 that allocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.

**Author**

dv

**Parameters**

| | |
|---|---|
| *start_address* | address (block number) to start searching for sufficient space |
| *desired_size* | number of desired blocks |
| *AK_allocation_set_mode* | a way of trying to fing AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND |
| *border* | number of allocated blocks gap |
| *target* | block address around which other blocks have to be searched |
| *header* | pointer to header that should be written to the new extent (all blocks) |
| *int* | gl gap size |

**Returns**

pointer to set of alocated block addresses

vars for loop [for]

if some blocks are not succesfully allocated, which means that the extend allocation has FAILED

### 7.19.2.16 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 that allocates a new blocks for increasing extent size.

**Author**

dv

**Parameters**

| | |
|---|---|
| *start_address* | first address of extent that is subject of increasing |
| *add_size* | number how many new blocks is to be added to existing extent |
| *AK_allocation_set_mode* | a way of trying to fing AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND |
| *border* | number of allocated blocks gap |
| *target* | block address around which other blocks have to be searched |
| *header* | pointer to header that should be written to the new extent (all blocks) |
| *int* | gl gap size |

**Returns**

pointer to set of alocated block addresses

#### 7.19.2.17 AK_init_allocation_table()

```
int AK_init_allocation_table ( )
```

Function that initializes the allocation table, writes it to the disk and caches it in memory.

**Author**

dv

**Returns**

EXIT_SUCCESS if the file has been written to disk, EXIT_ERROR otherwise

#### 7.19.2.18 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

#### 7.19.2.19 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

#### 7.19.2.20 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.

#### 7.19.2.21 AK_init_system_catalog()

```
int AK_init_system_catalog ( )
```

Function that 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

#### 7.19.2.22 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 that 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**

| | |
|---|---|
| *relation* | address of system table of relation in db_file |
| *attribute* | address of system table of attribute in db_file |
| *index* | address of system table of index in db_file |
| *view* | address of system table of view in db_file |
| *sequence* | address of system table of sequence in db_file |
| *function* | address of system table of function in db_file |
| *function_arguments* | address of system table of function_arguments in db_file |
| *trigger* | address of system table of trigger in db_file |
| *trigger_conditions* | address of system table of trigger_conditions in db_file |
| *db* | address of system table of db in db_file |
| *db_obj* | address of system table of db_obj in db_file |
| *user* | address of system table of user in db_file |
| *group* | address of system table of group in db_file |
| *user_group* | address of system table of users associated with groups in db_file |
| *user_right* | address of system table of user right in db_file |
| *group_right* | address of system table of group right in db_file |
| *constraint* | address of system table of constraint in db_file |
| *constraintNull* | address of system table of constraintNull in db_file |
| *constraintCheck* | system table address for check constraint |
| *reference* | 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

#### 7.19.2.23 AK_insert_entry()

```
void AK_insert_entry (
            AK_block * block_address,
            int type,
            void * entry_data,
            int i )
```

Function that inserts an 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**

| | |
|---|---|
| *block_adress* | adress of a block in which we want insert data |
| *type* | type of entry_data |
| *entry_data* | (char) data which is inserted, can be int but must first be converted to char |
| *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 orginal 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

#### 7.19.2.24 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**

| | |
|---|---|
| *block* | pointer to the block of memory to fill |
| *value* | int value to be set |
| *num* | number of ints in the block of memory to be set |

**Returns**

No return value

#### 7.19.2.25 AK_new_extent()

```
int AK_new_extent (
            int start_address,
            int old_size,
            int extent_type,
            AK_header * header )
```

Function that 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 Laktašiæ (fixed header BUG), refurbished by dv

**Parameters**

| | |
|---|---|
| *start_address* | address (block number) to start searching for sufficient space |
| *old_size* | size of previous extent in same segment (in blocks) |
| *extent_type* | type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP |
| *header* | 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

#### 7.19.2.26 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**

| | |
|---|---|
| *name* | (character pointer) name of segment |
| *type* | segment type (possible values: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP) |
| *header* | (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

#### 7.19.2.27 AK_print_block()

```
int AK_print_block (
            AK_block * block,
            int num,
            char * gg,
            FILE * fpp )
```

Function that dumps a block.

**Author**

dv

**Returns**

nothing

#### 7.19.2.28 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 by dv and Domagoj Šitum (thread-safe enabled)

**Parameters**

| | |
|---|---|
| *address* | block number (address) |

**Returns**

pointer to block allocated in memory

#### 7.19.2.29 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

#### 7.19.2.30 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**

| | |
|---|---|
| *relation* | relation in database |
| *attribute* | attribute in databse |
| *index* | index in database |
| *view* | view in database |
| *sequence* | sequence in database |
| *function* | function in database |
| *function_arguments* | functional_arguments in databse |
| *trigger* | trigger in database |
| *trigger_conditions* | trigger conditions in databse |
| *db* | database |
| *db_obj* | database object |
| *user* | user in database |
| *group* | group in database |
| *user_group* | user associated with group in database |
| *user_right* | user right in database |
| *group_right* | group right in database |
| *constraint* | constraint in database |
| *constraintNull* | Null constraint in database |
| *constraintCheck* | Check constraint in database |
| *reference* | reference database |

**Returns**

EXIT_SUCCESS

#### 7.19.2.31 AK_thread_safe_block_access_test()

```
TestResult 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

#### 7.19.2.32 AK_write_block()

```
int AK_write_block (
            AK_block * block )
```

Function that writes a block to the DB file. Database file is opened. Position is set to provided address block. Block is written to provided address. Completely thread-safe.

Function that writes the new value in block when index is updated.

**Author**

Markus Schatten, updated by Domagoj Šitum (thread-safe enabled)

**Parameters**

| | |
|---|---|
| *block* | poiner to block allocated in memory to write |

**Returns**

EXIT_SUCCESS if successful, EXIT_ERROR otherwise

#### 7.19.2.33 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

#### 7.19.2.34 fsize()

```
int fsize (
            FILE * fp )
```

Helper function to determine file size.

**Returns**

file size

### 7.19.3 Variable Documentation

#### 7.19.3.1 fileLockMutex

```
pthread_mutex_t fileLockMutex = PTHREAD_MUTEX_INITIALIZER
```

#### 7.19.3.2 test_lastCharacterWritten

```
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 containted in this wariables. If they don't match, then the error occured! It is assumed that the same block is being written to and read from (just like AK_thread_safe_block_access_test function works!)

#### 7.19.3.3 test_threadSafeBlockAccessSucceeded

```
test_threadSafeBlockAccessSucceeded = 1
```

Used in combination with test_lastCharacterWritten. Will give the answer to question: "Has AK_thread_safe_↩ block_access_test suceeded?" 0 means NO, 1 means YES.

## 7.20 dm/dbman.h File Reference

```
#include "../auxi/test.h"
#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 uninteruptable, 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.*

## 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 od allocation table LOWER - set tries to place itself to lower part od 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 a block.*
- TestResult AK_allocationbit_test ()
- TestResult 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 that allocates a 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 that 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 updates the last initialized index.*
- AK_block * AK_init_block ()

  *Function that initializes new block.*
- int 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 the disk.*
- TestResult 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 the disk.*
- int fsize (FILE *fp)

  *Helper function to determine file size.*
- int AK_init_allocation_table ()

  *Function that initializes the allocation table, writes it to the disk and caches it 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 that writes a block to the 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 that 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_E↩ XTENT_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 *contr_code)

  *Function that creates header and initalize 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 that inserts an 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 that 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 that 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 that 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 that deletes an extent between the first and the last block.*

- 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

### 7.20.1 Detailed Description

Header file that contains all defines, includes and data structures for the disk manager of Kalashnikov DB

### 7.20.2 Macro Definition Documentation

#### 7.20.2.1 AK_ALLOCATION_TABLE_SIZE

```
#define AK_ALLOCATION_TABLE_SIZE sizeof(AK_blocktable)
```

Holds size of allocation table.

**Author**

dv

#### 7.20.2.2 BITCLEAR

```
#define BITCLEAR(
              a,
              b ) ((a)[BITSLOT(b)] &= ~BITMASK(b))
```

#### 7.20.2.3 BITMASK

```
#define BITMASK(
              b ) (1 << ((b) % CHAR_BIT))
```

#### 7.20.2.4 BITNSLOTS

```
#define BITNSLOTS(
              nb ) ((int)(nb + CHAR_BIT - 1) / CHAR_BIT)
```

#### 7.20.2.5 BITSET

```
#define BITSET(
              a,
              b ) ((a)[BITSLOT(b)] |= BITMASK(b))
```

#### 7.20.2.6 BITSLOT

```
#define BITSLOT(
              b ) ((int)((b) / CHAR_BIT))
```

#### 7.20.2.7 BITTEST

```
#define BITTEST(
              a,
              b ) ((a)[BITSLOT(b)] & BITMASK(b))
```

#### 7.20.2.8 CHAR_IN_LINE

```
#define CHAR_IN_LINE 80
```

How many characters could line contain.

**Author**

dv

#### 7.20.2.9 DB_FILE_BLOCKS_NUM_EX

```
#define DB_FILE_BLOCKS_NUM_EX (int)(1024 * 1024 * DB_FILE_SIZE_EX / sizeof(AK_block))
```

#### 7.20.2.10 DB_FILE_SIZE_EX

```
#define DB_FILE_SIZE_EX 200
```

#### 7.20.2.11 MAX_BLOCK_INIT_NUM

```
#define MAX_BLOCK_INIT_NUM MAX_CACHE_MEMORY
```

How many blocks would be initially allocated.

**Author**

dv

#### 7.20.2.12 SEGMENTLENGTH

```
#define SEGMENTLENGTH( ) (BITNSLOTS(DB_FILE_BLOCKS_NUM) + 2*sizeof(int))
```

### 7.20.3 Enumeration Type Documentation

#### 7.20.3.1 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 od allocation table LOWER - set tries to place itself to lower part od allocation table AROUND - set tries to place itself around targeted index.

**Author**

dv

**Enumerator**

| | |
|---|---|
| allocationSEQUENCE | |
| allocationUPPER | |
| allocationLOWER | |
| allocationAROUND | |
| allocationNOMODE | |

### 7.20.4 Function Documentation

#### 7.20.4.1 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 updates the last initialized index.

**Author**

Markus Schatten , rearranged by dv

**Returns**

EXIT_SUCCESS if the file has been written to disk, EXIT_ERROR otherwise

#### 7.20.4.2 AK_allocationbit_test()

```
TestResult AK_allocationbit_test ( )
```

#### 7.20.4.3 AK_allocationtable_dump()

```
int AK_allocationtable_dump (
            int verbosity )
```

Dumps the allocation table from the global allocation bit-vector onto standard output.

**Author**

dv

**Parameters**

| | |
|---|---|
| *verbosity* | level of verbosity (1 - minimal, 0 - no output) |

#### 7.20.4.4 AK_allocationtable_test()

```
TestResult AK_allocationtable_test ( )
```

#### 7.20.4.5 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**

| | |
|---|---|
| *verbosity* | level of verbosity (1 - verbose, 0 - minimal) |

#### 7.20.4.6 AK_blocktable_flush()

```
int AK_blocktable_flush ( )
```

Function flushes bitmask table to the disk.

**Author**

dv

**Returns**

EXIT_SUCCESS if the file has been written to the disk, EXIT_ERROR otherwise

#### 7.20.4.7 AK_blocktable_get()

```
int AK_blocktable_get ( )
```

Function gets allocation table from the disk.

**Author**

dv

**Returns**

EXIT_SUCCESS if the file has been taken from disk, EXIT_ERROR otherwise

#### 7.20.4.8 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**

| | |
|---|---|
| *header* | Pointer to header which will be copied into each block in blockSet |
| *blockSet* | Pointer to array of block addresses into which to copy header |
| *blockSetSize* | Number of blocks in blockSet |

**Returns**

number of performed header copy

#### 7.20.4.9 AK_create_header()

```
AK_header* AK_create_header (
            char * attribute_name,
            int type,
            int integrity,
            char * constr_name,
            char * contr_code )
```

Function that creates header and initalize integrity, constraint name and constraint code with parameter values of function.

**Author**

Matija Novak

**Parameters**

| | |
|---|---|
| *name* | name of the atribute |
| *type* | type of the atribute |
| *integrity* | standard integrity costraint |
| *constr_name* | extra integrity constraint name |
| *contr_code* | extra integrity costraint code |

**Returns**

AK_header

#### 7.20.4.10 AK_delete_block()

```
int AK_delete_block (
            int address )
```

Function that 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**

| | |
|---|---|
| *address* | address of the block to be deleted |

**Returns**

returns EXIT_SUCCESS if deletion successful, else EXIT_ERROR

#### 7.20.4.11 AK_delete_extent()

```
int AK_delete_extent (
            int begin,
            int end )
```

Function that deletes an extent between the first and the last block.

**Author**

Dejan Sambolić

**Parameters**

| | |
|---|---|
| *begin* | address of extent's first block |
| *end* | address of extent's last block |

**Returns**

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

#### 7.20.4.12 AK_delete_segment()

```
int AK_delete_segment (
            char * name,
            int type )
```

**Author**

Mislav Èakariæ, fixed by Josip Susnjara

**Parameters**

| | |
|---|---|
| *name* | name of the segment |
| *type* | type of the segment |

**Returns**

EXIT_SUCCESS if extent has been successfully deleted, EXIT_ERROR otherwise

#### 7.20.4.13 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**

| | |
|---|---|
| *allocationSet* | Pointer to array which will be filled and represent the allocation set |
| *fromWhere* | Has meaning only if mode is SEQUENCE. It describes from which address searching starts. |
| *gaplength* | Tells how many used blocks can be tolerated in allocation set |
| *numRequestedBlocks* | Tells how many AK_free blocks have been requested |
| *mode* | Defines how to obtain set of indexes to AK_free addresses |
| *target* | 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

#### 7.20.4.14 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 that allocates new extent of blocks. Number of blocks is not ordered as well as a way of search for them.

**Author**

dv

**Parameters**

| | |
|---|---|
| *start_address* | address (block number) to start searching for sufficient space |
| *desired_size* | number of desired blocks |
| *AK_allocation_set_mode* | a way of trying to fing AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND |
| *border* | number of allocated blocks gap |
| *target* | block address around which other blocks have to be searched |
| *header* | pointer to header that should be written to the new extent (all blocks) |
| *int* | gl gap size |

**Returns**

pointer to set of alocated block addresses

vars for loop [for]

if some blocks are not succesfully allocated, which means that the extend allocation has FAILED

#### 7.20.4.15 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 that allocates a new blocks for increasing extent size.

**Author**

dv

**Parameters**

| | |
|---|---|
| *start_address* | first address of extent that is subject of increasing |
| *add_size* | number how many new blocks is to be added to existing extent |
| *AK_allocation_set_mode* | a way of trying to fing AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND |
| *border* | number of allocated blocks gap |
| *target* | block address around which other blocks have to be searched |
| *header* | pointer to header that should be written to the new extent (all blocks) |
| *int* | gl gap size |

**Returns**

pointer to set of alocated block addresses

#### 7.20.4.16 AK_init_allocation_table()

```
int AK_init_allocation_table ( )
```

Function that initializes the allocation table, writes it to the disk and caches it in memory.

**Author**

dv

**Returns**

EXIT_SUCCESS if the file has been written to disk, EXIT_ERROR otherwise

#### 7.20.4.17 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

#### 7.20.4.18 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

#### 7.20.4.19 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.

#### 7.20.4.20 AK_init_system_catalog()

```
int AK_init_system_catalog ( )
```

Function that 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

#### 7.20.4.21 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 that 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**

| | |
|---|---|
| *relation* | address of system table of relation in db_file |
| *attribute* | address of system table of attribute in db_file |
| *index* | address of system table of index in db_file |
| *view* | address of system table of view in db_file |
| *sequence* | address of system table of sequence in db_file |
| *function* | address of system table of function in db_file |
| *function_arguments* | address of system table of function_arguments in db_file |
| *trigger* | address of system table of trigger in db_file |
| *trigger_conditions* | address of system table of trigger_conditions in db_file |
| *db* | address of system table of db in db_file |
| *db_obj* | address of system table of db_obj in db_file |
| *user* | address of system table of user in db_file |
| *group* | address of system table of group in db_file |
| *user_group* | address of system table of users associated with groups in db_file |
| *user_right* | address of system table of user right in db_file |
| *group_right* | address of system table of group right in db_file |
| *constraint* | address of system table of constraint in db_file |
| *constraintNull* | address of system table of constraintNull in db_file |
| *constraintCheck* | system table address for check constraint |
| *reference* | 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

#### 7.20.4.22 AK_insert_entry()

```
void AK_insert_entry (
            AK_block * block_address,
            int type,
            void * entry_data,
            int i )
```

Function that inserts an 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**

| | |
|---|---|
| *block_adress* | adress of a block in which we want insert data |
| *type* | type of entry_data |
| *entry_data* | (char) data which is inserted, can be int but must first be converted to char |
| *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 orginal 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

#### 7.20.4.23 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**

| | |
|---|---|
| *block* | pointer to the block of memory to fill |
| *value* | int value to be set |
| *num* | number of ints in the block of memory to be set |

**Returns**

No return value

#### 7.20.4.24 AK_new_extent()

```
int AK_new_extent (
            int start_address,
            int old_size,
            int extent_type,
            AK_header * header )
```

Function that 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 Laktašiæ (fixed header BUG), refurbished by dv

**Parameters**

| | |
|---|---|
| *start_address* | address (block number) to start searching for sufficient space |
| *old_size* | size of previous extent in same segment (in blocks) |
| *extent_type* | type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP |
| *header* | 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

#### 7.20.4.25 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**

| | |
|---|---|
| *name* | (character pointer) name of segment |
| *type* | segment type (possible values: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP) |
| *header* | (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

#### 7.20.4.26 AK_print_block()

```
int AK_print_block (
            AK_block * block,
            int num,
            char * gg,
            FILE * fpp )
```

Function that dumps a block.

**Author**

dv

**Returns**

nothing

#### 7.20.4.27 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 by dv and Domagoj Šitum (thread-safe enabled)

**Parameters**

| | |
|---|---|
| *address* | block number (address) |

**Returns**

pointer to block allocated in memory

#### 7.20.4.28 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

#### 7.20.4.29 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**

| | |
|---|---|
| *relation* | relation in database |
| *attribute* | attribute in databse |
| *index* | index in database |
| *view* | view in database |
| *sequence* | sequence in database |
| *function* | function in database |
| *function_arguments* | functional_arguments in databse |
| *trigger* | trigger in database |
| *trigger_conditions* | trigger conditions in databse |
| *db* | database |
| *db_obj* | database object |
| *user* | user in database |
| *group* | group in database |
| *user_group* | user associated with group in database |
| *user_right* | user right in database |
| *group_right* | group right in database |
| *constraint* | constraint in database |
| *constraintNull* | Null constraint in database |
| *constraintCheck* | Check constraint in database |
| *reference* | reference database |

**Returns**

EXIT_SUCCESS

#### 7.20.4.30 AK_thread_safe_block_access_test()

```
TestResult 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

#### 7.20.4.31 AK_write_block()

```
int AK_write_block (
            AK_block * block )
```

Function that writes a block to the 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**

| | |
|---|---|
| *block* | poiner to block allocated in memory to write |

**Returns**

EXIT_SUCCESS if successful, EXIT_ERROR otherwise

#### 7.20.4.32 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

#### 7.20.4.33 fsize()

```
int fsize (
            FILE * fp )
```

Helper function to determine file size.

**Returns**

file size

### 7.20.5 Variable Documentation

#### 7.20.5.1 AK_allocationbit

```
AK_allocationbit
```

Global variable that holds allocation bit-vector.

**Author**

dv

#### 7.20.5.2 AK_block_activity_info

```
AK_block_activity* AK_block_activity_info
```

#### 7.20.5.3 db

```
db
```

Variable that defines the DB file file handle.

**Author**

Markus Schatten

#### 7.20.5.4 db_file_size

```
db_file_size
```

Variable that defines the size of the DB file (in blocks)

**Author**

Markus Schatten

#### 7.20.5.5 dbmanFileLock

```
AK_synchronization_info* dbmanFileLock
```

## 7.21 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 that generates GUID.*
- int AK_folder_exists (char *foldername)

  *Function that checks if folder blobs already exists.*
- int AK_mkdir (const char *path)

  *Function that creates 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 that checks if folder blobs exists.*
- void AK_split_path_file (char **p, char **f, char *pf)

  *Function that splits a 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 that imports large objects to database.*
- int AK_lo_export (char *oid, char *filepath)

  *Function that retrieves large objects.*
- int AK_lo_unlink (char *oid)

  *Function that deletes large objects.*
- TestResult AK_lo_test ()

  *Tests.*

### Variables

- int success = 0
- int failed = 0

### 7.21.1 Detailed Description

Provides functions for manipulations of binary large objects

### 7.21.2 Function Documentation

#### 7.21.2.1 AK_check_folder_blobs()

```
int AK_check_folder_blobs ( )
```

Function that checks if folder blobs exists.

**Author**

Samuel Picek

**Returns**

OID (object ID)

#### 7.21.2.2 AK_clear_all_newline()

```
char* AK_clear_all_newline (
            char * s )
```

#### 7.21.2.3 AK_concat()

```
char* AK_concat (
            char * s1,
            char * s2 )
```

Function for AK_concatinating 2 strings.

**Author**

Samuel Picek

**Returns**

returns new string

#### 7.21.2.4 AK_copy()

```
int AK_copy (
            const char * from,
            const char * to )
```

#### 7.21.2.5 AK_File_Metadata_malloc()

```
AK_File_Metadata AK_File_Metadata_malloc ( )
```

#### 7.21.2.6 AK_folder_exists()

```
int AK_folder_exists (
            char * foldername )
```

Function that checks if folder blobs already exists.

**Author**

Samuel Picek

**Returns**

returns 0 for true and 1 for false

#### 7.21.2.7 AK_GUID()

```
char* AK_GUID ( )
```

Function that generates GUID.

**Author**

Samuel Picek

**Returns**

returns globaly universal identifier based on kernel implementation

#### 7.21.2.8 AK_lo_export()

```
int AK_lo_export (
             char * oid,
             char * filepath )
```

Function that retrieves large objects.

**Author**

Samuel Picek

**Returns**

returns 0 for true and 1 for false

#### 7.21.2.9 AK_lo_import()

```
char* AK_lo_import (
             char * filepath )
```

Function that imports large objects to database.

**Author**

Samuel Picek

**Returns**

OID (object ID)

#### 7.21.2.10 AK_lo_test()

```
TestResult AK_lo_test ( )
```

Tests.

**Author**

Samuel Picek

#### 7.21.2.11 AK_lo_unlink()

```
int AK_lo_unlink (
            char * oid )
```

Function that deletes large objects.

**Author**

Samuel Picek

**Returns**

OID (object ID)

#### 7.21.2.12 AK_mkdir()

```
int AK_mkdir (
            const char * path )
```

Function that creates new folder.

**Author**

Samuel Picek

**Returns**

returns 0 for true and 1 for false

#### 7.21.2.13 AK_read_metadata()

```
AK_File_Metadata AK_read_metadata (
            char * oid )
```

#### 7.21.2.14 AK_split_path_file()

```
void AK_split_path_file (
            char ** p,
            char ** f,
            char * pf )
```

Function that splits a path from filename.

**Author**

Samuel Picek

**Returns**

void

#### 7.21.2.15 AK_write_metadata()

```
int AK_write_metadata (
            char * oid,
            AK_File_Metadata meta )
```

### 7.21.3 Variable Documentation

#### 7.21.3.1 failed

```
int failed = 0
```

#### 7.21.3.2 success

```
int success = 0
```

## 7.22 file/blobs.h File Reference

```
#include "../auxi/test.h"
#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 that creates 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 that splits a path from filename.*
- char * AK_GUID ()

  *Function that generates GUID.*
- int AK_folder_exists (char *foldername)

  *Function that checks if folder blobs already exists.*
- int AK_check_folder_blobs ()

  *Function that checks 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 that imports large objects to database.*
- int AK_lo_export (char *oid, char *filepath)

  *Function that retrieves large objects.*
- int AK_lo_unlink (char *oid)

  *Function that deletes large objects.*
- TestResult AK_lo_test ()

  *Tests.*

### 7.22.1 Detailed Description

Provides data structures, functions and defines for manipulating blobs

### 7.22.2 Typedef Documentation

#### 7.22.2.1 AK_File_Metadata

```
typedef struct _file_metadata* AK_File_Metadata
```

#### 7.22.2.2 AK_Metadata

```
typedef struct _file_metadata AK_Metadata
```

## 7.22.3 Function Documentation

### 7.22.3.1 AK_check_folder_blobs()

```
int AK_check_folder_blobs ( )
```

Function that checks if folder blobs exists.

**Author**

Samuel Picek

**Returns**

OID (object ID)

### 7.22.3.2 AK_clear_all_newline()

```
char* AK_clear_all_newline (
            char * str )
```

### 7.22.3.3 AK_concat()

```
char* AK_concat (
            char * s1,
            char * s2 )
```

Function for AK_concatinating 2 strings.

**Author**

Samuel Picek

**Returns**

returns new string

#### 7.22.3.4 AK_copy()

```
int AK_copy (
            const char * from,
            const char * to )
```

#### 7.22.3.5 AK_File_Metadata_malloc()

```
AK_File_Metadata AK_File_Metadata_malloc ( )
```

#### 7.22.3.6 AK_folder_exists()

```
int AK_folder_exists (
            char * foldername )
```

Function that checks if folder blobs already exists.

**Author**

Samuel Picek

**Returns**

returns 0 for true and 1 for false

#### 7.22.3.7 AK_GUID()

```
char* AK_GUID ( )
```

Function that generates GUID.

**Author**

Samuel Picek

**Returns**

returns globaly universal identifier based on kernel implementation

#### 7.22.3.8 AK_lo_export()

```
int AK_lo_export (
             char * oid,
             char * filepath )
```

Function that retrieves large objects.

**Author**

Samuel Picek

**Returns**

returns 0 for true and 1 for false

#### 7.22.3.9 AK_lo_import()

```
char* AK_lo_import (
             char * filepath )
```

Function that imports large objects to database.

**Author**

Samuel Picek

**Returns**

OID (object ID)

#### 7.22.3.10 AK_lo_test()

```
TestResult AK_lo_test ( )
```

Tests.

**Author**

Samuel Picek

#### 7.22.3.11 AK_lo_unlink()

```
int AK_lo_unlink (
            char * oid )
```

Function that deletes large objects.

**Author**

Samuel Picek

**Returns**

OID (object ID)

#### 7.22.3.12 AK_mkdir()

```
int AK_mkdir (
            const char * path )
```

Function that creates new folder.

**Author**

Samuel Picek

**Returns**

returns 0 for true and 1 for false

#### 7.22.3.13 AK_read_metadata()

```
AK_File_Metadata AK_read_metadata (
            char * oid )
```

#### 7.22.3.14 AK_split_path_file()

```
void AK_split_path_file (
            char ** p,
            char ** f,
            char * pf )
```

Function that splits a path from filename.

**Author**

Samuel Picek

**Returns**

void

#### 7.22.3.15 AK_write_metadata()

```
int AK_write_metadata (
            char * oid,
            AK_File_Metadata meta )
```

## 7.23 file/fileio.c File Reference

`#include "fileio.h"`
Include dependency graph for fileio.c:

### Functions

- void AK_Insert_New_Element_For_Update (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore, int newconstraint)

  *!! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. 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_Update_Existing_Element (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore)

  *Used to add a constraint attribute which will define what element gets updated when the operation is executed.*
- void AK_Insert_New_Element (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore)

  *Used to add a 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.*
- TestResult AK_fileio_test ()

### 7.23.1 Detailed Description

Provides functions for file input/output

## 7.23.2 Function Documentation

### 7.23.2.1 AK_delete_row()

```
int AK_delete_row (
            struct list_node * row_root )
```

Function deletes rows.

**Author**

Matija Novak, Dejan Frankovic (added referential integrity)

**Parameters**

| | |
|---|---|
| *row_root* | elements of one row @returs EXIT_SUCCESS if success |

### 7.23.2.2 AK_delete_row_by_id()

```
void AK_delete_row_by_id (
            int id,
            char * tableName )
```

Function deletes row by id.

**Author**

Dražen Bandić

**Parameters**

| | |
|---|---|
| *id* | id of row |
| *tableName* | name of table to delete the row |

### 7.23.2.3 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**

| | |
|---|---|
| *temp_block* | block to work with |
| *row_list* | list of elements which contain data for delete or update |

**Returns**

No return value

#### 7.23.2.4 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**

| | |
|---|---|
| *row_root* | elements of one row |
| *del* | - DELETE or UPDATE |

**Returns**

EXIT_SUCCESS if success

#### 7.23.2.5 AK_fileio_test()

```
TestResult AK_fileio_test ( )
```

#### 7.23.2.6 AK_Insert_New_Element()

```
void AK_Insert_New_Element (
            int newtype,
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore )
```

Used to add a 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**

| | |
|---|---|
| *newtype* | type of the data |
| *data* | the data |
| *table* | table name |
| *attribute_name* | attribute name |
| *element* | element after we which insert the new element |
| *constraint* | is NEW_VALUE |

**Returns**

No return value

#### 7.23.2.7 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 )
```

!! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. 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**

| | |
|---|---|
| *newtype* | type of the data |
| *data* | the data |
| *table* | table name |
| *attribute_name* | attribute name |
| *element* | element after we which insert the new element |
| *constraint* | NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for |

**Returns**

No return value

#### 7.23.2.8 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**

| | |
|---|---|
| *row_root* | list of elements which contain data of one row |

**Returns**

EXIT_SUCCESS if success else EXIT_ERROR

#### 7.23.2.9 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**

| | |
|---|---|
| *row_root* | list of elements to insert |
| *temp_block* | block in which we insert data |

**Returns**

EXIT SUCCES if success

#### 7.23.2.10 AK_Update_Existing_Element()

```
void AK_Update_Existing_Element (
            int newtype,
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore )
```

Used to add a constraint attribute which will define what element gets updated when the operation is executed.

**Author**

Igor Rinkovec

**Parameters**

| | |
|---|---|
| *newtype* | type of the data |
| *data* | the data |
| *table* | table name |
| *attribute_name* | attribute name |
| *element* | element after we which insert the new element |
| *constraint* | is NEW_VALUE |

**Returns**

No return value

#### 7.23.2.11 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**

| | |
|---|---|
| *row_root* | elements of one row |

**Returns**

EXIT_SUCCESS if success

#### 7.23.2.12 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**

| | |
|---|---|
| *temp_block* | block to work with |
| *row_list* | list of elements which contain data for delete or update |

**Returns**

No return value

## 7.24 file/fileio.h File Reference

```
#include "../auxi/test.h"
#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 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)

*!! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. 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)

  *Used to add a 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.*
- TestResult AK_fileio_test ()
- void AK_delete_row_by_id (int id, char ∗tableName)

  *Function deletes row by id.*

## 7.24.1 Detailed Description

Header file provides functions and defines for file input/output

## 7.24.2 Function Documentation

### 7.24.2.1 AK_delete_row()

```
int AK_delete_row (
            struct list_node * row_root )
```

Function deletes rows.

**Author**

Matija Novak, Dejan Frankovic (added referential integrity)

**Parameters**

| | |
|---|---|
| *row_root* | elements of one row @returs EXIT_SUCCESS if success |

#### 7.24.2.2 AK_delete_row_by_id()

```
void AK_delete_row_by_id (
            int id,
            char * tableName )
```

Function deletes row by id.

**Author**

Dražen Bandić

**Parameters**

| | |
|---|---|
| *id* | id of row |
| *tableName* | name of table to delete the row |

#### 7.24.2.3 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**

| | |
|---|---|
| *temp_block* | block to work with |
| *row_list* | list of elements which contain data for delete or update |

**Returns**

No return value

#### 7.24.2.4 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**

| | |
|---|---|
| *row_root* | elements of one row |
| *del* | - DELETE or UPDATE |

**Returns**

EXIT_SUCCESS if success

#### 7.24.2.5 AK_fileio_test()

```
TestResult AK_fileio_test ( )
```

#### 7.24.2.6 AK_Insert_New_Element()

```
void AK_Insert_New_Element (
            int newtype,
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore )
```

Used to add a 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**

| | |
|---|---|
| *newtype* | type of the data |
| *data* | the data |
| *table* | table name |
| *attribute_name* | attribute name |
| *element* | element after we which insert the new element |
| *constraint* | is NEW_VALUE |

**Returns**

No return value

#### 7.24.2.7 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 )
```

!! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. 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**

| | |
|---|---|
| *newtype* | type of the data |
| *data* | the data |
| *table* | table name |
| *attribute_name* | attribute name |
| *element* | element after we which insert the new element |
| *constraint* | NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for |

**Returns**

No return value

#### 7.24.2.8 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**

| | |
|---|---|
| *row_root* | list of elements which contain data of one row |

**Returns**

EXIT_SUCCESS if success else EXIT_ERROR

#### 7.24.2.9 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**

| | |
|---|---|
| *row_root* | list of elements to insert |
| *temp_block* | block in which we insert data |

**Returns**

EXIT SUCCES if success

#### 7.24.2.10 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**

| | |
|---|---|
| *row_root* | elements of one row |

**Returns**

EXIT_SUCCESS if success

#### 7.24.2.11 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**

| | |
|---|---|
| *temp_block* | block to work with |
| *row_list* | list of elements which contain data for delete or update |

**Returns**

No return value

## 7.25 file/files.c File Reference

```
#include "files.h"
#include <pthread.h>
```

Include dependency graph for files.c:

### Functions

- int AK_initialize_new_segment (char *name, int type, AK_header *header)

  *Function that initializes a 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 that initializes a new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.*
- TestResult AK_files_test ()

  *Test function.*

### Variables

- pthread_mutex_t fileMut = PTHREAD_MUTEX_INITIALIZER

### 7.25.1 Detailed Description

Header file provides functions for file management

### 7.25.2 Function Documentation

#### 7.25.2.1 AK_files_test()

```
TestResult AK_files_test ( )
```

Test function.

**Author**

Unknown

**Returns**

No return value

#### 7.25.2.2 AK_initialize_new_index_segment()

```
int AK_initialize_new_index_segment (
            char * name,
            char * table_id,
            int attr_id,
            AK_header * header )
```

Function that initializes a 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**

| | |
|---|---|
| *name* | segment name |
| *type* | segment type |
| *header* | pointer to header that should be written to the new extent (all blocks) |

**Returns**

start address of new segment

#### 7.25.2.3 AK_initialize_new_segment()

```
int AK_initialize_new_segment (
            char * name,
            int type,
            AK_header * header )
```

Function that initializes a 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**

| | |
|---|---|
| *name* | segment name |
| *type* | segment type |
| *header* | pointer to header that should be written to the new extent (all blocks) |

**Returns**

start address of new segment

### 7.25.3 Variable Documentation

#### 7.25.3.1 fileMut

```
pthread_mutex_t fileMut = PTHREAD_MUTEX_INITIALIZER
```

## 7.26 file/files.h File Reference

```
#include "../auxi/test.h"
#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 that initializes a 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 that initializes a new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.*
- TestResult AK_files_test ()

  *Test function.*

### 7.26.1 Detailed Description

Header file that provides functions and defines for file management

### 7.26.2 Function Documentation

#### 7.26.2.1 AK_files_test()

```
TestResult AK_files_test ( )
```

Test function.

**Author**

Unknown

**Returns**

No return value

#### 7.26.2.2 AK_initialize_new_index_segment()

```
int AK_initialize_new_index_segment (
            char * name,
            char * table_id,
            int attr_id,
            AK_header * header )
```

Function that initializes a 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**

| | |
|---|---|
| *name* | segment name |
| *type* | segment type |
| *header* | pointer to header that should be written to the new extent (all blocks) |

**Returns**

start address of new segment

#### 7.26.2.3 AK_initialize_new_segment()

```
int AK_initialize_new_segment (
            char * name,
            int type,
            AK_header * header )
```

Function that initializes a 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**

| | |
|---|---|
| *name* | segment name |
| *type* | segment type |
| *header* | pointer to header that should be written to the new extent (all blocks) |

**Returns**

start address of new segment

## 7.27 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)

*Function that 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 that deallocates memory used by the search result returned by AK_search_unsorted.*

- TestResult AK_filesearch_test ()

  *Function that tests file search.*

## 7.27.1 Detailed Description

Provides functions for file searching

## 7.27.2 Function Documentation

### 7.27.2.1 AK_deallocate_search_result()

```
void AK_deallocate_search_result (
            search_result srResult )
```

Function that deallocates memory used by the search result returned by AK_search_unsorted.

**Author**

Miroslav Policki

**Parameters**

| | |
|---|---|
| *srResult* | search result |

**Returns**

No return value

### 7.27.2.2 AK_filesearch_test()

```
TestResult AK_filesearch_test ( )
```

Function that tests file search.

**Author**

Miroslav Policki

**Returns**

No return value

#### 7.27.2.3 AK_search_unsorted()

```
search_result AK_search_unsorted (
             char * szRelation,
             search_params * aspParams,
             int iNum_search_params )
```

Function that 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**

| | |
|---|---|
| *szRelation* | relation name |
| *aspParams* | array of search parameters |
| *iNum_search_params* | 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

## 7.28 file/filesearch.h File Reference

```
#include "../auxi/test.h"
#include "../mm/memoman.h"
#include "files.h"
#include "../auxi/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)

  *Function that 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 that deallocates memory used by the search result returned by AK_search_unsorted.*
- TestResult AK_filesearch_test ()

  *Function that tests file search.*

### 7.28.1 Detailed Description

Header file provides data structures, functions and defines for file searching

### 7.28.2 Macro Definition Documentation

#### 7.28.2.1 SEARCH_ALL

```
#define SEARCH_ALL 1
```

#### 7.28.2.2 SEARCH_NULL

```
#define SEARCH_NULL 0
```

#### 7.28.2.3 SEARCH_PARTICULAR

```
#define SEARCH_PARTICULAR 2
```

#### 7.28.2.4 SEARCH_RANGE

```
#define SEARCH_RANGE 3
```

### 7.28.3 Function Documentation

#### 7.28.3.1 AK_deallocate_search_result()

```
void AK_deallocate_search_result (
            search_result srResult )
```

Function that deallocates memory used by the search result returned by AK_search_unsorted.

**Author**

Miroslav Policki

**Parameters**

| | |
|---|---|
| *srResult* | search result |

**Returns**

No return value

#### 7.28.3.2 AK_filesearch_test()

```
TestResult AK_filesearch_test ( )
```

Function that tests file search.

**Author**

Miroslav Policki

**Returns**

No return value

#### 7.28.3.3 AK_search_unsorted()

```
search_result AK_search_unsorted (
            char * szRelation,
            search_params * aspParams,
            int iNum_search_params )
```

Function that 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**

| | |
|---|---|
| *szRelation* | relation name |
| *aspParams* | array of search parameters |
| *iNum_search_params* | 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

# 7.29 file/filesort.c File Reference

```
#include "filesort.h"
```
Include dependency graph for filesort.c:

## Functions

- int AK_get_total_headers (AK_block *iBlock)

  *Function that returns the total number of headers in the block.*
- int AK_get_header_number (AK_block *iBlock, char *attribute_name)

  *Function that returns the number of header in the block which to sort.*
- int AK_get_num_of_tuples (AK_block *iBlock)

  *Function that returns tuples number in block.*
- int AK_sort_segment (char *srcTable, char *destTable, struct list_node *attributes)

  *Function that sorts a segment.*
- void AK_reset_block (AK_block *block)

  *Function that resets block.*
- void AK_block_sort (AK_block *iBlock, char *attribute_name)

  *Function that sorts the given block.*
- TestResult AK_filesort_test ()

## 7.29.1 Function Documentation

### 7.29.1.1 AK_block_sort()

```
void AK_block_sort (
            AK_block * iBlock,
            char * attribute_name )
```

Function that sorts the given block.

**Author**

Bakoš Nikola

**Version**

v1.0

**Parameters**

| | |
|---|---|
| *iBlock* | block to be sorted |

**Returns**

No return value

#### 7.29.1.2 AK_filesort_test()

```
TestResult AK_filesort_test ( )
```

#### 7.29.1.3 AK_get_header_number()

```
int AK_get_header_number (
            AK_block * iBlock,
            char * attribute_name )
```

Function that returns the 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]...

#### 7.29.1.4 AK_get_num_of_tuples()

```
int AK_get_num_of_tuples (
            AK_block * iBlock )
```

Function that returns tuples number in block.

**Author**

Unknown

**Returns**

tuples number in block

#### 7.29.1.5 AK_get_total_headers()

```
int AK_get_total_headers (
            AK_block * iBlock )
```

Function that returns the total number of headers in the block.

**Author**

Unknown

**Returns**

number of attribute in header (0 - MAX_ATTRIBUTES). USE in tuple_dict[num]...

#### 7.29.1.6 AK_reset_block()

```
void AK_reset_block (
            AK_block * block )
```

Function that resets block.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *block* | block to be resetted |

**Returns**

No return value

#### 7.29.1.7 AK_sort_segment()

```
int AK_sort_segment (
            char * srcTable,
            char * destTable,
            struct list_node * attributes )
```

Function that sorts a segment.

**Author**

Tomislav Bobinac, updated by Filip Žmuk

**Todo** Make it to suport multiple sort atributes and ASC|DESC ordering

**Returns**

No return value.

## 7.30 file/filesort.h File Reference

```
#include "../auxi/test.h"
#include "../mm/memoman.h"
#include "table.h"
#include "files.h"
#include "fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for filesort.h: This graph shows which files directly or indirectly include this file:

### Macros

- #define DATA_ROW_SIZE 200

  *Constatnt declaring size of data to be compared.*
- #define DATA_TUPLE_SIZE 500

  *Constant declaring size of data to be copied.*

### Functions

- int AK_get_total_headers (AK_block *iBlock)

  *Function that returns the total number of headers in the block.*
- int AK_get_header_number (AK_block *iBlock, char *attribute_name)

  *Function that returns the number of header in the block which to sort.*
- int AK_get_num_of_tuples (AK_block *iBlock)

  *Function that returns tuples number in block.*
- int AK_sort_segment (char *srcTable, char *destTable, struct list_node *attributes)

  *Function that sorts a segment.*
- void AK_reset_block (AK_block *block)

  *Function that resets block.*
- void AK_block_sort (AK_block *iBlock, char *atr_name)

  *Function that sorts the given block.*
- TestResult AK_filesort_test ()

### 7.30.1 Detailed Description

Header file that provides functions and defines for file sorting

## 7.30.2 Macro Definition Documentation

### 7.30.2.1 DATA_ROW_SIZE

```
#define DATA_ROW_SIZE 200
```

Constatnt declaring size of data to be compared.

### 7.30.2.2 DATA_TUPLE_SIZE

```
#define DATA_TUPLE_SIZE 500
```

Constant declaring size of data to be copied.

## 7.30.3 Function Documentation

### 7.30.3.1 AK_block_sort()

```
void AK_block_sort (
            AK_block * iBlock,
            char * attribute_name )
```

Function that sorts the given block.

**Author**

Bakoš Nikola

**Version**

v1.0

**Parameters**

| | |
|---|---|
| *iBlock* | block to be sorted |

**Returns**

No return value

#### 7.30.3.2 AK_filesort_test()

```
TestResult AK_filesort_test ( )
```

#### 7.30.3.3 AK_get_header_number()

```
int AK_get_header_number (
            AK_block * iBlock,
            char * attribute_name )
```

Function that returns the 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]...

#### 7.30.3.4 AK_get_num_of_tuples()

```
int AK_get_num_of_tuples (
            AK_block * iBlock )
```

Function that returns tuples number in block.

**Author**

Unknown

**Returns**

tuples number in block

#### 7.30.3.5 AK_get_total_headers()

```
int AK_get_total_headers (
            AK_block * iBlock )
```

Function that returns the total number of headers in the block.

**Author**

Unknown

**Returns**

number of attribute in header (0 - MAX_ATTRIBUTES). USE in tuple_dict[num]...

#### 7.30.3.6 AK_reset_block()

```
void AK_reset_block (
            AK_block * block )
```

Function that resets block.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *block* | block to be resetted |

**Returns**

No return value

#### 7.30.3.7 AK_sort_segment()

```
int AK_sort_segment (
            char * srcTable,
            char * destTable,
            struct list_node * attributes )
```

Function that sorts a segment.

**Author**

Tomislav Bobinac, updated by Filip Žmuk

**Todo** Make it to suport multiple sort atributes and ASC|DESC ordering

**Returns**

No return value.

**Author**

Tomislav Bobinac, updated by Filip Žmuk

**Todo** Make it to suport multiple sort atributes and ASC|DESC ordering

**Returns**

No return value.

## 7.31 file/id.c File Reference

```
#include "id.h"
```
Include dependency graph for id.c:

### Functions

- int AK_get_id ()

  *Function that fetches unique ID for any object, stored in a sequence.*
- char AK_get_table_id (char *tableName)

  *Function that fetches unique ID for any object, stored in sequence based on table name.*
- TestResult AK_id_test ()

  *Function for testing getting ID's.*

### 7.31.1 Detailed Description

Provides functions for creating id of objects

### 7.31.2 Function Documentation

#### 7.31.2.1 AK_get_id()

```
int AK_get_id ( )
```

Function that fetches unique ID for any object, stored in a sequence.

**Author**

Saša Vukšić, updated by Mislav Čakarić, changed by Mario Peroković, now uses AK_update_row, updated by Nenad Makar

**Returns**

objectID

#### 7.31.2.2 AK_get_table_id()

```
char AK_get_table_id (
            char * tableName )
```

Function that fetches unique ID for any object, stored in sequence based on table name.

**Author**

Lovro Predovan

**Returns**

objectID in string(char) format

#### 7.31.2.3 AK_id_test()

```
TestResult AK_id_test ( )
```

Function for testing getting ID's.

**Author**

Mislav Čakarić, updated by Nenad Makar

**Returns**

No retun value

## 7.32 file/id.h File Reference

```
#include "../auxi/test.h"
#include "table.h"
#include "fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for id.h: This graph shows which files directly or indirectly include this file:

### Macros

- #define ID_START_VALUE 100

  *Constant declaring start value of id.*

### Functions

- int AK_get_id ()

  *Function that fetches unique ID for any object, stored in a sequence.*
- TestResult AK_id_test ()

  *Function for testing getting ID's.*

### 7.32.1 Detailed Description

Provides functions and defines for creating id of objects

### 7.32.2 Macro Definition Documentation

#### 7.32.2.1 ID_START_VALUE

```
#define ID_START_VALUE 100
```

Constant declaring start value of id.

### 7.32.3 Function Documentation

#### 7.32.3.1 AK_get_id()

```
int AK_get_id ( )
```

Function that fetches unique ID for any object, stored in a sequence.

**Author**

Saša Vukšić, updated by Mislav Čakarić, changed by Mario Peroković, now uses AK_update_row, updated by Nenad Makar

**Returns**

objectID

#### 7.32.3.2 AK_id_test()

```
TestResult AK_id_test ( )
```

Function for testing getting ID's.

**Author**

Mislav Čakarić, updated by Nenad Makar

**Returns**

No retun value

# 7.33 file/idx/bitmap.c File Reference

```
#include "bitmap.h"
#include "../../auxi/iniparser.h"
```
Include dependency graph for bitmap.c:

## Functions

- int AK_If_ExistOp (struct list_node *L, char *ele)

  *Function that examines whether list L contains operator ele.*
- void AK_create_Index_Table (char *tblName, struct list_node *attributes)

  *Function that 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.*
- void AK_create_Index (char *tblName, char *tblNameIndex, char *attributeName, int positionTbl, int numAtributes, AK_header *headerIndex)

  *Function that loads index table with the value of particulary atribute.*
- list_ad * AK_get_attribute (char *indexName, char *attribute)

  *Function that gets adresses of the particuliar 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. That data is put in a list called add_root.*
- void AK_print_Att_Test (list_ad *list)

  *Function that prints the list of adresses.*
- list_ad * AK_get_Attribute (char *tableName, char *attributeName, char *attributeValue)

  *Function that fetches the values from the bitmap index if there is one for a 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 that updates the index, only on values that alredy 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.*
- void AK_add_to_bitmap_index (char *tableName, char *attributeName)

  *Function that writes the new value in block when index is updated.*
- void AK_print_Header_Test (char *tblName)

  *Function that tests printing header of table.*
- void AK_delete_bitmap_index (char *indexName)

  *Function that deletes bitmap index based on the name of index.*
- TestResult 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.*

### 7.33.1 Detailed Description

Provides functions for bitmap indexes

### 7.33.2 Function Documentation

#### 7.33.2.1 AK_add_to_bitmap_index()

```
void AK_add_to_bitmap_index (
             char * tableName,
             char * attributeName )
```

Function that writes the new value in block when index is updated.

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected.

**Author**

Saša Vukšić

**Parameters**

| | |
|---|---|
| *block* | block to write on |

**Returns**

EXIT_SUCESS when write operation is successful, otherwise EXIT_ERROR

**Author**

Lovro Predovan

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected

**Parameters**

| | |
|---|---|
| *tableName* | name of table |
| *attributeName* | name of attribute |
| *newAttributeValue* | new value of updated attribute |

**Returns**

No return value

#### 7.33.2.2 AK_bitmap_test()

```
TestResult 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

#### 7.33.2.3 AK_create_Index()

```
void AK_create_Index (
            char * tblName,
            char * tblNameIndex,
            char * attributeName,
            int positionTbl,
            int numAtributes,
            AK_header * headerIndex )
```

Function that loads index table with the value of particulary atribute.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

| | |
|---|---|
| *tblName* | source table |
| *tblNameIndex* | new name of index table |
| *attributeName* | attribute on which we make index |
| *positionTbl* | position of attribute in header of table |
| *numAtributes* | number of attributes in table |
| *headerIndex* | header of index table |

**Returns**

No return value

#### 7.33.2.4 AK_create_Index_Table()

```
void AK_create_Index_Table (
            char * tblName,
            struct list_node * attributes )
```

Function that 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**

| | |
|---|---|
| *tblName* | name of table |
| *attributes* | list of attributes on which we will create indexes |

**Returns**

No return value

#### 7.33.2.5 AK_delete_bitmap_index()

```
void AK_delete_bitmap_index (
            char * indexName )
```

Function that deletes bitmap index based on the name of index.

**Author**

Lovro Predovan

**Parameters**

| | |
|---|---|
| *Bitmap* | index table name |

**Returns**

No return value

#### 7.33.2.6 AK_get_attribute()

```
list_ad* AK_get_attribute (
            char * indexName,
            char * attribute )
```

Function that gets adresses of the particuliar 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. That data is put in a list called add_root.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *attribute* | name of attribute |

**Returns**

list of adresses

#### 7.33.2.7 AK_get_Attribute()

```
list_ad* AK_get_Attribute (
             char * tableName,
             char * attributeName,
             char * attributeValue )
```

Function that fetches the values from the bitmap index if there is one for a given table. It should be started when we are making selection on the table with bitmap index.

**Author**

Saša Vukšić

**Parameters**

| | |
|---|---|
| *tableName* | name of table |
| *attributeValue* | value of attribute |

**Returns**

list of adresses

#### 7.33.2.8 AK_If_ExistOp()

```
int AK_If_ExistOp (
             struct list_node * L,
             char * ele )
```

Function that examines whether list L contains operator ele.

**Author**

Saša Vukšić

**Parameters**

| | |
|---|---|
| *L* | list of elements |
| *ele* | operator to be found in list |

**Returns**

1 if operator ele is found in list, otherwise 0

#### 7.33.2.9 AK_print_Att_Test()

```
void AK_print_Att_Test (
            list_ad * list )
```

Function that prints the list of adresses.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

| | |
|---|---|
| *list* | list of adresses |

**Returns**

No return value

#### 7.33.2.10 AK_print_Header_Test()

```
void AK_print_Header_Test (
            char * tblName )
```

Function that tests printing header of table.

**Author**

Saša Vukšić

**Parameters**

| | |
|---|---|
| *tblName* | name of table who's header we are printing |

**Returns**

No return value

#### 7.33.2.11 AK_update()

```
void AK_update (
            int addBlock,
            int addTd,
            char * tableName,
            char * attributeName,
            char * attributeValue,
            char * newAttributeValue )
```

Function that updates the index, only on values that alredy 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**

| | |
|---|---|
| *addBlock* | adress of block |
| *addTD* | adress of tuple dict |
| *tableName* | name of table |
| *attributeName* | name of attribute |
| *attributeValue* | value of atribute |
| *newAttributeValue* | new value of updated attribute |

**Returns**

No return value

## 7.34 file/idx/bitmap.h File Reference

```
#include "../../auxi/test.h"
#include "../../mm/memoman.h"
#include "index.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../file/files.h"
#include "../../auxi/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 that examines whether list L contains operator ele.*
- void AK_create_Index_Table (char *tblName, struct list_node *attributes)

  *Function that 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.*
- 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 num↩ Atributes, AK_header *headerIndex)

  *Function that loads index table with the value of particulary atribute.*
- list_ad * AK_get_attribute (char *indexName, char *attribute)

  *Function that gets adresses of the particuliar 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. That data is put in a list called add_root.*
- void AK_create_List_Address_Test ()
- void AK_print_Att_Test (list_ad *list)

  *Function that prints the list of adresses.*
- list_ad * AK_get_Attribute (char *tableName, char *attributeName, char *attributeValue)

  *Function that fetches the values from the bitmap index if there is one for a 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 that updates the index, only on values that alredy 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 that writes the new value in block when index is updated.*
- TestResult 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 the name of index.*
- void AK_add_to_bitmap_index (char *tableName, char *attributeName)

  *Function that updates the index. Function deletes and recreates the index values again if different number of params is detected.*

## 7.34.1 Detailed Description

Header file that declares functions

## 7.34.2 Function Documentation

### 7.34.2.1 AK_add_to_bitmap_index()

```
void AK_add_to_bitmap_index (
            char * tableName,
            char * attributeName )
```

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected.

**Author**

Lovro Predovan

**Parameters**

| | |
|---|---|
| *tableName* | name of table |
| *attributeName* | name of attribute |
| *newAttributeValue* | new value of updated attribute |

**Returns**

No return value

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected.

**Author**

Saša Vukšić

**Parameters**

| | |
|---|---|
| *block* | block to write on |

**Returns**

EXIT_SUCESS when write operation is successful, otherwise EXIT_ERROR

**Author**

Lovro Predovan

Function that updates the index. Function deletes and recreates the index values again if different number of params is detected

**Parameters**

| | |
|---|---|
| *tableName* | name of table |
| *attributeName* | name of attribute |
| *newAttributeValue* | new value of updated attribute |

**Returns**

No return value

#### 7.34.2.2 AK_bitmap_test()

```
TestResult 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

#### 7.34.2.3 AK_create_Index()

```
void AK_create_Index (
            char * tblName,
            char * tblNameIndex,
            char * attributeName,
            int positionTbl,
            int numAtributes,
            AK_header * headerIndex )
```

Function that loads index table with the value of particulary atribute.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

| | |
|---|---|
| *tblName* | source table |
| *tblNameIndex* | new name of index table |
| *attributeName* | attribute on which we make index |
| *positionTbl* | position of attribute in header of table |
| *numAtributes* | number of attributes in table |
| *headerIndex* | header of index table |

**Returns**

No return value

#### 7.34.2.4 AK_create_Index_Table()

```
void AK_create_Index_Table (
            char * tblName,
            struct list_node * attributes )
```

Function that 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**

| | |
|---|---|
| *tblName* | name of table |
| *attributes* | list of attributes on which we will create indexes |

**Returns**

No return value

#### 7.34.2.5 AK_create_List_Address_Test()

```
void AK_create_List_Address_Test ( )
```

#### 7.34.2.6 AK_delete_bitmap_index()

```
void AK_delete_bitmap_index (
            char * indexName )
```

Function that deletes bitmap index based on the name of index.

**Author**

Lovro Predovan

**Parameters**

| | |
|---|---|
| *Bitmap* | index table name |

**Returns**

No return value

#### 7.34.2.7 AK_get_attribute()

```
list_ad* AK_get_attribute (
            char * indexName,
            char * attribute )
```

Function that gets adresses of the particuliar 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. That data is put in a list called add_root.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *attribute* | name of attribute |

**Returns**

list of adresses

#### 7.34.2.8 AK_get_Attribute()

```
list_ad* AK_get_Attribute (
              char * tableName,
              char * attributeName,
              char * attributeValue )
```

Function that fetches the values from the bitmap index if there is one for a given table. It should be started when we are making selection on the table with bitmap index.

**Author**

Saša Vukšić

**Parameters**

| | |
|---|---|
| *tableName* | name of table |
| *attributeValue* | value of attribute |

**Returns**

list of adresses

#### 7.34.2.9 AK_If_ExistOp()

```
int AK_If_ExistOp (
              struct list_node * L,
              char * ele )
```

Function that examines whether list L contains operator ele.

**Author**

Saša Vukšić

**Parameters**

| | |
|---|---|
| *L* | list of elements |
| *ele* | operator to be found in list |

**Returns**

1 if operator ele is found in list, otherwise 0

#### 7.34.2.10 AK_print_Att_Test()

```
void AK_print_Att_Test (
            list_ad * list )
```

Function that prints the list of adresses.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

| | |
|---|---|
| *list* | list of adresses |

**Returns**

No return value

#### 7.34.2.11 AK_print_Header_Test()

```
void AK_print_Header_Test (
            char * tblName )
```

Function that tests printing header of table.

**Author**

Saša Vukšić

**Parameters**

| | |
|---|---|
| *tblName* | name of table who's header we are printing |

**Returns**

No return value

#### 7.34.2.12 AK_update()

```
void AK_update (
            int addBlock,
            int addTd,
            char * tableName,
            char * attributeName,
            char * attributeValue,
            char * newAttributeValue )
```

Function that updates the index, only on values that alredy 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**

| | |
|---|---|
| *addBlock* | adress of block |
| *addTD* | adress of tuple dict |
| *tableName* | name of table |
| *attributeName* | name of attribute |
| *attributeValue* | value of atribute |
| *newAttributeValue* | new value of updated attribute |

**Returns**

No return value

#### 7.34.2.13 AK_write_block()

```
int AK_write_block (
            AK_block * block )
```

Function that writes the new value in block when index is updated.

**Author**

Saša Vukšić

**Parameters**

| | |
|---|---|
| *block* | block to write on |

**Returns**

EXIT_SUCESS when write operation is successful, otherwise EXIT_ERROR

Function that writes the new value in block when index is updated.

**Author**

Markus Schatten, updated by Domagoj Šitum (thread-safe enabled)

**Parameters**

| | |
|---|---|
| *block* | poiner to block allocated in memory to write |

**Returns**

EXIT_SUCCESS if successful, EXIT_ERROR otherwise

## 7.35 file/idx/btree.c File Reference

`#include "btree.h"`
Include dependency graph for btree.c:

### Functions

- AK_block * AK_btree_create (char *tblName, struct list_node *attributes, char *indexName)

  *Function that creates new btree index on integer attribute in table.*
- int AK_btree_delete (char *indexName)

  *Function that deletes index.*
- int AK_btree_search_delete (char *indexName, int *searchValue, int *endRange, int *toDo, AK_block *inputBlock)

  *Function that searches or deletes a value in btree index.*
- void btree_delete (btree_node *temp, AK_block *block, int idNext, int i)

  *Function that deletes a value in btree index.*
- int AK_btree_insert (char *indexName, int *insertValue, int *insertTd, int *insertBlock, AK_block *inputBlock)

  *Function that inserts a value in btree index.*
- btree_node * makevalues (btree_node *temp_help, int insertValue, int insertTd, int insertBlock, int i)

  *Function that sets values for node.*
- int findCorrectNumber (int number)

  *returns data about a leaf*
- btree_node * searchValue (int inserted, int insertValue, btree_node *temp, btree_node *temp_help, int *insertTd, int *insertBlock, int *increase, int number)

  *Function that sets values for node.*

- btree_node * setNodePointers (btree_node *temp, btree_node *temp_help, int pointerIndex, int secondValue, int firstPointer, int secondPointer)

  *Function that sets values for node.*
- btree_node * findPointers (btree_node *temp_node_one, btree_node *temp, int id, int *nodeInserted, int *nodeIncrease, int number, int pointerIndex)

  *Function that sets values for node.*
- btree_node * findValues (btree_node *temp_node_one, AK_block *block, int *helpAddress, int *helpType, btree_node *value_help)

  *Function that sets values for node.*
- TestResult AK_btree_test ()

  *Returns the amount of successful and failed tests.*

## 7.35.1 Detailed Description

Header file that provides functions for BTree indices

## 7.35.2 Function Documentation

### 7.35.2.1 AK_btree_create()

```
AK_block* AK_btree_create (
            char * tblName,
            struct list_node * attributes,
            char * indexName )
```

Function that creates new btree index on integer attribute in table.

**Author**

Anđelko Spevec

**Parameters**

| | |
|---|---|
| *tblName* | - name of the table on which we are creating index |
| *attributes* | - attribute on which we are creating index |
| *indexName* | - name of the index |

### 7.35.2.2 AK_btree_delete()

```
int AK_btree_delete (
            char * indexName )
```

Function that deletes index.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *indexName* | - name of the index+ |

#### 7.35.2.3 AK_btree_insert()

```
int AK_btree_insert (
             char * indexName,
             int * insertValue,
             int * insertTd,
             int * insertBlock,
             AK_block * inputBlock )
```

Function that inserts a value in btree index.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *indexName* | - name of the index |
| *insertValue* | - value for insert |
| *insertTd* | - index table destination |
| *insertBlock* | - block address |
| *inputBlock* | - block containing btree |
| *EXIT_SUCCESS* | if successful |

#### 7.35.2.4 AK_btree_search_delete()

```
int AK_btree_search_delete (
             char * indexName,
             int * searchValue,
             int * endRange,
             int * toDo,
             AK_block * inputBlock )
```

Function that searches or deletes a value in btree index.

**Author**

Anđelko Spevec

**Parameters**

| | |
|---|---|
| *indexName* | - name of the index |
| *searchValue* | - value that we are searching in the index |
| *endRange* | - if 0 search is for 0 value, else searching in range |
| *toDo* | - if 0 we just search else we delete the element if we find it |

#### 7.35.2.5 AK_btree_test()

```
TestResult AK_btree_test ( )
```

Returns the amount of successful and failed tests.

**Author**

unknown

**Returns**

TestResult

#### 7.35.2.6 btree_delete()

```
void btree_delete (
            btree_node * temp,
            AK_block * block,
            int idNext,
            int i )
```

Function that deletes a value in btree index.

**Author**

Anđelko Spevec

**Parameters**

| | |
|---|---|
| *temp* | - node for deletion |
| *block* | - block that contains binary tree |
| *idNext* | - index of the node that is to be deleted |

#### 7.35.2.7 findCorrectNumber()

```
int findCorrectNumber (
            int number )
```

returns data about a leaf

**Author**

unknown

**Returns**

required value

#### 7.35.2.8 findPointers()

```
btree_node* findPointers (
            btree_node * temp_node_one,
            btree_node * temp,
            int id,
            int * nodeInserted,
            int * nodeIncrease,
            int number,
            int pointerIndex )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *temp_node_one* | - node that has it's values set |
| *temp* | - node with data about existing node |
| *id* | - value to which a pointer of a node is to be set |
| *nodeInserted* | - determins if a node has value |
| *nodeIncrease* | - shows node that is currently watched |
| *number* | - determins the way node values are checked |
| *pointerIndex* | - indicates what pointer is used |

**Returns**

node that has it's values set

#### 7.35.2.9 findValues()

```
btree_node* findValues (
             btree_node * temp_node_one,
             AK_block * block,
             int * helpAddress,
             int * helpType,
             btree_node * value_help )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *temp_node_one* | - node that has it's values set |
| *block* | - block containing btree |
| *helpAddress* | - address of current node |
| *helpType* | - type of current node |
| *value_help* | - node in helpAddress |

**Returns**

node that has it's values set

#### 7.35.2.10 makevalues()

```
btree_node* makevalues (
             btree_node * temp_help,
             int insertValue,
             int insertTd,
             int insertBlock,
             int i )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *temp_help* | - node that has it's values set |
| *insertValue* | - value for insert |
| *insertTd* | - index table destination |
| *insertBlock* | - block address |
| *i* | - determins the index of element of node |

**Returns**

node that has it's values set

#### 7.35.2.11 searchValue()

```
btree_node* searchValue (
            int inserted,
            int insertValue,
            btree_node * temp,
            btree_node * temp_help,
            int * insertTd,
            int * insertBlock,
            int * increase,
            int number )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *inserted* | - determins if a value in a tree smaller then the value for insert has been found |
| *insertValue* | - value for insert |
| *temp* | - node with data about existing node |
| *temp_help* | - node that has it's values set |
| *insertTd* | - index table destination |
| *insertBlock* | - block address |
| *increase* | - determins the index of element of a node |
| *number* | - gives information about the number of elements in a leaf |

**Returns**

node that has it's values set

#### 7.35.2.12 setNodePointers()

```
btree_node* setNodePointers (
            btree_node * temp,
            btree_node * temp_help,
            int pointerIndex,
            int secondValue,
            int firstPointer,
            int secondPointer )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *temp* | - node with data about existing node |
| *temp_help* | - node that has it's values set |
| *pointerIndex* | - indicates what pointer is used |
| *secondValue* | - value of a node |
| *firstPointer* | - value to which a pointer of a node is to be set |
| *secondPointer* | - value to which a pointer of a node is to be set |

**Returns**

node that has it's values set

## 7.36 file/idx/btree.h File Reference

```
#include "../../auxi/test.h"
#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

- AK_block * AK_btree_create (char *tblName, struct list_node *attributes, char *indexName)

  *Function that creates new btree index on integer attribute in table.*
- int AK_btree_delete (char *indexName)

  *Function that deletes index.*
- btree_node * makevalues (btree_node *temp_help, int insertValue, int insertTd, int insertBlock, int i)

  *Function that sets values for node.*

- btree_node * searchValue (int inserted, int insertValue, btree_node *temp, btree_node *temp_help, int *insertTd, int *insertBlock, int *increase, int number)

  *Function that sets values for node.*
- btree_node * setNodePointers (btree_node *temp, btree_node *temp_help, int pointerIndex, int secondValue, int firstPointer, int secondPointer)

  *Function that sets values for node.*
- int findCorrectNumber (int number)

  *returns data about a leaf*
- btree_node * findPointers (btree_node *temp_node_one, btree_node *temp, int id, int *nodeInserted, int *nodeIncrease, int number, int pointerIndex)

  *Function that sets values for node.*
- btree_node * findValues (btree_node *temp_node_one, AK_block *block, int *helpAddress, int *helpType, btree_node *value_help)

  *Function that sets values for node.*
- void btree_delete (btree_node *temp, AK_block *block, int idNext, int i)

  *Function that deletes a value in btree index.*
- int AK_btree_search_delete (char *indexName, int *searchValue, int *endRange, int *toDo, AK_block *inputBlock)

  *Function that searches or deletes a value in btree index.*
- int AK_btree_insert (char *indexName, int *insertValue, int *insertTd, int *insertBlock, AK_block *inputBlock)

  *Function that inserts a value in btree index.*
- TestResult AK_btree_test ()

  *Returns the amount of successful and failed tests.*

## 7.36.1 Detailed Description

Header file that provides data strucures, functions and defines for BTree indices

## 7.36.2 Macro Definition Documentation

### 7.36.2.1 B

```
#define B 3
```

### 7.36.2.2 LEAF

```
#define LEAF 0
```

### 7.36.2.3 NODE

```
#define NODE 1
```

#### 7.36.2.4 ORDER

```
#define ORDER 6
```

### 7.36.3 Function Documentation

#### 7.36.3.1 AK_btree_create()

```
AK_block* AK_btree_create (
             char * tblName,
             struct list_node * attributes,
             char * indexName )
```

Function that creates new btree index on integer attribute in table.

**Author**

Anđelko Spevec

**Parameters**

| | |
|---|---|
| *tblName* | - name of the table on which we are creating index |
| *attributes* | - attribute on which we are creating index |
| *indexName* | - name of the index |

#### 7.36.3.2 AK_btree_delete()

```
int AK_btree_delete (
             char * indexName )
```

Function that deletes index.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *indexName* | - name of the index+ |

#### 7.36.3.3 AK_btree_insert()

```
int AK_btree_insert (
              char * indexName,
              int * insertValue,
              int * insertTd,
              int * insertBlock,
              AK_block * inputBlock )
```

Function that inserts a value in btree index.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *indexName* | - name of the index |
| *insertValue* | - value for insert |
| *insertTd* | - index table destination |
| *insertBlock* | - block address |
| *inputBlock* | - block containing btree |
| *EXIT_SUCCESS* | if successful |

#### 7.36.3.4 AK_btree_search_delete()

```
int AK_btree_search_delete (
              char * indexName,
              int * searchValue,
              int * endRange,
              int * toDo,
              AK_block * inputBlock )
```

Function that searches or deletes a value in btree index.

**Author**

Anđelko Spevec

**Parameters**

| | |
|---|---|
| *indexName* | - name of the index |
| *searchValue* | - value that we are searching in the index |
| *endRange* | - if 0 search is for 0 value, else searching in range |
| *toDo* | - if 0 we just search else we delete the element if we find it |

#### 7.36.3.5 AK_btree_test()

```
TestResult AK_btree_test ( )
```

Returns the amount of successful and failed tests.

**Author**

unknown

**Returns**

TestResult

#### 7.36.3.6 btree_delete()

```
void btree_delete (
            btree_node * temp,
            AK_block * block,
            int idNext,
            int i )
```

Function that deletes a value in btree index.

**Author**

Anđelko Spevec

**Parameters**

| | |
|---|---|
| *temp* | - node for deletion |
| *block* | - block that contains binary tree |
| *idNext* | - index of the node that is to be deleted |

#### 7.36.3.7 findCorrectNumber()

```
int findCorrectNumber (
            int number )
```

returns data about a leaf

**Author**

unknown

**Returns**

required value

#### 7.36.3.8 findPointers()

```
btree_node* findPointers (
            btree_node * temp_node_one,
            btree_node * temp,
            int id,
            int * nodeInserted,
            int * nodeIncrease,
            int number,
            int pointerIndex )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *temp_node_one* | - node that has it's values set |
| *temp* | - node with data about existing node |
| *id* | - value to which a pointer of a node is to be set |
| *nodeInserted* | - determins if a node has value |
| *nodeIncrease* | - shows node that is currently watched |
| *number* | - determins the way node values are checked |
| *pointerIndex* | - indicates what pointer is used |

**Returns**

node that has it's values set

#### 7.36.3.9 findValues()

```
btree_node* findValues (
            btree_node * temp_node_one,
            AK_block * block,
            int * helpAddress,
            int * helpType,
            btree_node * value_help )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *temp_node_one* | - node that has it's values set |
| *block* | - block containing btree |
| *helpAddress* | - address of current node |
| *helpType* | - type of current node |
| *value_help* | - node in helpAddress |

**Returns**

node that has it's values set

#### 7.36.3.10 makevalues()

```
btree_node* makevalues (
            btree_node * temp_help,
            int insertValue,
            int insertTd,
            int insertBlock,
            int i )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *temp_help* | - node that has it's values set |
| *insertValue* | - value for insert |
| *insertTd* | - index table destination |
| *insertBlock* | - block address |
| *i* | - determins the index of element of node |

**Returns**

node that has it's values set

#### 7.36.3.11 searchValue()

```
btree_node* searchValue (
            int inserted,
            int insertValue,
            btree_node * temp,
            btree_node * temp_help,
            int * insertTd,
            int * insertBlock,
            int * increase,
            int number )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *inserted* | - determins if a value in a tree smaller then the value for insert has been found |
| *insertValue* | - value for insert |
| *temp* | - node with data about existing node |
| *temp_help* | - node that has it's values set |
| *insertTd* | - index table destination |
| *insertBlock* | - block address |
| *increase* | - determins the index of element of a node |
| *number* | - gives information about the number of elements in a leaf |

**Returns**

node that has it's values set

#### 7.36.3.12 setNodePointers()

```
btree_node* setNodePointers (
             btree_node * temp,
             btree_node * temp_help,
             int pointerIndex,
             int secondValue,
             int firstPointer,
             int secondPointer )
```

Function that sets values for node.

**Author**

unknown

**Parameters**

| | |
|---|---|
| *temp* | - node with data about existing node |
| *temp_help* | - node that has it's values set |
| *pointerIndex* | - indicates what pointer is used |
| *secondValue* | - value of a node |
| *firstPointer* | - value to which a pointer of a node is to be set |
| *secondPointer* | - value to which a pointer of a node is to be set |

**Returns**

node that has it's values set

## 7.37 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 that computes a hash value from varchar or integer.*
- struct_add * AK_insert_bucket_to_block (char *indexName, char *data, int type)

  *Function that inserts a bucket to block.*
- void AK_update_bucket_in_block (struct_add *add, char *data)

  *Function that updates a bucket in block.*
- void AK_change_hash_info (char *indexName, int modulo, int main_bucket_num, int hash_bucket_num)

  *Function that changes a info of hash index.*
- hash_info * AK_get_hash_info (char *indexName)

  *Function that fetches the info for hash index.*
- struct_add * AK_get_nth_main_bucket_add (char *indexName, int n)

  *Function that fetches nth main bucket.*
- void AK_insert_in_hash_index (char *indexName, int hashValue, struct_add *add)

  *Function that inserts a record in hash bucket.*
- struct_add * AK_find_delete_in_hash_index (char *indexName, struct list_node *values, int delete)

  *Function that fetches or deletes a record from hash index.*
- struct_add * AK_find_in_hash_index (char *indexName, struct list_node *values)

  *Function that fetches a record from the hash index.*
- void AK_delete_in_hash_index (char *indexName, struct list_node *values)

  *Function that deletes a record from the hash index.*
- int AK_create_hash_index (char *tblName, struct list_node *attributes, char *indexName)

  *Function that creates a hash index.*
- void AK_delete_hash_index (char *indexName)
- TestResult AK_hash_test ()

  *Function that tests hash index.*

### 7.37.1 Detailed Description

Provides functions for Hash indices

### 7.37.2 Function Documentation

#### 7.37.2.1 AK_change_hash_info()

```
void AK_change_hash_info (
            char * indexName,
            int modulo,
            int main_bucket_num,
            int hash_bucket_num )
```

Function that changes a info of hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *modulo* | value for modulo hash function |
| *main_bucket_num* | number of main buckets |
| *hash_bucket_num* | number of hash buckets |

**Returns**

No return value

### 7.37.2.2 AK_create_hash_index()

```
int AK_create_hash_index (
            char * tblName,
            struct list_node * attributes,
            char * indexName )
```

Function that creates a hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *tblName* | name of table for which the index is being created |
| *indexName* | name of index |
| *attributes* | list of attributes over which the index is being created |

**Returns**

success or error

### 7.37.2.3 AK_delete_hash_index()

```
void AK_delete_hash_index (
            char * indexName )
```

#### 7.37.2.4 AK_delete_in_hash_index()

```
void AK_delete_in_hash_index (
            char * indexName,
            struct list_node * values )
```

Function that deletes a record from the hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *values* | list of values (one row) to search in hash index |

**Returns**

No return value

#### 7.37.2.5 AK_elem_hash_value()

```
int AK_elem_hash_value (
            struct list_node * elem )
```

Function that computes a hash value from varchar or integer.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *elem* | element of row for wich value is to be computed |

**Returns**

hash value

#### 7.37.2.6 AK_find_delete_in_hash_index()

```
struct_add* AK_find_delete_in_hash_index (
            char * indexName,
            struct list_node * values,
            int delete )
```

Function that fetches or deletes a record from hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *values* | list of values (one row) to search in hash index |
| *delete* | 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

#### 7.37.2.7 AK_find_in_hash_index()

```
struct_add* AK_find_in_hash_index (
             char * indexName,
             struct list_node * values )
```

Function that fetches a record from the hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *values* | list of values (one row) to search in hash index |

**Returns**

address structure with data where the record is in table

#### 7.37.2.8 AK_get_hash_info()

```
hash_info* AK_get_hash_info (
             char * indexName )
```

Function that fetches the info for hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |

**Returns**

info bucket with info data for hash index

#### 7.37.2.9 AK_get_nth_main_bucket_add()

```
struct_add* AK_get_nth_main_bucket_add (
            char * indexName,
            int n )
```

Function that fetches nth main bucket.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *n* | number of main bucket |

**Returns**

address structure with data where the bucket is stored

#### 7.37.2.10 AK_hash_test()

```
TestResult AK_hash_test ( )
```

Function that tests hash index.

**Author**

Mislav Čakarić

**Returns**

No return value

#### 7.37.2.11 AK_insert_bucket_to_block()

```
struct_add* AK_insert_bucket_to_block (
            char * indexName,
            char * data,
            int type )
```

Function that inserts a bucket to block.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *data* | content of bucket stored in char array |
| *type* | type of bucket (MAIN_BUCKET or HASH_BUCKET) |

**Returns**

address structure with data where the bucket is stored

#### 7.37.2.12 AK_insert_in_hash_index()

```
void AK_insert_in_hash_index (
            char * indexName,
            int hashValue,
            struct_add * add )
```

Function that inserts a record in hash bucket.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *hashValue* | hash value of record that is being inserted |
| *add* | address structure with data where the hash bucket is stored |

**Returns**

No return value

#### 7.37.2.13 AK_update_bucket_in_block()

```
void AK_update_bucket_in_block (
            struct_add * add,
            char * data )
```

Function that updates a bucket in block.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *add* | address of where the bucket is stored |
| *data* | content of bucket stored in char array |

**Returns**

No return value

## 7.38 file/idx/hash.h File Reference

```
#include "../../auxi/test.h"
#include "index.h"
#include "../../file/table.h"
#include "../../auxi/constants.h"
#include "../../auxi/configuration.h"
#include "../files.h"
#include "../../auxi/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 that computes a hash value from varchar or integer.*
- struct_add * AK_insert_bucket_to_block (char *indexName, char *data, int type)

  *Function that inserts a bucket to block.*
- void AK_update_bucket_in_block (struct_add *add, char *data)

  *Function that updates a bucket in block.*
- void AK_change_hash_info (char *indexName, int modulo, int main_bucket_num, int hash_bucket_num)

  *Function that changes a info of hash index.*
- hash_info * AK_get_hash_info (char *indexName)

  *Function that fetches the info for hash index.*
- struct_add * AK_get_nth_main_bucket_add (char *indexName, int n)

  *Function that fetches nth main bucket.*
- void AK_insert_in_hash_index (char *indexName, int hashValue, struct_add *add)

  *Function that inserts a record in hash bucket.*
- struct_add * AK_find_delete_in_hash_index (char *indexName, struct list_node *values, int delete)

  *Function that fetches or deletes a record from hash index.*
- struct_add * AK_find_in_hash_index (char *indexName, struct list_node *values)

  *Function that fetches a record from the hash index.*
- void AK_delete_in_hash_index (char *indexName, struct list_node *values)

  *Function that deletes a record from the hash index.*
- int AK_create_hash_index (char *tblName, struct list_node *attributes, char *indexName)

  *Function that creates a hash index.*
- void AK_delete_hash_index (char *indexName)
- TestResult AK_hash_test ()

  *Function that tests hash index.*

### 7.38.1 Detailed Description

Header file that provides data structures, functions and defines for Hash indices

### 7.38.2 Function Documentation

#### 7.38.2.1 AK_change_hash_info()

```
void AK_change_hash_info (
            char * indexName,
            int modulo,
            int main_bucket_num,
            int hash_bucket_num )
```

Function that changes a info of hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *modulo* | value for modulo hash function |
| *main_bucket_num* | number of main buckets |
| *hash_bucket_num* | number of hash buckets |

**Returns**

No return value

#### 7.38.2.2 AK_create_hash_index()

```
int AK_create_hash_index (
            char * tblName,
            struct list_node * attributes,
            char * indexName )
```

Function that creates a hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *tblName* | name of table for which the index is being created |
| *indexName* | name of index |
| *attributes* | list of attributes over which the index is being created |

**Returns**

success or error

#### 7.38.2.3 AK_delete_hash_index()

```
void AK_delete_hash_index (
            char * indexName )
```

#### 7.38.2.4 AK_delete_in_hash_index()

```
void AK_delete_in_hash_index (
            char * indexName,
            struct list_node * values )
```

Function that deletes a record from the hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *values* | list of values (one row) to search in hash index |

**Returns**

No return value

#### 7.38.2.5 AK_elem_hash_value()

```
int AK_elem_hash_value (
            struct list_node * elem )
```

Function that computes a hash value from varchar or integer.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *elem* | element of row for wich value is to be computed |

**Returns**

hash value

#### 7.38.2.6 AK_find_delete_in_hash_index()

```
struct_add* AK_find_delete_in_hash_index (
            char * indexName,
            struct list_node * values,
            int delete )
```

Function that fetches or deletes a record from hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *values* | list of values (one row) to search in hash index |
| *delete* | 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

#### 7.38.2.7 AK_find_in_hash_index()

```
struct_add* AK_find_in_hash_index (
            char * indexName,
            struct list_node * values )
```

Function that fetches a record from the hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *values* | list of values (one row) to search in hash index |

**Returns**

address structure with data where the record is in table

#### 7.38.2.8 AK_get_hash_info()

```
hash_info* AK_get_hash_info (
            char * indexName )
```

Function that fetches the info for hash index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |

**Returns**

info bucket with info data for hash index

#### 7.38.2.9 AK_get_nth_main_bucket_add()

```
struct_add* AK_get_nth_main_bucket_add (
            char * indexName,
            int n )
```

Function that fetches nth main bucket.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *n* | number of main bucket |

**Returns**

address structure with data where the bucket is stored

#### 7.38.2.10 AK_hash_test()

```
TestResult AK_hash_test ( )
```

Function that tests hash index.

**Author**

Mislav Čakarić

**Returns**

No return value

#### 7.38.2.11 AK_insert_bucket_to_block()

```
struct_add* AK_insert_bucket_to_block (
            char * indexName,
            char * data,
            int type )
```

Function that inserts a bucket to block.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *data* | content of bucket stored in char array |
| *type* | type of bucket (MAIN_BUCKET or HASH_BUCKET) |

**Returns**

address structure with data where the bucket is stored

#### 7.38.2.12 AK_insert_in_hash_index()

```
void AK_insert_in_hash_index (
            char * indexName,
            int hashValue,
            struct_add * add )
```

Function that inserts a record in hash bucket.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *indexName* | name of index |
| *hashValue* | hash value of record that is being inserted |
| *add* | address structure with data where the hash bucket is stored |

**Returns**

No return value

#### 7.38.2.13 AK_update_bucket_in_block()

```
void AK_update_bucket_in_block (
            struct_add * add,
            char * data )
```

Function that updates a bucket in block.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *add* | address of where the bucket is stored |
| *data* | content of bucket stored in char array |

**Returns**

No return value

## 7.39 file/idx/index.c File Reference

```
#include "index.h"
#include <stdlib.h>
#include "../../auxi/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 that initialises a linked list.*
- element_ad AK_Get_First_elementAd (list_ad *L)

  *Function that finds the first node of linked list.*
- element_ad AK_Get_Last_elementAd (list_ad *L)

  *Function that finds the last node of linked list.*
- element_ad AK_Get_Next_elementAd (element_ad Currentelement_op)

  *Function that finds the next node of a node in linked list.*
- element_ad AK_Get_Previous_elementAd (element_ad Currentelement_op, element_ad L)

  *Function that finds the previous node of a node in linked list.*
- int AK_Get_Position_Of_elementAd (element_ad Searchedelement_op, list_ad *L)

  *Function that finds the position of a node in linked list.*
- void AK_Delete_elementAd (element_ad Deletedelement_op, list_ad *L)

  *Function that deletes a node from a linked list.*
- void AK_Delete_All_elementsAd (list_ad *L)

  *Function that deletes all nodes in a linked list.*

- void AK_Insert_NewelementAd (int addBlock, int indexTd, char ∗attName, element_ad elementBefore)

  *Function that inserts a new element into a linked list.*
- int AK_num_index_attr (char ∗indexTblName)

  *Function that fetches the number of elements in a 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 that 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 that prints out the index table.*
- void AK_index_test ()

  *Test funtion for index structures(list) and printing table.*

## 7.39.1 Detailed Description

Provides functions for indexes

## 7.39.2 Function Documentation

### 7.39.2.1 AK_Delete_All_elementsAd()

```
void AK_Delete_All_elementsAd (
            list_ad * L )
```

Function that deletes all nodes in a linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *L* | list head |

**Returns**

No return value

#### 7.39.2.2 AK_Delete_elementAd()

```
void AK_Delete_elementAd (
             element_ad Deletedelement_op,
             list_ad * L )
```

Function that deletes a node from a linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *Deletedelement_op* | - address of node to delete |
| *list_ad* | *L - list head |

**Returns**

No return value

#### 7.39.2.3 AK_Get_First_elementAd()

```
element_ad AK_Get_First_elementAd (
             list_ad * L )
```

Function that finds the first node of linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *list_ad* | *L linked list head |

**Returns**

Address of first node

#### 7.39.2.4 AK_get_index_header()

```
AK_header* AK_get_index_header (
             char * indexTblName )
```

Function that gets 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**

| | |
|---|---|
| ∗*tblName* | table name |

**Returns**

array of table header

#### 7.39.2.5 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**

| | |
|---|---|
| ∗*tableName* | table name |

**Returns**

number of rows in the table

#### 7.39.2.6 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**

| | |
|---|---|
| *row* | zero-based row index |
| *column* | zero-based column index |
| *∗tblName* | table name |

**Returns**

value in the list

#### 7.39.2.7 AK_Get_Last_elementAd()

```
element_ad AK_Get_Last_elementAd (
            list_ad * L )
```

Function that finds the last node of linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *list_ad* | ∗L linked list head |

**Returns**

Address of last node or 0 if list is empty

#### 7.39.2.8 AK_Get_Next_elementAd()

```
element_ad AK_Get_Next_elementAd (
            element_ad Currentelement_op )
```

Function that finds the next node of a node in linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *Currentelement_op* | address of current node |

**Returns**

Address of next node or 0 if current node is last in list

#### 7.39.2.9 AK_Get_Position_Of_elementAd()

```
int AK_Get_Position_Of_elementAd (
            element_ad Searchedelement_op,
            list_ad * L )
```

Function that finds the position of a node in linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *Searchedelement_op* | address of current note |
| *∗L* | linked list head |

**Returns**

Integer value of current node's order in the list

#### 7.39.2.10 AK_Get_Previous_elementAd()

```
element_ad AK_Get_Previous_elementAd (
            element_ad Currentelement_op,
            element_ad L )
```

Function that finds the previous node of a node in linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *Currentelement_op* | Address of current node |
| *L* | previous element |

**Returns**

Address of previous node or 0 if the current node is the head or the list is empty

#### 7.39.2.11 AK_index_table_exist()

```
int AK_index_table_exist (
            char * indexTblName )
```

Function that 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**

| | |
|---|---|
| *tblName* | table name |

**Returns**

returns 1 if table exist or returns 0 if table does not exist

#### 7.39.2.12 AK_index_test()

```
void AK_index_test ( )
```

Test funtion for index structures(list) and printing table.

**Author**

Lovro Predovan

**Returns**

No return value

#### 7.39.2.13 AK_InitializelistAd()

```
void AK_InitializelistAd (
            list_ad * L )
```

Function that initialises a linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *list_ad* | ∗L linked list head |

**Returns**

No return value

#### 7.39.2.14 AK_Insert_NewelementAd()

```
void AK_Insert_NewelementAd (
            int addBlock,
            int indexTd,
            char * attName,
            element_ad elementBefore )
```

Function that inserts a new element into a linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *addBlock* | address block |
| *indexTd* | index table destination |
| ∗*attname* | attribute name |
| *elementBefore* | address of the node after which the new node will be inserted |

**Returns**

No return value

#### 7.39.2.15 AK_num_index_attr()

```
int AK_num_index_attr (
            char * indexTblName )
```

Function that fetches the number of elements in a index table.

**Author**

Lovro Predovan

**Parameters**

| | |
|---|---|
| *index* | table name |

**Returns**

No return value

#### 7.39.2.16 AK_print_index_table()

```
void AK_print_index_table (
            char * indexTblName )
```

Function that prints out the index table.

**Author**

Matija Šestak, modified for indexes by Lovro Predovan

**Parameters**

| | |
|---|---|
| ∗*tblName* | table name |

**Returns**

No return value

## 7.40 file/idx/index.h File Reference

```
#include "../../auxi/mempro.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../file/files.h"
```

Include dependency graph for index.h: This graph shows which files directly or indirectly include this file:

## Classes

- struct struct_add

  *Structure defining node address.*
- struct list_structure_ad

## Typedefs

- typedef struct list_structure_ad list_structure_ad
- typedef list_structure_ad ∗ element_ad
- typedef list_structure_ad list_ad

## Functions

- int AK_index_table_exist (char ∗indexTblName)

  *Function that examines whether there is a table with the name "tblName" in the system catalog (AK_relation)*
- void AK_print_index_table (char ∗indexTblName)

  *Function that prints out the 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 that fetches the number of elements in a index table.*
- void AK_InitializelistAd (list_ad ∗L)

  *Function that initialises a linked list.*
- element_ad AK_Get_First_elementAd (list_ad ∗L)

  *Function that finds the first node of linked list.*
- element_ad AK_Get_Last_elementAd (list_ad ∗L)

  *Function that finds the last node of linked list.*
- element_ad AK_Get_Next_elementAd (element_ad Currentelement_op)

  *Function that finds the next node of a node in linked list.*
- element_ad AK_Get_Previous_elementAd (element_ad Currentelement_op, element_ad L)

  *Function that finds the previous node of a node in linked list.*
- int AK_Get_Position_Of_elementAd (element_ad Searchedelement_op, list_ad ∗L)

  *Function that finds the position of a node in linked list.*
- void AK_Delete_elementAd (element_ad Deletedelement_op, list_ad ∗L)

  *Function that deletes a node from a linked list.*
- void AK_Delete_All_elementsAd (list_ad ∗L)

  *Function that deletes all nodes in a linked list.*
- void AK_Insert_NewelementAd (int addBlock, int indexTd, char ∗attName, element_ad elementBefore)

  *Function that inserts a new element into a linked list.*
- void AK_index_test ()

  *Test funtion for index structures(list) and printing table.*

### 7.40.1 Detailed Description

Header file that provides data structures, functions and defines for bitmap index

## 7.40.2 Typedef Documentation

### 7.40.2.1 element_ad

```
typedef list_structure_ad* element_ad
```

### 7.40.2.2 list_ad

```
typedef list_structure_ad list_ad
```

### 7.40.2.3 list_structure_ad

```
typedef struct list_structure_ad list_structure_ad
```

## 7.40.3 Function Documentation

### 7.40.3.1 AK_Delete_All_elementsAd()

```
void AK_Delete_All_elementsAd (
            list_ad * L )
```

Function that deletes all nodes in a linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *L* | list head |

**Returns**

No return value

#### 7.40.3.2 AK_Delete_elementAd()

```
void AK_Delete_elementAd (
            element_ad Deletedelement_op,
            list_ad * L )
```

Function that deletes a node from a linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *Deletedelement_op* | - address of node to delete |
| *list_ad* | *L - list head |

**Returns**

No return value

#### 7.40.3.3 AK_Get_First_elementAd()

```
element_ad AK_Get_First_elementAd (
            list_ad * L )
```

Function that finds the first node of linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *list_ad* | *L linked list head |

**Returns**

Address of first node

#### 7.40.3.4 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**

| | |
|---|---|
| ∗*tableName* | table name |

**Returns**

number of rows in the table

#### 7.40.3.5 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**

| | |
|---|---|
| *row* | zero-based row index |
| *column* | zero-based column index |
| ∗*tblName* | table name |

**Returns**

value in the list

#### 7.40.3.6 AK_Get_Last_elementAd()

```
element_ad AK_Get_Last_elementAd (
            list_ad * L )
```

Function that finds the last node of linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *list_ad* | ∗L linked list head |

**Returns**

Address of last node or 0 if list is empty

#### 7.40.3.7 AK_Get_Next_elementAd()

```
element_ad AK_Get_Next_elementAd (
            element_ad Currentelement_op )
```

Function that finds the next node of a node in linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *Currentelement_op* | address of current node |

**Returns**

Address of next node or 0 if current node is last in list

#### 7.40.3.8 AK_Get_Position_Of_elementAd()

```
int AK_Get_Position_Of_elementAd (
            element_ad Searchedelement_op,
            list_ad * L )
```

Function that finds the position of a node in linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *Searchedelement_op* | address of current note |
| *∗L* | linked list head |

**Returns**

Integer value of current node's order in the list

#### 7.40.3.9 AK_Get_Previous_elementAd()

```
element_ad AK_Get_Previous_elementAd (
            element_ad Currentelement_op,
            element_ad L )
```

Function that finds the previous node of a node in linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *Currentelement_op* | Address of current node |
| *L* | previous element |

**Returns**

Address of previous node or 0 if the current node is the head or the list is empty

#### 7.40.3.10 AK_index_table_exist()

```
int AK_index_table_exist (
            char * indexTblName )
```

Function that 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**

| | |
|---|---|
| *tblName* | table name |

**Returns**

returns 1 if table exist or returns 0 if table does not exist

#### 7.40.3.11 AK_index_test()

```
void AK_index_test ( )
```

Test funtion for index structures(list) and printing table.

**Author**

Lovro Predovan

**Returns**

No return value

#### 7.40.3.12 AK_InitializelistAd()

```
void AK_InitializelistAd (
            list_ad * L )
```

Function that initialises a linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *list_ad* | *L linked list head |

**Returns**

No return value

#### 7.40.3.13 AK_Insert_NewelementAd()

```
void AK_Insert_NewelementAd (
             int addBlock,
             int indexTd,
             char * attName,
             element_ad elementBefore )
```

Function that inserts a new element into a linked list.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *addBlock* | address block |
| *indexTd* | index table destination |
| ∗*attname* | attribute name |
| *elementBefore* | address of the node after which the new node will be inserted |

**Returns**

No return value

#### 7.40.3.14 AK_num_index_attr()

```
int AK_num_index_attr (
             char * indexTblName )
```

Function that fetches the number of elements in a index table.

**Author**

Lovro Predovan

**Parameters**

| | |
|---|---|
| *index* | table name |

**Returns**

No return value

#### 7.40.3.15 AK_print_index_table()

```
void AK_print_index_table (
            char * indexTblName )
```

Function that prints out the index table.

**Author**

Matija Šestak, modified for indexes by Lovro Predovan

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |

**Returns**

No return value

## 7.41 file/sequence.c File Reference

```
#include "sequence.h"
```
Include dependency graph for sequence.c:

### 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 that returns the current value of the sequence.*
- int AK_sequence_next_value (char ∗name)
  *Function that 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 that fetches sequence id.*
- int AK_sequence_rename (char ∗old_name, char ∗new_name)
  *Function that 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 a sequence.*
- TestResult AK_sequence_test ()
  *Function used for sequences testing.*

### 7.41.1 Detailed Description

Provides functions for sequences

## 7.41.2 Function Documentation

### 7.41.2.1 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**

| | |
|---|---|
| *name* | name of the sequence |
| *start_value* | start value of the sequence |
| *increment* | increment of the sequence |
| *max_value* | maximium value of the sequence |
| *min_value* | minimum value of the sequence |
| *cycle* | 0:non-cyclic sequence, 1:cyclic sequence |

**Returns**

sequence_id or EXIT_ERROR

### 7.41.2.2 AK_sequence_current_value()

```
int AK_sequence_current_value (
            char * name )
```

Function that returns the current value of the sequence.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *name* | name of the sequence |

**Returns**

current_value or EXIT_ERROR

#### 7.41.2.3 AK_sequence_get_id()

```
int AK_sequence_get_id (
             char * name )
```

Function that fetches sequence id.

**Author**

Ljubo Barać

**Parameters**

| | |
|---|---|
| *name* | Name of the sequence |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.41.2.4 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 a sequence.

**Author**

Boris Kišić fixed by Ljubo Barać

**Parameters**

| | |
|---|---|
| *name* | Name of the sequence |
| *start_value* | start value of the sequence |
| *increment* | increment of the sequence |
| *max_value* | maximium value of the sequence |
| *min_value* | minimum value of the sequence |
| *cycle* | 0:non-cyclic sequence, 1:cyclic sequence |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.41.2.5 AK_sequence_next_value()

```
int AK_sequence_next_value (
            char * name )
```

Function that returns the next value of the sequence and writes it in a system table as current value.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *name* | name of the sequence |

**Returns**

next_value or EXIT_ERROR

#### 7.41.2.6 AK_sequence_remove()

```
int AK_sequence_remove (
            char * name )
```

Function for removing sequence.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *name* | name of the sequence |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.41.2.7 AK_sequence_rename()

```
int AK_sequence_rename (
            char * old_name,
            char * new_name )
```

Function that renames the sequence.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *old_name* | Name of the sequence to be renamed |
| *new_name* | New name of the sequence |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.41.2.8 AK_sequence_test()

```
TestResult AK_sequence_test ( )
```

Function used for sequences testing.

**Author**

Boris Kišić fixed by Ljubo Barać

**Returns**

No return value

## 7.42 file/sequence.h File Reference

```
#include "../auxi/test.h"
#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 that returns the current value of the sequence.*
- int AK_sequence_next_value (char *name)

  *Function that 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 that 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 a sequence.*
- int AK_sequence_get_id (char *name)

  *Function that fetches sequence id.*
- TestResult AK_sequence_test ()

  *Function used for sequences testing.*

### 7.42.1 Detailed Description

Header file that provides functions and defines for sequences

### 7.42.2 Function Documentation

#### 7.42.2.1 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**

| | |
|---|---|
| *name* | name of the sequence |
| *start_value* | start value of the sequence |
| *increment* | increment of the sequence |
| *max_value* | maximium value of the sequence |
| *min_value* | minimum value of the sequence |
| *cycle* | 0:non-cyclic sequence, 1:cyclic sequence |

**Returns**

sequence_id or EXIT_ERROR

#### 7.42.2.2 AK_sequence_current_value()

```
int AK_sequence_current_value (
            char * name )
```

Function that returns the current value of the sequence.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *name* | name of the sequence |

**Returns**

current_value or EXIT_ERROR

#### 7.42.2.3 AK_sequence_get_id()

```
int AK_sequence_get_id (
            char * name )
```

Function that fetches sequence id.

**Author**

Ljubo Barać

**Parameters**

| | |
|---|---|
| *name* | Name of the sequence |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.42.2.4 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 a sequence.

**Author**

Boris Kišić fixed by Ljubo Barać

**Parameters**

| | |
|---|---|
| *name* | Name of the sequence |
| *start_value* | start value of the sequence |
| *increment* | increment of the sequence |
| *max_value* | maximium value of the sequence |
| *min_value* | minimum value of the sequence |
| *cycle* | 0:non-cyclic sequence, 1:cyclic sequence |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.42.2.5 AK_sequence_next_value()

```
int AK_sequence_next_value (
             char * name )
```

Function that returns the next value of the sequence and writes it in a system table as current value.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *name* | name of the sequence |

**Returns**

next_value or EXIT_ERROR

#### 7.42.2.6 AK_sequence_remove()

```
int AK_sequence_remove (
            char * name )
```

Function for removing sequence.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *name* | name of the sequence |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.42.2.7 AK_sequence_rename()

```
int AK_sequence_rename (
            char * old_name,
            char * new_name )
```

Function that renames the sequence.

/**

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *old_name* | Name of the sequence to be renamed |
| *new_name* | New name of the sequence |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *old_name* | Name of the sequence to be renamed |
| *new_name* | New name of the sequence |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.42.2.8 AK_sequence_test()

```
TestResult AK_sequence_test ( )
```

Function used for sequences testing.

**Author**

Boris Kišić fixed by Ljubo Barać

**Returns**

No return value

## 7.43 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)

  *Temporary function that creates table, and inserts an entry to the system_relation catalog.*
- void AK_temp_create_table (char *table, AK_header *header, int type_segment)

  *Temporary function that creates table, and inserts an entry to the system_relation catalog.*
- int AK_num_attr (char *tblName)

  *Functions that determines the number of attributes in the table.*
- int AK_get_num_records (char *tblName)

  *Function that determines the number of rows in the table.*
- AK_header * AK_get_header (char *tblName)

  *Function that fetches the table header.*
- char * AK_get_attr_name (char *tblName, int index)

  *Function that fetches attribute name for some zero-based index.*
- int AK_get_attr_index (char *tblName, char *attrName)

  *Function that fetches zero-based index for atrribute.*

- struct list_node * AK_get_column (int num, char *tblName)

  *Function that fetches all values in some column and put on the list.*
- struct list_node * AK_get_row (int num, char *tblName)

  *Function that fetches all values in some row and put on the list.*
- struct list_node * AK_get_tuple (int row, int column, char *tblName)

  *Function that fetches a 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 that 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 the value of an attribute from the row.*
- void AK_print_row_to_file (int col_len[ ], struct list_node *row)

  *Function that prints the table row update by Luka Rajcevic.*
- void AK_print_table_to_file (char *tblName)

  *Function that prints a table.*
- int AK_table_empty (char *tblName)

  *Function that checks whether the table is empty.*
- int AK_get_table_obj_id (char *table)

  *Function that fetches an 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 that checks 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)*
- TestResult AK_table_test ()

  *Function for testing table abstraction.*
- TestResult AK_op_rename_test ()

  *Function for renaming operator testing (moved from rename.c)*

### 7.43.1 Detailed Description

Provides functions for table abstraction

### 7.43.2 Function Documentation

#### 7.43.2.1 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 that checks 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**

| | |
|---|---|
| *tbl1_temp_block* | first cache block of the first table |
| *tbl2_temp_block* | first cache block of the second table |
| *operator_name* | the name of operator, used for displaying error message |

**Returns**

if success returns num of attributes in schema, else returns EXIT_ERROR

#### 7.43.2.2 AK_create_create_table_parameter()

```
AK_create_table_parameter* AK_create_create_table_parameter (
            int type,
            char * name )
```

#### 7.43.2.3 AK_create_table()

```
void AK_create_table (
            char * tblName,
            AK_create_table_parameter * parameters,
            int attribute_count )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *table* | table name |
| *header* | AK_header of the new table |
| *type_segment* | type of the new segment |

**Returns**

No return value

#### 7.43.2.4 AK_get_attr_index()

```
int AK_get_attr_index (
            char * tblName,
            char * attrName )
```

Function that fetches zero-based index for atrribute.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |
| *∗attrName* | attribute name |

**Returns**

zero-based index

#### 7.43.2.5 AK_get_attr_name()

```
char* AK_get_attr_name (
            char * tblName,
            int index )
```

Function that fetches attribute name for some zero-based index.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |
| *index* | zero-based index |

**Returns**

attribute name

#### 7.43.2.6 AK_get_column()

```
struct list_node* AK_get_column (
            int num,
            char * tblName )
```

Function that fetches all values in some column and put on the list.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *num* | zero-based column index |
| **tblName* | table name |

**Returns**

column values list

#### 7.43.2.7 AK_get_header()

```
AK_header* AK_get_header (
            char * tblName )
```

Function that fetches the 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**

| | |
|---|---|
| **tblName* | table name |

**Returns**

array of table header

#### 7.43.2.8 AK_get_num_records()

```
int AK_get_num_records (
            char * tblName )
```

Function that determines the 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**

| | |
|---|---|
| *tableName | table name |

**Returns**

number of rows in the table

#### 7.43.2.9 AK_get_row()

```
struct list_node* AK_get_row (
            int num,
            char * tblName )
```

Function that fetches all values in some row and put on the list.

**Author**

Markus Schatten, Matija Šestak.

**Parameters**

| | |
|---|---|
| *num* | zero-based row index |
| * | tblName table name |

**Returns**

row values list

#### 7.43.2.10 AK_get_table_obj_id()

```
int AK_get_table_obj_id (
            char * table )
```

Function that fetches an obj_id of named table from AK_relation system table.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *table | table name |

**Returns**

obj_id of the table or EXIT_ERROR if there is no table with that name

#### 7.43.2.11 AK_get_tuple()

```
struct list_node* AK_get_tuple (
            int row,
            int column,
            char * tblName )
```

Function that fetches a value in some row and column.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *row* | zero-based row index |
| *column* | zero-based column index |
| *tblName | table name |

**Returns**

value in the list

#### 7.43.2.12 AK_num_attr()

```
int AK_num_attr (
            char * tblName )
```

Functions that determines 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

#### 7.43.2.13 AK_op_rename_test()

```
TestResult AK_op_rename_test ( )
```

Function for renaming operator testing (moved from rename.c)

**Author**

Mislav Čakarić, edited by Ljubo Barać

**Returns**

No return value

#### 7.43.2.14 AK_print_row()

```
void AK_print_row (
            int col_len[],
            struct list_node * row )
```

Function that prints table row.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *col_len[]* | array of max lengths for each attribute |
| *∗row* | list with row elements |

**Returns**

No return value

#### 7.43.2.15 AK_print_row_spacer()

```
void AK_print_row_spacer (
            int col_len[],
            int length )
```

Function that prints row spacer.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *col_len[]* | max lengths for each attribute cell |
| *length* | total table width |

**Returns**

printed row spacer

#### 7.43.2.16 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**

| | |
|---|---|
| *col_len[]* | max lengths for each attribute cell |
| *length* | total table width |

**Returns**

printed row spacer

#### 7.43.2.17 AK_print_row_to_file()

```
void AK_print_row_to_file (
            int col_len[],
            struct list_node * row )
```

Function that prints the table row update by Luka Rajcevic.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *col_len[]* | array of max lengths for each attribute |
| *∗row* | list with row elements |

**Returns**

No return value

#### 7.43.2.18 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**

| | |
|---|---|
| *∗tblName* | table name |

**Returns**

No return value

#### 7.43.2.19 AK_print_table_to_file()

```
void AK_print_table_to_file (
            char * tblName )
```

Function that prints a table.

**Author**

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one) update by Luka Rajcevic

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |

**Returns**

No return value update by Anto Tomaš (corrected the AK_DeleteAll_L3 function)

#### 7.43.2.20 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**

| | |
|---|---|
| *old_table_name* | old name of the table |
| *new_table_name* | new name of the table |
| *old_attr* | name of the attribute to rename |
| *new_attr* | new name for the attribute to rename |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.43.2.21 AK_table_empty()

```
int AK_table_empty (
            char * tblName )
```

Function that checks whether the table is empty.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *tblName | table name |

**Returns**

true/false

#### 7.43.2.22 AK_table_exist()

```
int AK_table_exist (
            char * tblName )
```

Function that examines whether there is a table with the name "tblName" in the system catalog (AK_relation)

**Author**

Jurica Hlevnjak

**Parameters**

| | |
|---|---|
| *tblName* | table name |

**Returns**

returns 1 if table exist or returns 0 if table does not exist

#### 7.43.2.23 AK_table_test()

```
TestResult AK_table_test ( )
```

Function for testing table abstraction.

**Author**

Unknown

**Returns**

No return value

@update by Ana-Marija Balen - added getRow function to the test

#### 7.43.2.24 AK_temp_create_table()

```
void AK_temp_create_table (
            char * table,
            AK_header * header,
            int type_segment )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *table* | table name |
| *header* | AK_header of the new table |
| *type_segment* | type of the new segment |

**Returns**

No return value

#### 7.43.2.25 AK_tuple_to_string()

```
char* AK_tuple_to_string (
            struct list_node * tuple )
```

Function that converts tuple value to string.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tuple* | tuple in the list |

**Returns**

tuple value as a string

#### 7.43.2.26 get_row_attr_data()

```
char* get_row_attr_data (
            int column,
            struct list_node * node )
```

Function that returns the value of an attribute from the row.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *column* | index of column atribute |
| *∗row* | list with row elements |

**Returns**

atribute data

## 7.44 file/table.h File Reference

```
#include "../auxi/test.h"
#include "../mm/memoman.h"
#include "../auxi/mempro.h"
#include <time.h>
#include "../sql/drop.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../file/sequence.h"
#include "../auxi/constants.h"
#include "test.h"
#include "../dm/dbman.h"
```

Include dependency graph for table.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct AK_create_table_struct

## Macros

- #define TABLE

## 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)

  *Temporary function that creates table, and inserts an entry to the system_relation catalog.*
- void AK_temp_create_table (char *table, AK_header *header, int type_segment)

  *Temporary function that creates table, and inserts an entry to the system_relation catalog.*
- int AK_num_attr (char *tblName)

  *Functions that determines the number of attributes in the table.*
- int AK_get_num_records (char *tblName)

  *Function that determines the number of rows in the table.*
- AK_header * AK_get_header (char *tblName)

  *Function that fetches the table header.*
- char * AK_get_attr_name (char *tblName, int index)

  *Function that fetches attribute name for some zero-based index.*
- int AK_get_attr_index (char *tblName, char *attrName)

  *Function that fetches zero-based index for atrribute.*
- struct list_node * AK_get_column (int num, char *tblName)

  *Function that fetches all values in some column and put on the list.*
- struct list_node * AK_get_row (int num, char *tblName)

  *Function that fetches all values in some row and put on the list.*
- struct list_node * AK_get_tuple (int row, int column, char *tblName)

  *Function that fetches a 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 the table row update by Luka Rajcevic.*
- void AK_print_table_to_file (char *tblName)

  *Function that prints a table.*
- int AK_table_empty (char *tblName)

  *Function that checks whether the table is empty.*
- int AK_get_table_obj_id (char *table)

  *Function that fetches an 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 that checks if tables have the same relation schema.*
- char * get_row_attr_data (int column, struct list_node *node)

  *Function that returns the value of an attribute from the row.*
- TestResult 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)*
- TestResult AK_op_rename_test ()

  *Function for renaming operator testing (moved from rename.c)*

### 7.44.1 Detailed Description

Header file that provides data structures, functions and defines 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

### 7.44.2 Macro Definition Documentation

#### 7.44.2.1 TABLE

```
#define TABLE
```

### 7.44.3 Typedef Documentation

#### 7.44.3.1 AK_create_table_parameter

```
typedef struct AK_create_table_struct AK_create_table_parameter
```

### 7.44.4 Function Documentation

#### 7.44.4.1 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 that checks 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**

| | |
|---|---|
| *tbl1_temp_block* | first cache block of the first table |
| *tbl2_temp_block* | first cache block of the second table |
| *operator_name* | the name of operator, used for displaying error message |

**Returns**

if success returns num of attributes in schema, else returns EXIT_ERROR

#### 7.44.4.2 AK_create_create_table_parameter()

```
AK_create_table_parameter* AK_create_create_table_parameter (
            int type,
            char * name )
```

#### 7.44.4.3 AK_create_table()

```
void AK_create_table (
            char * tblName,
            AK_create_table_parameter * parameters,
            int attribute_count )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *table* | table name |
| *header* | AK_header of the new table |
| *type_segment* | type of the new segment |

**Returns**

No return value

#### 7.44.4.4 AK_get_attr_index()

```
int AK_get_attr_index (
            char * tblName,
            char * attrName )
```

Function that fetches zero-based index for atrribute.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |
| *∗attrName* | attribute name |

**Returns**

zero-based index

#### 7.44.4.5 AK_get_attr_name()

```
char* AK_get_attr_name (
            char * tblName,
            int index )
```

Function that fetches attribute name for some zero-based index.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |
| *index* | zero-based index |

**Returns**

attribute name

#### 7.44.4.6 AK_get_column()

```
struct list_node* AK_get_column (
            int num,
            char * tblName )
```

Function that fetches all values in some column and put on the list.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *num* | zero-based column index |
| **tblName* | table name |

**Returns**

column values list

#### 7.44.4.7 AK_get_header()

```
AK_header* AK_get_header (
            char * tblName )
```

Function that fetches the 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**

| | |
|---|---|
| **tblName* | table name |

**Returns**

array of table header

#### 7.44.4.8 AK_get_num_records()

```
int AK_get_num_records (
            char * tblName )
```

Function that determines the 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**

| | |
|---|---|
| *tableName | table name |

**Returns**

number of rows in the table

#### 7.44.4.9 AK_get_row()

```
struct list_node* AK_get_row (
            int num,
            char * tblName )
```

Function that fetches all values in some row and put on the list.

**Author**

Markus Schatten, Matija Šestak.

**Parameters**

| | |
|---|---|
| *num* | zero-based row index |
| * | tblName table name |

**Returns**

row values list

#### 7.44.4.10 AK_get_table_obj_id()

```
int AK_get_table_obj_id (
            char * table )
```

Function that fetches an obj_id of named table from AK_relation system table.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *table | table name |

**Returns**

obj_id of the table or EXIT_ERROR if there is no table with that name

#### 7.44.4.11 AK_get_tuple()

```
struct list_node* AK_get_tuple (
            int row,
            int column,
            char * tblName )
```

Function that fetches a value in some row and column.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *row* | zero-based row index |
| *column* | zero-based column index |
| *tblName | table name |

**Returns**

value in the list

#### 7.44.4.12 AK_num_attr()

```
int AK_num_attr (
            char * tblName )
```

Functions that determines 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

#### 7.44.4.13 AK_op_rename_test()

```
TestResult AK_op_rename_test ( )
```

Function for renaming operator testing (moved from rename.c)

**Author**

Mislav Čakarić, edited by Ljubo Barać

**Returns**

No return value

#### 7.44.4.14 AK_print_row()

```
void AK_print_row (
            int col_len[],
            struct list_node * row )
```

Function that prints table row.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *col_len[]* | array of max lengths for each attribute |
| *∗row* | list with row elements |

**Returns**

No return value

#### 7.44.4.15 AK_print_row_spacer()

```
void AK_print_row_spacer (
            int col_len[],
            int length )
```

Function that prints row spacer.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *col_len[]* | max lengths for each attribute cell |
| *length* | total table width |

**Returns**

printed row spacer

#### 7.44.4.16 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**

| | |
|---|---|
| *col_len[]* | max lengths for each attribute cell |
| *length* | total table width |

**Returns**

printed row spacer

#### 7.44.4.17 AK_print_row_to_file()

```
void AK_print_row_to_file (
            int col_len[],
            struct list_node * row )
```

Function that prints the table row update by Luka Rajcevic.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *col_len[]* | array of max lengths for each attribute |
| *∗row* | list with row elements |

**Returns**

No return value

#### 7.44.4.18 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**

| | |
|---|---|
| *∗tblName* | table name |

**Returns**

No return value

#### 7.44.4.19 AK_print_table_to_file()

```
void AK_print_table_to_file (
            char * tblName )
```

Function that prints a table.

**Author**

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one) update by Luka Rajcevic

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |

**Returns**

No return value update by Anto Tomaš (corrected the AK_DeleteAll_L3 function)

#### 7.44.4.20 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**

| | |
|---|---|
| *old_table_name* | old name of the table |
| *new_table_name* | new name of the table |
| *old_attr* | name of the attribute to rename |
| *new_attr* | new name for the attribute to rename |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.44.4.21 AK_table_empty()

```
int AK_table_empty (
            char * tblName )
```

Function that checks whether the table is empty.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *tblName | table name |

**Returns**

true/false

#### 7.44.4.22 AK_table_test()

```
TestResult AK_table_test ( )
```

Function for testing table abstraction.

**Author**

Unknown

**Returns**

No return value

@update by Ana-Marija Balen - added getRow function to the test

#### 7.44.4.23 AK_temp_create_table()

```
void AK_temp_create_table (
            char * table,
            AK_header * header,
            int type_segment )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *table* | table name |
| *header* | AK_header of the new table |
| *type_segment* | type of the new segment |

**Returns**

No return value

#### 7.44.4.24 AK_tuple_to_string()

```
char* AK_tuple_to_string (
            struct list_node * tuple )
```

Function that converts tuple value to string.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tuple* | tuple in the list |

**Returns**

tuple value as a string

#### 7.44.4.25 get_row_attr_data()

```
char* get_row_attr_data (
            int column,
            struct list_node * node )
```

Function that returns the value of an attribute from the row.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *column* | index of column atribute |
| *∗row* | list with row elements |

**Returns**

atribute data

## 7.45 file/tableOld.c File Reference

`#include "../file/table.h"`
Include dependency graph for tableOld.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)

  *Temporary function that creates table, and inserts an entry to the system_relation catalog.*
- void AK_temp_create_table (char *table, AK_header *header, int type_segment)

  *Temporary function that creates table, and inserts an entry to the system_relation catalog.*
- int AK_num_attr (char *tblName)

  *Functions that determines the number of attributes in the table.*
- int AK_get_num_records (char *tblName)

  *Function that determines the number of rows in the table.*
- AK_header * AK_get_header (char *tblName)

  *Function that fetches the table header.*
- char * AK_get_attr_name (char *tblName, int index)

  *Function that fetches attribute name for some zero-based index.*
- int AK_get_attr_index (char *tblName, char *attrName)

  *Function that fetches zero-based index for atrribute.*
- struct list_node * AK_get_column (int num, char *tblName)

  *Function that fetches all values in some column and put on the list.*
- struct list_node * AK_get_row (int num, char *tblName)

  *Function that fetches all values in some row and put on the list.*
- struct list_node * AK_get_tuple (int row, int column, char *tblName)

  *Function that fetches a 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 that 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 the value of an attribute from the row.*
- void AK_print_row_to_file (int col_len[ ], struct list_node *row)

  *Function that prints the table row update by Luka Rajcevic.*
- void AK_print_table_to_file (char *tblName)

*Function that prints a table.*

- int AK_table_empty (char *tblName)

  *Function that checks whether the table is empty.*

- int AK_get_table_obj_id (char *table)

  *Function that fetches an 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 that checks 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)*

- TestResult AK_table_test ()

  *Function for testing table abstraction.*

- TestResult AK_op_rename_test ()

  *Function for renaming operator testing (moved from rename.c)*

## 7.45.1 Function Documentation

### 7.45.1.1 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 that checks 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**

| | |
|---|---|
| *tbl1_temp_block* | first cache block of the first table |
| *tbl2_temp_block* | first cache block of the second table |
| *operator_name* | the name of operator, used for displaying error message |

**Returns**

if success returns num of attributes in schema, else returns EXIT_ERROR

### 7.45.1.2 AK_create_create_table_parameter()

```
AK_create_table_parameter* AK_create_create_table_parameter (
            int type,
            char * name )
```

#### 7.45.1.3 AK_create_table()

```
void AK_create_table (
            char * tblName,
            AK_create_table_parameter * parameters,
            int attribute_count )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *table* | table name |
| *header* | AK_header of the new table |
| *type_segment* | type of the new segment |

**Returns**

No return value

#### 7.45.1.4 AK_get_attr_index()

```
int AK_get_attr_index (
            char * tblName,
            char * attrName )
```

Function that fetches zero-based index for atrribute.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |
| *∗attrName* | attribute name |

**Returns**

zero-based index

#### 7.45.1.5 AK_get_attr_name()

```
char* AK_get_attr_name (
            char * tblName,
            int index )
```

Function that fetches attribute name for some zero-based index.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |
| *index* | zero-based index |

**Returns**

attribute name

#### 7.45.1.6 AK_get_column()

```
struct list_node* AK_get_column (
            int num,
            char * tblName )
```

Function that fetches all values in some column and put on the list.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *num* | zero-based column index |
| *∗tblName* | table name |

**Returns**

column values list

#### 7.45.1.7 AK_get_header()

```
AK_header* AK_get_header (
            char * tblName )
```

Function that fetches the 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**

| | |
|---|---|
| ∗*tblName* | table name |

**Returns**

array of table header

#### 7.45.1.8 AK_get_num_records()

```
int AK_get_num_records (
            char * tblName )
```

Function that determines the 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**

| | |
|---|---|
| ∗*tableName* | table name |

**Returns**

number of rows in the table

#### 7.45.1.9 AK_get_row()

```
struct list_node* AK_get_row (
            int num,
            char * tblName )
```

Function that fetches all values in some row and put on the list.

**Author**

Markus Schatten, Matija Šestak.

**Parameters**

| | |
|---|---|
| *num* | zero-based row index |
| * | tblName table name |

**Returns**

row values list

#### 7.45.1.10 AK_get_table_obj_id()

```
int AK_get_table_obj_id (
            char * table )
```

Function that fetches an obj_id of named table from AK_relation system table.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| **table* | table name |

**Returns**

obj_id of the table or EXIT_ERROR if there is no table with that name

#### 7.45.1.11 AK_get_tuple()

```
struct list_node* AK_get_tuple (
            int row,
            int column,
            char * tblName )
```

Function that fetches a value in some row and column.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *row* | zero-based row index |
| *column* | zero-based column index |
| ∗*tblName* | table name |

**Returns**

value in the list

#### 7.45.1.12 AK_num_attr()

```
int AK_num_attr (
            char * tblName )
```

Functions that determines 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

#### 7.45.1.13 AK_op_rename_test()

```
TestResult AK_op_rename_test ( )
```

Function for renaming operator testing (moved from rename.c)

**Author**

Mislav Čakarić, edited by Ljubo Barać

**Returns**

No return value

#### 7.45.1.14 AK_print_row()

```
void AK_print_row (
            int col_len[],
            struct list_node * row )
```

Function that prints table row.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *col_len[]* | array of max lengths for each attribute |
| ∗*row* | list with row elements |

**Returns**

No return value

#### 7.45.1.15 AK_print_row_spacer()

```
void AK_print_row_spacer (
            int col_len[],
            int length )
```

Function that prints row spacer.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *col_len[]* | max lengths for each attribute cell |
| *length* | total table width |

**Returns**

printed row spacer

#### 7.45.1.16 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**

| | |
|---|---|
| *col_len[]* | max lengths for each attribute cell |
| *length* | total table width |

**Returns**

printed row spacer

#### 7.45.1.17 AK_print_row_to_file()

```
void AK_print_row_to_file (
            int col_len[],
            struct list_node * row )
```

Function that prints the table row update by Luka Rajcevic.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *col_len[]* | array of max lengths for each attribute |
| **row* | list with row elements |

**Returns**

No return value

#### 7.45.1.18 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**

| | |
|---|---|
| *tblName | table name |

**Returns**

No return value

#### 7.45.1.19 AK_print_table_to_file()

```
void AK_print_table_to_file (
            char * tblName )
```

Function that prints a table.

**Author**

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one) update by Luka Rajcevic

**Parameters**

| | |
|---|---|
| *tblName | table name |

**Returns**

No return value update by Anto Tomaš (corrected the AK_DeleteAll_L3 function)

#### 7.45.1.20 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**

| | |
|---|---|
| *old_table_name* | old name of the table |
| *new_table_name* | new name of the table |
| *old_attr* | name of the attribute to rename |
| *new_attr* | new name for the attribute to rename |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.45.1.21 AK_table_empty()

```
int AK_table_empty (
            char * tblName )
```

Function that checks whether the table is empty.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| ∗*tblName* | table name |

**Returns**

true/false

#### 7.45.1.22 AK_table_exist()

```
int AK_table_exist (
            char * tblName )
```

Function that examines whether there is a table with the name "tblName" in the system catalog (AK_relation)

**Author**

Jurica Hlevnjak

**Parameters**

| | |
|---|---|
| *tblName* | table name |

**Returns**

returns 1 if table exist or returns 0 if table does not exist

### 7.45.1.23 AK_table_test()

```
TestResult AK_table_test ( )
```

Function for testing table abstraction.

**Author**

Unknown

**Returns**

No return value

@update by Ana-Marija Balen - added getRow function to the test

### 7.45.1.24 AK_temp_create_table()

```
void AK_temp_create_table (
            char * table,
            AK_header * header,
            int type_segment )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *table* | table name |
| *header* | AK_header of the new table |
| *type_segment* | type of the new segment |

**Returns**

No return value

#### 7.45.1.25 AK_tuple_to_string()

```
char* AK_tuple_to_string (
            struct list_node * tuple )
```

Function that converts tuple value to string.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tuple* | tuple in the list |

**Returns**

tuple value as a string

#### 7.45.1.26 get_row_attr_data()

```
char* get_row_attr_data (
            int column,
            struct list_node * node )
```

Function that returns the value of an attribute from the row.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *column* | index of column atribute |
| *∗row* | list with row elements |

**Returns**

atribute data

# 7.46 file/tableOld.h File Reference

```
#include "../auxi/test.h"
#include "../mm/memoman.h"
#include "../auxi/mempro.h"
#include <time.h>
```

Include dependency graph for tableOld.h:

## Classes

- struct AK_create_table_struct

## Macros

- #define TABLE

## 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)

  *Temporary function that creates table, and inserts an entry to the system_relation catalog.*
- void AK_temp_create_table (char *table, AK_header *header, int type_segment)

  *Temporary function that creates table, and inserts an entry to the system_relation catalog.*
- int AK_num_attr (char *tblName)

  *Functions that determines the number of attributes in the table.*
- int AK_get_num_records (char *tblName)

  *Function that determines the number of rows in the table.*
- AK_header * AK_get_header (char *tblName)

  *Function that fetches the table header.*
- char * AK_get_attr_name (char *tblName, int index)

  *Function that fetches attribute name for some zero-based index.*
- int AK_get_attr_index (char *tblName, char *attrName)

  *Function that fetches zero-based index for atrribute.*
- struct list_node * AK_get_column (int num, char *tblName)

  *Function that fetches all values in some column and put on the list.*
- struct list_node * AK_get_row (int num, char *tblName)

  *Function that fetches all values in some row and put on the list.*
- struct list_node * AK_get_tuple (int row, int column, char *tblName)

  *Function that fetches a 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 the table row update by Luka Rajcevic.*

- void AK_print_table_to_file (char *tblName)

  *Function that prints a table.*

- int AK_table_empty (char *tblName)

  *Function that checks whether the table is empty.*

- int AK_get_table_obj_id (char *table)

  *Function that fetches an 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 that checks if tables have the same relation schema.*

- char * get_row_attr_data (int column, struct list_node *node)

  *Function that returns the value of an attribute from the row.*

- TestResult 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)*

- TestResult AK_op_rename_test ()

  *Function for renaming operator testing (moved from rename.c)*

## 7.46.1 Macro Definition Documentation

### 7.46.1.1 TABLE

```
#define TABLE
```

## 7.46.2 Typedef Documentation

### 7.46.2.1 AK_create_table_parameter

```
typedef struct AK_create_table_struct AK_create_table_parameter
```

## 7.46.3 Function Documentation

#### 7.46.3.1 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 that checks 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**

| | |
|---|---|
| *tbl1_temp_block* | first cache block of the first table |
| *tbl2_temp_block* | first cache block of the second table |
| *operator_name* | the name of operator, used for displaying error message |

**Returns**

if success returns num of attributes in schema, else returns EXIT_ERROR

#### 7.46.3.2 AK_create_create_table_parameter()

```
AK_create_table_parameter* AK_create_create_table_parameter (
            int type,
            char * name )
```

#### 7.46.3.3 AK_create_table()

```
void AK_create_table (
            char * tblName,
            AK_create_table_parameter * parameters,
            int attribute_count )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *table* | table name |
| *header* | AK_header of the new table |
| *type_segment* | type of the new segment |

**Returns**

No return value

#### 7.46.3.4 AK_get_attr_index()

```
int AK_get_attr_index (
            char * tblName,
            char * attrName )
```

Function that fetches zero-based index for atrribute.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |
| *∗attrName* | attribute name |

**Returns**

zero-based index

#### 7.46.3.5 AK_get_attr_name()

```
char* AK_get_attr_name (
            char * tblName,
            int index )
```

Function that fetches attribute name for some zero-based index.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |
| *index* | zero-based index |

**Returns**

attribute name

#### 7.46.3.6 AK_get_column()

```
struct list_node* AK_get_column (
            int num,
            char * tblName )
```

Function that fetches all values in some column and put on the list.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *num* | zero-based column index |
| ∗*tblName* | table name |

**Returns**

column values list

#### 7.46.3.7 AK_get_header()

```
AK_header* AK_get_header (
            char * tblName )
```

Function that fetches the 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**

| | |
|---|---|
| ∗*tblName* | table name |

**Returns**

array of table header

#### 7.46.3.8 AK_get_num_records()

```
int AK_get_num_records (
            char * tblName )
```

Function that determines the 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**

| | |
|---|---|
| *tableName | table name |

**Returns**

number of rows in the table

#### 7.46.3.9 AK_get_row()

```
struct list_node* AK_get_row (
            int num,
            char * tblName )
```

Function that fetches all values in some row and put on the list.

**Author**

Markus Schatten, Matija Šestak.

**Parameters**

| | |
|---|---|
| *num* | zero-based row index |
| * | tblName table name |

**Returns**

row values list

#### 7.46.3.10 AK_get_table_obj_id()

```
int AK_get_table_obj_id (
            char * table )
```

Function that fetches an obj_id of named table from AK_relation system table.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *table | table name |

**Returns**

obj_id of the table or EXIT_ERROR if there is no table with that name

#### 7.46.3.11 AK_get_tuple()

```
struct list_node* AK_get_tuple (
            int row,
            int column,
            char * tblName )
```

Function that fetches a value in some row and column.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *row* | zero-based row index |
| *column* | zero-based column index |
| *tblName | table name |

**Returns**

value in the list

#### 7.46.3.12 AK_num_attr()

```
int AK_num_attr (
            char * tblName )
```

Functions that determines 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

#### 7.46.3.13 AK_op_rename_test()

```
TestResult AK_op_rename_test ( )
```

Function for renaming operator testing (moved from rename.c)

**Author**

Mislav Čakarić, edited by Ljubo Barać

**Returns**

No return value

#### 7.46.3.14 AK_print_row()

```
void AK_print_row (
            int col_len[],
            struct list_node * row )
```

Function that prints table row.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *col_len[]* | array of max lengths for each attribute |
| *∗row* | list with row elements |

**Returns**

No return value

#### 7.46.3.15 AK_print_row_spacer()

```
void AK_print_row_spacer (
            int col_len[],
            int length )
```

Function that prints row spacer.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *col_len[]* | max lengths for each attribute cell |
| *length* | total table width |

**Returns**

printed row spacer

#### 7.46.3.16 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**

| | |
|---|---|
| *col_len[]* | max lengths for each attribute cell |
| *length* | total table width |

**Returns**

printed row spacer

#### 7.46.3.17 AK_print_row_to_file()

```
void AK_print_row_to_file (
            int col_len[],
            struct list_node * row )
```

Function that prints the table row update by Luka Rajcevic.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *col_len[]* | array of max lengths for each attribute |
| *∗row* | list with row elements |

**Returns**

No return value

#### 7.46.3.18 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**

| | |
|---|---|
| *∗tblName* | table name |

**Returns**

No return value

#### 7.46.3.19 AK_print_table_to_file()

```
void AK_print_table_to_file (
            char * tblName )
```

Function that prints a table.

**Author**

Dino Laktašić and Mislav Čakarić (replaced old print table function by new one) update by Luka Rajcevic

**Parameters**

| | |
|---|---|
| *∗tblName* | table name |

**Returns**

No return value update by Anto Tomaš (corrected the AK_DeleteAll_L3 function)

#### 7.46.3.20 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**

| | |
|---|---|
| *old_table_name* | old name of the table |
| *new_table_name* | new name of the table |
| *old_attr* | name of the attribute to rename |
| *new_attr* | new name for the attribute to rename |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.46.3.21 AK_table_empty()

```
int AK_table_empty (
            char * tblName )
```

Function that checks whether the table is empty.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *tblName | table name |

**Returns**

true/false

#### 7.46.3.22 AK_table_test()

```
TestResult AK_table_test ( )
```

Function for testing table abstraction.

**Author**

Unknown

**Returns**

No return value

@update by Ana-Marija Balen - added getRow function to the test

#### 7.46.3.23 AK_temp_create_table()

```
void AK_temp_create_table (
            char * table,
            AK_header * header,
            int type_segment )
```

Temporary function that creates table, and inserts an entry to the system_relation catalog.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *table* | table name |
| *header* | AK_header of the new table |
| *type_segment* | type of the new segment |

**Returns**

No return value

#### 7.46.3.24 AK_tuple_to_string()

```
char* AK_tuple_to_string (
            struct list_node * tuple )
```

Function that converts tuple value to string.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗tuple* | tuple in the list |

**Returns**

tuple value as a string

#### 7.46.3.25 get_row_attr_data()

```
char* get_row_attr_data (
            int column,
            struct list_node * node )
```

Function that returns the value of an attribute from the row.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *column* | index of column atribute |
| *∗row* | list with row elements |

**Returns**

atribute data

## 7.47 mm/memoman.c File Reference

```
#include "memoman.h"
#include "../dm/dbman.h"
```

Include dependency graph for memoman.c:

### Functions

- int AK_cache_block (int num, AK_mem_block *mem_block)

  *Function that caches a block into the memory.*
- int AK_cache_AK_malloc ()

  *Function that initializes the global cache memory (variable db_cache)*
- int AK_redo_log_AK_malloc ()

  *Function that initializes the global redo log memory (variable redo_log)*
- int AK_find_available_result_block ()

  *Function that finds the available block for result caching in a circular array.*
- unsigned long AK_generate_result_id (unsigned char *str)

  *Function that generates a unique hash identifier for each cached result by using djb2 algorithm.*
- void AK_cache_result (char *srcTable, AK_block *temp_block, AK_header header[])

  *Function that caches the fetched result block in memory.*
- int AK_query_mem_AK_malloc ()

  *Function that initializes the global query memory (variable query_mem)*
- void AK_query_mem_AK_free ()

  *Function that releases the global query memory (variable query_mem)*
- int AK_memoman_init ()

  *Function that initializes the memory manager (cache, redo log and query memory)*
- AK_mem_block * AK_get_block (int num)

  *Function that reads a block from the 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.*
- int AK_release_oldest_cache_block ()

  *Functions that flushes the oldest block to disk and recalculates the next block to remove.*
- int AK_mem_block_modify (AK_mem_block *mem_block, int dirty)

  *Function that modifies the "dirty" bit of a block, and update the timestamps accordingly.*
- int AK_refresh_cache ()

  *Function that re-reads all the blocks from the disk.*
- table_addresses * AK_get_index_segment_addresses (char *segmentName)

  *Function for getting a index segment address.*
- table_addresses * AK_get_segment_addresses (char *segmentName)

  *Function for getting a relation segment address.*
- table_addresses * AK_get_segment_addresses_internal (char *tableName, char *segmentName)

  *Function for getting addresses of some table.*
- int AK_get_system_table_address (const char *name)

  *Function that gets the address of a system table by name.*
- 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 that finds AK_free space in some block betwen 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.*

- TestResult AK_memoman_test ()
- TestResult AK_memoman_test2 ()

## 7.47.1 Detailed Description

Defines functions for the memory manager of Kalashnikov DB

## 7.47.2 Function Documentation

### 7.47.2.1 AK_cache_AK_malloc()

```
int AK_cache_AK_malloc ( )
```

Function that 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

### 7.47.2.2 AK_cache_block()

```
int AK_cache_block (
            int num,
            AK_mem_block * mem_block )
```

Function that caches a block into the memory.

**Author**

Nikola Bakoš, Matija Šestak(revised)

**Parameters**

| | |
|---|---|
| *num* | block number (address) |
| *mem_block* | 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

#### 7.47.2.3 AK_cache_result()

```
void AK_cache_result (
            char * srcTable,
            AK_block * temp_block,
            AK_header header[] )
```

Function that caches the fetched result block in memory.

**Author**

Mario Novoselec

#### 7.47.2.4 AK_find_AK_free_space()

```
int AK_find_AK_free_space (
            table_addresses * addresses )
```

Function that finds AK_free space in some block betwen block addresses. It's made for insert_row()

**Author**

Matija Novak, updated by Matija Šestak( function now uses caching)

**Parameters**

| | |
|---|---|
| *address* | addresses of extents |

**Returns**

address of the block to write in

#### 7.47.2.5 AK_find_available_result_block()

```
int AK_find_available_result_block ( )
```

Function that finds the available block for result caching in a circular array.

**Author**

Mario Novoselec

**Returns**

available_index

#### 7.47.2.6 AK_flush_cache()

```
int AK_flush_cache ( )
```

Function that flushes memory blocks to disk file.

**Author**

Matija Šestak, updated by Antonio Martinović

**Returns**

EXIT_SUCCESS

if block form cache can not be writed to DB file -> EXIT_ERROR

block is clean after successfuly writing it to disk

#### 7.47.2.7 AK_generate_result_id()

```
unsigned long AK_generate_result_id (
            unsigned char * str )
```

Function that generates a unique hash identifier for each cached result by using djb2 algorithm.

**Author**

Mario Novoselec

**Returns**

hash

#### 7.47.2.8 AK_get_block()

```
AK_mem_block* AK_get_block (
            int num )
```

Function that reads a block from the 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, Antonio Martinović

**Parameters**

| | |
|---|---|
| *num* | block number (address) |

**Returns**

segment start address

found cached! we're done here

while looking for block we also want to find an empty block in case that the actual block is not found then there is no need to run through the blocks twice

created new cache block for specified address

no free cache blocks found, we need to clear some now

no cache for you

#### 7.47.2.9 AK_get_index_addresses()

```
table_addresses* AK_get_index_addresses (
            char * index )
```

Function for getting addresses of some index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *index* | index name that you search for |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

#### 7.47.2.10 AK_get_index_segment_addresses()

```
table_addresses* AK_get_index_segment_addresses (
            char * segmentName )
```

Function for getting a index segment address.

@Author Antonio Martinović

**Parameters**

| | |
|---|---|
| *segmentName* | table name that you search for |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

#### 7.47.2.11 AK_get_segment_addresses()

```
table_addresses* AK_get_segment_addresses (
            char * segmentName )
```

Function for getting a relation segment address.

Function for getting a index segment address.

@Author Antonio Martinović

**Parameters**

| | |
|---|---|
| *segmentName* | table name that you search for |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

#### 7.47.2.12 AK_get_segment_addresses_internal()

```
table_addresses* AK_get_segment_addresses_internal (
            char * tableName,
            char * segmentName )
```

Function for getting addresses of some table.

**Author**

Matija Novak, updated by Matija Šestak, Mislav Čakarić, Antonio Martinović

**Parameters**

| | |
|---|---|
| *tableName* | table name that you search for |
| *segmentName* | segment name |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

#### 7.47.2.13 AK_get_system_table_address()

```
int AK_get_system_table_address (
            const char * name )
```

Function that gets the address of a system table by name.

**Author**

Matija Novak, updated by Matija Šestak, Mislav Čakarić, Antonio Martinović

**Parameters**

| | |
|---|---|
| *name* | of system table |

**Returns**

table address

#### 7.47.2.14 AK_get_table_addresses()

```
table_addresses* AK_get_table_addresses (
            char * table )
```

Function for getting addresses of some table.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *table* | table name that you search for |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

### 7.47.2.15 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**

| | |
|---|---|
| *table_name* | name of segment to extent |
| *extent_type* | 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

### 7.47.2.16 AK_mem_block_modify()

```
int AK_mem_block_modify (
            AK_mem_block * mem_block,
            int dirty )
```

Function that modifies the "dirty" bit of a block, and update the timestamps accordingly.

**Author**

Alen Novosel.

#### 7.47.2.17 AK_memoman_init()

```
int AK_memoman_init ( )
```

Function that initializes the 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

#### 7.47.2.18 AK_memoman_test()

```
TestResult AK_memoman_test ( )
```

#### 7.47.2.19 AK_memoman_test2()

```
TestResult AK_memoman_test2 ( )
```

#### 7.47.2.20 AK_query_mem_AK_free()

```
void AK_query_mem_AK_free ( )
```

Function that releases the global query memory (variable query_mem)

**Author**

Elvis Popović

#### 7.47.2.21 AK_query_mem_AK_malloc()

```
int AK_query_mem_AK_malloc ( )
```

Function that 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

#### 7.47.2.22 AK_redo_log_AK_malloc()

```
int AK_redo_log_AK_malloc ( )
```

Function that 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

#### 7.47.2.23 AK_refresh_cache()

```
int AK_refresh_cache ( )
```

Function that re-reads all the blocks from the disk.

**Author**

Matija Šestak.

**Returns**

EXIT_SUCCESS

#### 7.47.2.24 AK_release_oldest_cache_block()

```
int AK_release_oldest_cache_block ( )
```

Functions that flushes the oldest block to disk and recalculates the next block to remove.

**Author**

Antonio Martinović

**Returns**

index of flushed cache block

if block form cache can not be writed to DB file -> EXIT_ERROR

block is clean after successfuly writing it to disk

## 7.48 mm/memoman.h File Reference

```
#include "../auxi/test.h"
#include "../dm/dbman.h"
#include "../auxi/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[])

  *Function that caches the fetched result block in memory.*
- int AK_find_available_result_block ()

  *Function that finds the available block for result caching in a circular array.*
- unsigned long AK_generate_result_id (unsigned char *str)

  *Function that generates a unique hash identifier for each cached result by using djb2 algorithm.*
- int AK_cache_block (int num, AK_mem_block *mem_block)

  *Function that caches a block into the memory.*
- int AK_cache_AK_malloc ()

  *Function that initializes the global cache memory (variable db_cache)*
- int AK_redo_log_AK_malloc ()

  *Function that initializes the global redo log memory (variable redo_log)*
- int AK_query_mem_AK_malloc ()

  *Function that initializes the global query memory (variable query_mem)*
- void AK_query_mem_AK_free ()

  *Function that releases the global query memory (variable query_mem)*
- int AK_memoman_init ()

  *Function that initializes the memory manager (cache, redo log and query memory)*
- AK_mem_block * AK_get_block (int num)

  *Function that reads a block from the 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.*
- int AK_release_oldest_cache_block ()

  *Functions that flushes the oldest block to disk and recalculates the next block to remove.*
- int AK_mem_block_modify (AK_mem_block *mem_block, int dirty)

  *Function that modifies the "dirty" bit of a block, and update the timestamps accordingly.*
- int AK_refresh_cache ()

  *Function that re-reads all the blocks from the disk.*
- table_addresses * AK_get_segment_addresses_internal (char *tableName, char *segmentName)

  *Function for getting addresses of some table.*
- table_addresses * AK_get_segment_addresses (char *segmentName)

  *Function for getting a index segment address.*
- table_addresses * AK_get_index_segment_addresses (char *segmentName)

  *Function for getting a index segment address.*
- 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 that finds AK_free space in some block betwen 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.*
- TestResult AK_memoman_test ()
- TestResult 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.*

### 7.48.1 Detailed Description

Header file that contains data structures, defines and functions for the memory manager of Kalashnikov DB

### 7.48.2 Function Documentation

#### 7.48.2.1 AK_cache_AK_malloc()

```
int AK_cache_AK_malloc ( )
```

Function that 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

#### 7.48.2.2 AK_cache_block()

```
int AK_cache_block (
            int num,
            AK_mem_block * mem_block )
```

Function that caches a block into the memory.

**Author**

Nikola Bakoš, Matija Šestak(revised)

**Parameters**

| | |
|---|---|
| *num* | block number (address) |
| *mem_block* | 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

#### 7.48.2.3 AK_cache_result()

```
void AK_cache_result (
            char * srcTable,
            AK_block * temp_block,
            AK_header header[] )
```

Function that caches the fetched result block in memory.

**Author**

Mario Novoselec

#### 7.48.2.4 AK_find_AK_free_space()

```
int AK_find_AK_free_space (
            table_addresses * addresses )
```

Function that finds AK_free space in some block betwen block addresses. It's made for insert_row()

**Author**

Matija Novak, updated by Matija Šestak( function now uses caching)

**Parameters**

| | |
|---|---|
| *address* | addresses of extents |

**Returns**

address of the block to write in

#### 7.48.2.5 AK_find_available_result_block()

```
int AK_find_available_result_block ( )
```

Function that finds the available block for result caching in a circular array.

**Author**

Mario Novoselec

**Returns**

available_index

#### 7.48.2.6 AK_flush_cache()

```
int AK_flush_cache ( )
```

Function that flushes memory blocks to disk file.

**Author**

Matija Šestak, updated by Antonio Martinović

**Returns**

EXIT_SUCCESS

if block form cache can not be writed to DB file -> EXIT_ERROR

block is clean after successfuly writing it to disk

#### 7.48.2.7 AK_generate_result_id()

```
unsigned long AK_generate_result_id (
            unsigned char * str )
```

Function that generates a unique hash identifier for each cached result by using djb2 algorithm.

**Author**

Mario Novoselec

**Returns**

hash

#### 7.48.2.8 AK_get_block()

```
AK_mem_block* AK_get_block (
            int num )
```

Function that reads a block from the 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, Antonio Martinović

**Parameters**

| | |
|---|---|
| *num* | block number (address) |

**Returns**

segment start address

found cached! we're done here

while looking for block we also want to find an empty block in case that the actual block is not found then there is no need to run through the blocks twice

created new cache block for specified address

no free cache blocks found, we need to clear some now

no cache for you

#### 7.48.2.9 AK_get_index_addresses()

```
table_addresses* AK_get_index_addresses (
            char * index )
```

Function for getting addresses of some index.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *index* | index name that you search for |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

#### 7.48.2.10 AK_get_index_segment_addresses()

```
table_addresses* AK_get_index_segment_addresses (
            char * segmentName )
```

Function for getting a index segment address.

@Author Antonio Martinović

**Parameters**

| | |
|---|---|
| *segmentName* | table name that you search for |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

#### 7.48.2.11 AK_get_segment_addresses()

```
table_addresses* AK_get_segment_addresses (
            char * segmentName )
```

Function for getting a index segment address.

@Author Antonio Martinović

**Parameters**

| | |
|---|---|
| *segmentName* | table name that you search for |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

Function for getting a index segment address.

@Author Antonio Martinović

**Parameters**

| | |
|---|---|
| *segmentName* | table name that you search for |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

### 7.48.2.12 AK_get_segment_addresses_internal()

```
table_addresses* AK_get_segment_addresses_internal (
            char * tableName,
            char * segmentName )
```

Function for getting addresses of some table.

**Author**

Matija Novak, updated by Matija Šestak, Mislav Čakarić, Antonio Martinović

**Parameters**

| | |
|---|---|
| *tableName* | table name that you search for |
| *segmentName* | segment name |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

### 7.48.2.13 AK_get_table_addresses()

```
table_addresses* AK_get_table_addresses (
            char * table )
```

Function for getting addresses of some table.

**Author**

Mislav Čakarić

**Parameters**

| | |
|---|---|
| *table* | table name that you search for |

**Returns**

structure table_addresses witch contains start and end adresses of table extents, when form and to are 0 you are on the end of addresses

### 7.48.2.14 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**

| | |
|---|---|
| *table_name* | name of segment to extent |
| *extent_type* | 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

### 7.48.2.15 AK_mem_block_modify()

```
int AK_mem_block_modify (
            AK_mem_block * mem_block,
            int dirty )
```

Function that modifies the "dirty" bit of a block, and update the timestamps accordingly.

**Author**

Alen Novosel.

#### 7.48.2.16 AK_memoman_init()

```
int AK_memoman_init ( )
```

Function that initializes the 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

#### 7.48.2.17 AK_memoman_test()

```
TestResult AK_memoman_test ( )
```

#### 7.48.2.18 AK_memoman_test2()

```
TestResult AK_memoman_test2 ( )
```

#### 7.48.2.19 AK_query_mem_AK_free()

```
void AK_query_mem_AK_free ( )
```

Function that releases the global query memory (variable query_mem)

**Author**

Elvis Popović

#### 7.48.2.20 AK_query_mem_AK_malloc()

```
int AK_query_mem_AK_malloc ( )
```

Function that 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

#### 7.48.2.21 AK_redo_log_AK_malloc()

```
int AK_redo_log_AK_malloc ( )
```

Function that 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

#### 7.48.2.22 AK_refresh_cache()

```
int AK_refresh_cache ( )
```

Function that re-reads all the blocks from the disk.

**Author**

Matija Šestak.

**Returns**

EXIT_SUCCESS

#### 7.48.2.23 AK_release_oldest_cache_block()

```
int AK_release_oldest_cache_block ( )
```

Functions that flushes the oldest block to disk and recalculates the next block to remove.

**Author**

Antonio Martinović

**Returns**

index of flushed cache block

if block form cache can not be writed to DB file -> EXIT_ERROR

block is clean after successfuly writing it to disk

### 7.48.3 Variable Documentation

#### 7.48.3.1 db_cache

```
db_cache
```

Variable that defines the db cache.

#### 7.48.3.2 query_mem

```
query_mem
```

Variable that defines the global query memory.

#### 7.48.3.3 redo_log

```
redo_log
```

Variable that defines the global redo log.

## 7.49 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)

  *Function that prints optimization table for testing purposes.*
- struct list_node * AK_execute_rel_eq (struct list_node *list_query, const char rel_eq, const char *FLAGS)

  *Function that calls and executes 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)

  *Function that executes 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.*
- TestResult AK_query_optimization_test ()

## Variables

- int error_message =0

### 7.49.1 Detailed Description

Provides functions for general query optimization

### 7.49.2 Function Documentation

#### 7.49.2.1 AK_execute_rel_eq()

```
struct list_node* AK_execute_rel_eq (
            struct list_node * list_query,
            const char rel_eq,
            const char * FLAGS )
```

Function that calls and executes 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

#### 7.49.2.2 AK_print_optimized_query()

```
void AK_print_optimized_query (
            struct list_node * list_query )
```

Function that prints optimization table for testing purposes.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *∗list_query* | optimized RA expresion list |

**Returns**

list output

#### 7.49.2.3 AK_query_optimization()

```
struct list_node* AK_query_optimization (
            struct list_node * list_query,
            const char * FLAGS,
            const int DIFF_PLANS )
```

Function that executes 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

#### 7.49.2.4 AK_query_optimization_test()

```
TestResult AK_query_optimization_test ( )
```

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *Function* | for testing *list_query query to be optimized |

**Returns**

No return value

### 7.49.3 Variable Documentation

#### 7.49.3.1 error_message

```
int error_message =0
```

## 7.50 opti/query_optimization.h File Reference

```
#include "../auxi/test.h"
#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)

  *Function that prints optimization table for testing purposes.*
- struct list_node * AK_execute_rel_eq (struct list_node *list_query, const char rel_eq, const char *FLAGS)

  *Function that calls and executes 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)

  *Function that executes 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.*
- TestResult AK_query_optimization_test ()

### 7.50.1 Detailed Description

Header file that provides data structure, functions and defines for general query optimization

### 7.50.2 Macro Definition Documentation

#### 7.50.2.1 MAX_PERMUTATION

```
#define MAX_PERMUTATION 24
```

Constant declaring maximum number of permutations.

### 7.50.3 Function Documentation

#### 7.50.3.1 AK_execute_rel_eq()

```
struct list_node* AK_execute_rel_eq (
            struct list_node * list_query,
            const char rel_eq,
            const char * FLAGS )
```

Function that calls and executes 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

#### 7.50.3.2 AK_print_optimized_query()

```
void AK_print_optimized_query (
            struct list_node * list_query )
```

Function that prints optimization table for testing purposes.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *∗list_query* | optimized RA expresion list |

**Returns**

list output

#### 7.50.3.3 AK_query_optimization()

```
struct list_node* AK_query_optimization (
            struct list_node * list_query,
            const char * FLAGS,
            const int DIFF_PLANS )
```

Function that executes 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

#### 7.50.3.4 AK_query_optimization_test()

```
TestResult AK_query_optimization_test ( )
```

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *Function* | for testing ∗list_query query to be optimized |

**Returns**

No return value

## 7.51 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 generation of 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.*
- TestResult AK_rel_eq_assoc_test ()

  *Function for testing relational equivalences regarding associativity.*

### 7.51.1 Detailed Description

Provides functions for relational equivalences regarding associativity

### 7.51.2 Function Documentation

#### 7.51.2.1 AK_compare()

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

Function for Struct cost_eval comparison.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *a | first value |
| *b | second value |

**Returns**

returns result of comparison

#### 7.51.2.2 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**

| | |
|---|---|
| *list_rel_eq | RA expresion as the struct list_node |

**Returns**

optimised RA expresion as the struct list_node

#### 7.51.2.3 AK_rel_eq_assoc()

```
struct list_node* AK_rel_eq_assoc (
            struct list_node * list_rel_eq )
```

Main function for generation of RA expresion according to associativity equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *∗list_rel_eq* | RA expresion as the struct list_node |

**Returns**

optimised RA expresion as the struct list_node

#### 7.51.2.4 AK_rel_eq_assoc_test()

```
TestResult AK_rel_eq_assoc_test ( )
```

Function for testing relational equivalences regarding associativity.

**Author**

Dino Laktašić.

**Returns**

No return value

## 7.52 opti/rel_eq_assoc.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../auxi/mempro.h"
#include "../auxi/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

  *Stucture 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 generation of 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.*
- TestResult AK_rel_eq_assoc_test ()

  *Function for testing relational equivalences regarding associativity.*

### 7.52.1 Detailed Description

Header file that provides data structures, functions and defines for relational equivalences regarding associativity

### 7.52.2 Typedef Documentation

#### 7.52.2.1 cost_eval

```
typedef struct cost_eval_t cost_eval
```

### 7.52.3 Function Documentation

#### 7.52.3.1 AK_compare()

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

Function for Struct cost_eval comparison.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *∗a* | first value |
| *∗b* | second value |

**Returns**

returns result of comparison

#### 7.52.3.2 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**

| | |
|---|---|
| *∗list_rel_eq* | RA expresion as the struct list_node |

**Returns**

optimised RA expresion as the struct list_node

#### 7.52.3.3 AK_rel_eq_assoc()

```
struct list_node* AK_rel_eq_assoc (
            struct list_node * list_rel_eq )
```

Main function for generation of RA expresion according to associativity equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *∗list_rel_eq* | RA expresion as the struct list_node |

**Returns**

optimised RA expresion as the struct list_node

#### 7.52.3.4 AK_rel_eq_assoc_test()

```
TestResult AK_rel_eq_assoc_test ( )
```

Function for testing relational equivalences regarding associativity.

**Author**

Dino Laktašić.

**Returns**

No return value

## 7.53 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)
  *Function that checks if the selection can commute with theta-join or product.*
- TestResult AK_rel_eq_comut_test ()
  *Function that tests relational equivalences regarding commutativity.*

### 7.53.1 Detailed Description

Provides functions for relational equivalences regarding commutativity

### 7.53.2 Function Documentation

#### 7.53.2.1 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**

| | |
|---|---|
| *∗list_rel_eq* | RA expresion as the struct list_node |

#### 7.53.2.2 AK_rel_eq_commute_with_theta_join()

```
char* AK_rel_eq_commute_with_theta_join (
            char * cond,
            char * tblName )
```

Function that checks if the 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**

| | |
|---|---|
| *∗cond* | condition array that contains condition data |
| *∗tblName* | name of the table |

**Returns**

pointer to char array that contains new condition for a given table

#### 7.53.2.3 AK_rel_eq_comut()

```
struct list_node* AK_rel_eq_comut (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to commutativity equivalence rules.

**Author**

Davor Tomala

**Parameters**

| | |
|---|---|
| *∗list_rel_eq* | RA expresion as the struct list_node |

**Returns**

optimised RA expresion as the struct list_node

#### 7.53.2.4 AK_rel_eq_comut_test()

```
TestResult AK_rel_eq_comut_test ( )
```

Function that tests relational equivalences regarding commutativity.

**Author**

Dino Laktašić (AK_rel_eq_commute_with_theta_join), Davor Tomala (AK_rel_eq_comut)

**Returns**

No return vlaue

## 7.54 opti/rel_eq_comut.h File Reference

```
#include "../auxi/test.h"
#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)

  *Function that checks if the selection can commute with theta-join or product.*
- TestResult AK_rel_eq_comut_test ()

  *Function that tests relational equivalences regarding commutativity.*

### 7.54.1 Detailed Description

Header file that provides data structures,functions and defines for relational equivalences regarding comutativity

## 7.54.2 Function Documentation

### 7.54.2.1 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**

| | |
|---|---|
| *∗list_rel_eq* | RA expresion as the struct list_node |

### 7.54.2.2 AK_rel_eq_commute_with_theta_join()

```
char* AK_rel_eq_commute_with_theta_join (
            char * cond,
            char * tblName )
```

Function that checks if the 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**

| | |
|---|---|
| *∗cond* | condition array that contains condition data |
| *∗tblName* | name of the table |

**Returns**

pointer to char array that contains new condition for a given table

#### 7.54.2.3 AK_rel_eq_comut()

```
struct list_node* AK_rel_eq_comut (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to commutativity equivalence rules.

**Author**

Davor Tomala

**Parameters**

| | |
|---|---|
| *∗list_rel_eq* | RA expresion as the struct list_node |

**Returns**

optimised RA expresion as the struct list_node

#### 7.54.2.4 AK_rel_eq_comut_test()

```
TestResult AK_rel_eq_comut_test ( )
```

Function that tests relational equivalences regarding commutativity.

**Author**

Dino Laktašić (AK_rel_eq_commute_with_theta_join), Davor Tomala (AK_rel_eq_comut)

**Returns**

No return vlaue

## 7.55 opti/rel_eq_projection.c File Reference

```
#include "rel_eq_projection.h"
#include "../auxi/auxiliary.h"
```

Include dependency graph for rel_eq_projection.c:

## Functions

- int AK_rel_eq_is_subset (struct list_node *list_elem_set, struct list_node *list_elem_subset)

  *Function that checks 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)

  *Function that checks if selection uses only attributes retained by the projection before commuting.*
- struct list_node * AK_rel_eq_get_attributes (char *tblName)

  *Function that gets attributes for a given table and store them to the struct list_node.*
- char * AK_rel_eq_projection_attributes (char *attribs, char *tblName)

  *Function used for 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)

  *Function used for 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 expresion.*
- struct list_node * AK_rel_eq_projection (struct list_node *list_rel_eq)

  *Main function for generating RA expresion 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.*
- TestResult AK_rel_eq_projection_test ()

  *Function for testing rel_eq_selection.*

### 7.55.1 Detailed Description

Provides functions for for relational equivalences in projection

### 7.55.2 Function Documentation

#### 7.55.2.1 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**

| | |
|---|---|
| **list_rel_eq* | RA expresion as the AK_list |

**Returns**

No return value

#### 7.55.2.2 AK_rel_eq_can_commute()

```
int AK_rel_eq_can_commute (
            struct list_node * list_elem_attribs,
            struct list_node * list_elem_conds )
```

Function that checks 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**

| | |
|---|---|
| *list_elem_attribs* | list element containing projection data |
| *list_elem_conds* | list element containing selection condition data |

**Returns**

EXIT_SUCCESS if selection uses only attributes retained by projection, else returns EXIT_FAILURE

#### 7.55.2.3 AK_rel_eq_collect_cond_attributes()

```
char* AK_rel_eq_collect_cond_attributes (
            struct list_node * list_elem )
```

Function used for filtering and returning only attributes from selection or theta_join condition.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *list_elem* | list element that contains selection or theta_join condition data |

**Returns**

only attributes from selection or theta_join condition as the AK_list

#### 7.55.2.4 AK_rel_eq_get_attributes()

```
struct list_node* AK_rel_eq_get_attributes (
            char * tblName )
```

Function that gets 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**

| | |
|---|---|
| ∗*tblName* | name of the table |

**Returns**

struct list_node

#### 7.55.2.5 AK_rel_eq_is_subset()

```
int AK_rel_eq_is_subset (
            struct list_node * list_elem_set,
            struct list_node * list_elem_subset )
```

Function that checks 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 u L2 p[L1 u L2](R1 t R2) = (p[L1](R1)) t (p[L2](R2)) Rule 3b. Let L1 u L2 be attributes from R1 and R2, respectively. Let L3 be attributes from R1, but are not in L1 u L2 and let L4 be attributes from R2, but are not in L1 u L2. p[L1 u L2](R1 t R2) = p[L1 u L2]((p[L1 u L3](R1)) t (p[L2 u L4](R2))) Rule 4. distribution according to union p[L](R1 u R2) = (p[L](R1)) u (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**

| | |
|---|---|
| *list_elem_set* | first list element containing projection attributes |
| *list_elem_subset* | second list element containing projection attributes |

**Returns**

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

#### 7.55.2.6 AK_rel_eq_projection()

```
struct list_node* AK_rel_eq_projection (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to projection equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| ∗*list_rel_eq* | RA expresion as the AK_list |

**Returns**

optimised RA expresion as the AK_list

### 7.55.2.7 AK_rel_eq_projection_attributes()

```
char* AK_rel_eq_projection_attributes (
            char * attribs,
            char * tblName )
```

Function used for 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 atributes char array
5. return pointer to char array with stored attribute/s

**Parameters**

| | |
|---|---|
| ∗*attribs* | projection attributes delimited by ";" (ATTR_DELIMITER) |
| ∗*tblName* | name of the table |

**Returns**

filtered list of projection attributes as the AK_list

### 7.55.2.8 AK_rel_eq_projection_test()

```
TestResult AK_rel_eq_projection_test ( )
```

Function for testing rel_eq_selection.

**Author**

Dino Laktašić.

**Returns**

No return value

#### 7.55.2.9 AK_rel_eq_remove_duplicates()

```
char* AK_rel_eq_remove_duplicates (
            char * attribs )
```

Function which removes duplicate attributes from attributes expresion.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *attribs | attributes from which to remove duplicates |

**Returns**

pointer to char array without duplicate attributes

## 7.56 opti/rel_eq_projection.h File Reference

```
#include "../auxi/test.h"
#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)

  *Function that checks 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)

  *Function that checks if selection uses only attributes retained by the projection before commuting.*
- struct list_node * AK_rel_eq_get_attributes (char *tblName)

  *Function that gets attributes for a given table and store them to the struct list_node.*
- char * AK_rel_eq_projection_attributes (char *attribs, char *tblName)

  *Function used for 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)

  *Function used for 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 expresion.*
- struct list_node * AK_rel_eq_projection (struct list_node *list_rel_eq)

  *Main function for generating RA expresion 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.*
- TestResult AK_rel_eq_projection_test ()

  *Function for testing rel_eq_selection.*

## 7.56.1 Detailed Description

Header file that provides data structures, functions and defines for relational equivalences in projection

## 7.56.2 Function Documentation

### 7.56.2.1 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**

| | |
|---|---|
| **list_rel_eq* | RA expresion as the AK_list |

**Returns**

No return value

### 7.56.2.2 AK_rel_eq_can_commute()

```
int AK_rel_eq_can_commute (
            struct list_node * list_elem_attribs,
            struct list_node * list_elem_conds )
```

Function that checks 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**

| | |
|---|---|
| *list_elem_attribs* | list element containing projection data |
| *list_elem_conds* | list element containing selection condition data |

**Returns**

EXIT_SUCCESS if selection uses only attributes retained by projection, else returns EXIT_FAILURE

#### 7.56.2.3 AK_rel_eq_collect_cond_attributes()

```
char* AK_rel_eq_collect_cond_attributes (
            struct list_node * list_elem )
```

Function used for filtering and returning only attributes from selection or theta_join condition.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *list_elem* | list element that contains selection or theta_join condition data |

**Returns**

only attributes from selection or theta_join condition as the AK_list

#### 7.56.2.4 AK_rel_eq_get_attributes()

```
struct list_node* AK_rel_eq_get_attributes (
            char * tblName )
```

Function that gets 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**

| | |
|---|---|
| **tblName* | name of the table |

**Returns**

struct list_node

### 7.56.2.5 AK_rel_eq_is_subset()

```
int AK_rel_eq_is_subset (
            struct list_node * list_elem_set,
            struct list_node * list_elem_subset )
```

Function that checks if some set of attributes is subset of larger set, used in cascading of the projections.

**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**

| | |
|---|---|
| *list_elem_set* | first list element containing projection attributes |
| *list_elem_subset* | second list element containing projection attributes |

**Returns**

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

**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 u L2 p[L1 u L2](R1 t R2) = (p[L1](R1)) t (p[L2](R2)) Rule 3b. Let L1 u L2 be attributes from R1 and R2, respectively. Let L3 be attributes from R1, but are not in L1 u L2 and let L4 be attributes from R2, but are not in L1 u L2. p[L1 u L2](R1 t R2) = p[L1 u L2]((p[L1 u L3](R1)) t (p[L2 u L4](R2))) Rule 4. distribution according to union p[L](R1 u R2) = (p[L](R1)) u (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**

| | |
|---|---|
| *list_elem_set* | first list element containing projection attributes |
| *list_elem_subset* | second list element containing projection attributes |

**Returns**

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

#### 7.56.2.6 AK_rel_eq_projection()

```
struct list_node* AK_rel_eq_projection (
            struct list_node * list_rel_eq )
```

Main function for generating RA expresion according to projection equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *∗list_rel_eq* | RA expresion as the AK_list |

**Returns**

optimised RA expresion as the AK_list

### 7.56.2.7 AK_rel_eq_projection_attributes()

```
char* AK_rel_eq_projection_attributes (
            char * attribs,
            char * tblName )
```

Function used for 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 atributes char array
5. return pointer to char array with stored attribute/s

**Parameters**

| | |
|---|---|
| *∗attribs* | projection attributes delimited by ";" (ATTR_DELIMITER) |
| *∗tblName* | name of the table |

**Returns**

filtered list of projection attributes as the AK_list

### 7.56.2.8 AK_rel_eq_projection_test()

```
TestResult AK_rel_eq_projection_test ( )
```

Function for testing rel_eq_selection.

**Author**

Dino Laktašić.

**Returns**

No return value

#### 7.56.2.9 AK_rel_eq_remove_duplicates()

```
char* AK_rel_eq_remove_duplicates (
            char * attribs )
```

Function which removes duplicate attributes from attributes expresion.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *attribs | attributes from which to remove duplicates |

**Returns**

pointer to char array without duplicate attributes

## 7.57 opti/rel_eq_selection.c File Reference

```
#include "rel_eq_selection.h"
#include "../auxi/auxiliary.h"
```

Include dependency graph for rel_eq_selection.c:

### Functions

- int AK_rel_eq_is_attr_subset (char *set, char *subset)

  *Function that checks if some set of attributes is subset of larger set.*
- char * AK_rel_eq_get_atrributes_char (char *tblName)

  *Function that fetches 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)

  *Function that checks if two sets share one or more of it's attributes.*
- struct list_node * AK_rel_eq_split_condition (char *cond)

  *Function that checks 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.*
- TestResult AK_rel_eq_selection_test ()

  *Function for testing rel_eq_selection.*

## 7.57.1 Detailed Description

Provides functions for for relational equivalences in selection

## 7.57.2 Function Documentation

### 7.57.2.1 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**

| | |
|---|---|
| *list_rel_eq | RA expresion as the struct list_node |

**Returns**

void

### 7.57.2.2 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**

| | |
|---|---|
| *∗cond* | condition array that contains condition data |

**Returns**

pointer to array that contains attributes for a given condition

#### 7.57.2.3 AK_rel_eq_get_atrributes_char()

```
char* AK_rel_eq_get_atrributes_char (
            char * tblName )
```

Function that fetches 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**

| | |
|---|---|
| *∗tblName* | name of the table |

**Returns**

pointer to char array

#### 7.57.2.4 AK_rel_eq_is_attr_subset()

```
int AK_rel_eq_is_attr_subset (
            char * set,
            char * subset )
```

Function that checks 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 ritched
8. on loop exit return EXIT_SUCCESS

**Parameters**

| | |
|---|---|
| ∗*set* | set array |
| ∗*subset* | subset array |

**Returns**

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

### 7.57.2.5 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**

| | |
|---|---|
| ∗*list_rel_eq* | RA expresion as the struct list_node |

**Returns**

optimised RA expresion as the struct list_node

### 7.57.2.6 AK_rel_eq_selection_test()

```
TestResult AK_rel_eq_selection_test ( )
```

Function for testing rel_eq_selection.

**Author**

Dino Laktašić.

**Returns**

No return value

#### 7.57.2.7 AK_rel_eq_share_attributes()

```
int AK_rel_eq_share_attributes (
            char * set,
            char * subset )
```

Function that checks 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**

| | |
|---|---|
| *∗set* | first set of attributes delimited by ";" (ATTR_DELIMITER) |
| *∗subset* | second set of attributes delimited by ";" (ATTR_DELIMITER) |

**Returns**

EXIT_SUCCESS if set and subset share at least one attribute, else returns EXIT_FAILURE

#### 7.57.2.8 AK_rel_eq_split_condition()

```
struct list_node* AK_rel_eq_split_condition (
            char * cond )
```

Function that checks 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)

Break conjunctive conditions to individual conditions.

**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 discarge 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**

| | |
|---|---|
| ∗*cond* | condition array that contains condition data |
| ∗*tblName* | 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**

| | |
|---|---|
| ∗*cond* | condition expression |

**Returns**

conditions list

## 7.58 opti/rel_eq_selection.h File Reference

```
#include "../auxi/test.h"
#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)

  *Function that checks if some set of attributes is subset of larger set.*
- char ∗ AK_rel_eq_get_atrributes_char (char ∗tblName)

  *Function that fetches 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)

  *Function that checks if two sets share one or more of it's attributes.*
- struct list_node ∗ AK_rel_eq_split_condition (char ∗cond)

  *Break conjunctive conditions to individual conditions.*
- 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.*
- TestResult AK_rel_eq_selection_test ()

  *Function for testing rel_eq_selection.*

### 7.58.1 Detailed Description

Header file that provides data structures, functions and defines for relational equivalences in selection

### 7.58.2 Function Documentation

#### 7.58.2.1 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**

| | |
|---|---|
| ∗*list_rel_eq* | RA expresion as the struct list_node |

**Returns**

void

#### 7.58.2.2 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**

| | |
|---|---|
| *∗cond* | condition array that contains condition data |

**Returns**

pointer to array that contains attributes for a given condition

#### 7.58.2.3 AK_rel_eq_get_atrributes_char()

```
char* AK_rel_eq_get_atrributes_char (
            char * tblName )
```

Function that fetches attributes for a given table and store them to the char array.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| *∗tblName* | name of the table |

**Returns**

pointer to 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**

| | |
|---|---|
| ∗*tblName* | name of the table |

**Returns**

pointer to char array

#### 7.58.2.4 AK_rel_eq_is_attr_subset()

```
int AK_rel_eq_is_attr_subset (
            char * set,
            char * subset )
```

Function that checks if some set of attributes is subset of larger set.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| ∗*set* | set array |
| ∗*subset* | subset array |

**Returns**

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

**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**

| | |
|---|---|
| ∗*set* | set array |
| ∗*subset* | subset array |

**Returns**

EXIT_SUCCESS if some set of attributes is subset of larger set, else returns EXIT_FAILURE

#### 7.58.2.5 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**

| | |
|---|---|
| *∗list_rel_eq* | RA expresion as the struct list_node |

**Returns**

optimised RA expresion as the struct list_node

#### 7.58.2.6 AK_rel_eq_selection_test()

```
TestResult AK_rel_eq_selection_test ( )
```

Function for testing rel_eq_selection.

**Author**

Dino Laktašić.

**Returns**

No return value

#### 7.58.2.7 AK_rel_eq_share_attributes()

```
int AK_rel_eq_share_attributes (
            char * set,
            char * subset )
```

Function that checks if two sets share one or more of it's attributes.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| ∗*set* | first set of attributes delimited by ";" (ATTR_DELIMITER) |
| ∗*subset* | second set of attributes delimited by ";" (ATTR_DELIMITER) |

**Returns**

EXIT_SUCCESS if set and subset share at least one attribute, else returns EXIT_FAILURE

**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**

| | |
|---|---|
| ∗*set* | first set of attributes delimited by ";" (ATTR_DELIMITER) |
| ∗*subset* | second set of attributes delimited by ";" (ATTR_DELIMITER) |

**Returns**

EXIT_SUCCESS if set and subset share at least one attribute, else returns EXIT_FAILURE

### 7.58.2.8 AK_rel_eq_split_condition()

```
struct list_node* AK_rel_eq_split_condition (
            char * cond )
```

Break conjunctive conditions to individual conditions.

**Author**

Dino Laktašić.

**Parameters**

| | |
|---|---|
| ∗*cond* | condition expression |

**Returns**

conditions list

Break conjunctive conditions to individual conditions.

**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 discarge 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**

| | |
|---|---|
| ∗*cond* | condition array that contains condition data |
| ∗*tblName* | 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**

| | |
|---|---|
| ∗*cond* | condition expression |

**Returns**

conditions list

## 7.59 rec/archive_log.c File Reference

```
#include "archive_log.h"
```

Include dependency graph for archive_log.c:

## Functions

- void AK_archive_log (int sig)

  *Function for making archive log.*
- char * AK_get_timestamp ()

  *Function that returns the current timestamp.*
- int AK_check_folder_archivelog ()

## 7.59.1 Function Documentation

### 7.59.1.1 AK_archive_log()

```
void AK_archive_log (
            int sig )
```

Function for making archive log.

Function that 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 retun value

### 7.59.1.2 AK_check_folder_archivelog()

```
int AK_check_folder_archivelog ( )
```

#### 7.59.1.3 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.bin

## 7.60 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 "../auxi/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.*
- char * AK_get_timestamp ()

  *Function that returns the current timestamp.*

### 7.60.1 Detailed Description

Header file that provides functions and defines for archive logging

### 7.60.2 Function Documentation

#### 7.60.2.1 AK_archive_log()

```
void AK_archive_log (
            int sig )
```

Function for making archive log.

**Author**

Dražen Bandić, update by Tomislav Turek

**Returns**

No retun value

Function that 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 retun value

#### 7.60.2.2 AK_get_timestamp()

```
char* AK_get_timestamp ( )
```

Function that returns the current timestamp.

**Author**

Dražen Bandić main logic, replaced by Tomislav Turek

**Returns**

char array in format day.month.year-hour:min:sec.usecu.bin

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.bin

## 7.61 rec/recovery.c File Reference

`#include "recovery.h"`
Include dependency graph for recovery.c:

### Functions

- void AK_recover_archive_log (char *fileName)

  *Function that reads the binary file in which last commands were saved, and executes them.*
- void AK_recovery_insert_row (char *table, int commandNumber)

  *Function that inserts a new row in the table with attributes.*
- int recovery_insert_row (char *table, char **attr_name, char **attributes, int n, int *type)

  *Function that inserts row in table.*
- char ** AK_recovery_tokenize (char *input, char *delimiter, int valuesOrNot)

  *Function that 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.*
- TestResult AK_recovery_test ()

  *Function for recovery testing.*
- void AK_load_chosen_log ()

  *Executes the recovery operation for the chosen bin file.*
- void AK_load_latest_log ()

  *Executes the recovery operation for the latest bin file.*

### Variables

- short grandfailure = 0

### 7.61.1 Function Documentation

#### 7.61.1.1 AK_load_chosen_log()

`void AK_load_chosen_log ( )`

Executes the recovery operation for the chosen bin file.

Function lists the contents of the archive_log directory. The user then types in the name of the desired bin file to open and perform the neccessary actions.

**Author**

Matija Večenaj

**Parameters**

| | |
|---|---|
| *none* | |

**Returns**

no value

#### 7.61.1.2 AK_load_latest_log()

```
void AK_load_latest_log ( )
```

Executes the recovery operation for the latest bin file.

Function reads the latest.txt file which contains the name of the latest bin file that's been created. Then it loads it and does the neccessary recovery operations.

**Author**

Matija Večenaj

**Parameters**

| | |
|---|---|
| *none* | |

**Returns**

no value

#### 7.61.1.3 AK_recover_archive_log()

```
void AK_recover_archive_log (
            char * fileName )
```

Function that reads the binary file in which 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**

| | |
|---|---|
| *fileName* | - name of the archive log |

**Returns**

no value

#### 7.61.1.4 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**

| | |
|---|---|
| *sig* | required integer parameter for SIGINT handler functions |

#### 7.61.1.5 AK_recovery_insert_row()

```
void AK_recovery_insert_row (
            char * table,
            int commandNumber )
```

Function that inserts a new row in the 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**

| | |
|---|---|
| *table* | - table name to insert to |
| *commandNumber* | - number of current command |

**Returns**

no value

#### 7.61.1.6 AK_recovery_test()

```
TestResult 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

#### 7.61.1.7 AK_recovery_tokenize()

```
char** AK_recovery_tokenize (
            char * input,
            char * delimiter,
            int valuesOrNot )
```

Function that 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**

| | |
|---|---|
| *input* | - input to tokenize |
| *delimiter* | - delimiter |
| *valuesOrNot* | - 1 if the input are values, 0 otherwise |

**Returns**

new double pointer structure with tokens

#### 7.61.1.8 recovery_insert_row()

```
int recovery_insert_row (
            char * table,
            char ** attr_name,
            char ** attributes,
            int n,
            int * type )
```

Function that inserts row in table.

**Author**

Danko Bukovac

**Returns**

EXIT_SUCCESS if insert is successful, else EXIT_FAILURE

### 7.61.2 Variable Documentation

#### 7.61.2.1 grandfailure

```
short grandfailure = 0
```

this variable flags if system failed

## 7.62 rec/recovery.h File Reference

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

## Functions

- void AK_recover_archive_log (char ∗fileName)

  *Function that reads the binary file in which last commands were saved, and executes them.*
- void AK_recovery_insert_row (char ∗table, int commandNumber)

  *Function that inserts a new row in the table with attributes.*
- char ∗∗ AK_recovery_tokenize (char ∗input, char ∗delimiter, int valuesOrNot)

  *Function that tokenizes the input with the given delimiter and puts them in an double pointer structure (so we can execute an insert)*
- TestResult AK_recovery_test ()

  *Function for recovery testing.*
- void AK_recover_operation (int sig)

  *Function that recovers and executes failed commands.*
- void AK_load_chosen_log ()

  *Executes the recovery operation for the chosen bin file.*
- void AK_load_latest_log ()

  *Executes the recovery operation for the latest bin file.*

## 7.62.1 Function Documentation

### 7.62.1.1 AK_load_chosen_log()

```
void AK_load_chosen_log ( )
```

Executes the recovery operation for the chosen bin file.

Function lists the contents of the archive_log directory. The user then writes the name of the desired bin file to perform the neccessary actions.

**Author**

Matija Večenaj

**Parameters**

| | |
|---|---|
| *none* | |

**Returns**

no value

Function lists the contents of the archive_log directory. The user then types in the name of the desired bin file to open and perform the neccessary actions.

**Author**

Matija Večenaj

**Parameters**

| | |
|---|---|
| *none* | |

**Returns**

no value

### 7.62.1.2 AK_load_latest_log()

```
void AK_load_latest_log ( )
```

Executes the recovery operation for the latest bin file.

Function reads the latest.txt file which contains the name of the latest bin file that's been created. Then it loads it and does the neccessary recovery operations.

**Author**

Matija Večenaj

**Parameters**

| | |
|---|---|
| *none* | |

**Returns**

no value

#### 7.62.1.3 AK_recover_archive_log()

```
void AK_recover_archive_log (
            char * fileName )
```

Function that reads the binary file in which 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**

| | |
|---|---|
| *fileName* | - name of the archive log |

**Returns**

no value

#### 7.62.1.4 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**

| | |
|---|---|
| *sig* | required integer parameter for SIGINT handler functions |

#### 7.62.1.5 AK_recovery_insert_row()

```
void AK_recovery_insert_row (
            char * table,
            int commandNumber )
```

Function that inserts a new row in the 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**

| | |
|---|---|
| *table* | - table name to insert to |
| *commandNumber* | - number of current command |

**Returns**

no value

#### 7.62.1.6 AK_recovery_test()

```
TestResult 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

#### 7.62.1.7 AK_recovery_tokenize()

```
char** AK_recovery_tokenize (
            char * input,
            char * delimiter,
            int valuesOrNot )
```

Function that 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**

| | |
|---|---|
| *input* | - input to tokenize |
| *delimiter* | - delimiter |
| *valuesOrNot* | - 1 if the input are values, 0 otherwise |

**Returns**

new double pointer structure with tokens

## 7.63 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 that adds a new element to redolog.*
- void AK_redolog_commit ()
- int AK_add_to_redolog_select (int command, struct list_node *condition, char *srcTable)

  *Function that adds a new select to redolog, commented code with the new select from select.c, current code works with selection.c.*
- int AK_check_redo_log_select (int command, struct list_node *condition, char *srcTable)

  *Function that checks redolog for select, works only with selection.c, not select.c.*
- void AK_printout_redolog ()

  *Function that prints out the content of redolog memory.*
- char * AK_check_attributes (char *attributes)

  *Function that checks if the attribute contains '|', and if it does it replaces it with "\|".*

### 7.63.1 Detailed Description

Provides redolog functions.

## 7.63.2 Function Documentation

### 7.63.2.1 AK_add_to_redolog()

```
int AK_add_to_redolog (
            int command,
            struct list_node * row_root )
```

Function that adds a 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

### 7.63.2.2 AK_add_to_redolog_select()

```
int AK_add_to_redolog_select (
            int command,
            struct list_node * condition,
            char * srcTable )
```

Function that adds a new select to redolog, commented code with the new select from select.c, current code works with selection.c.

**Author**

Danko Bukovac

**Returns**

EXIT_FAILURE if not allocated memory for ispis, otherwise EXIT_SUCCESS

### 7.63.2.3 AK_check_attributes()

```
char* AK_check_attributes (
            char * attributes )
```

Function that checks if the attribute contains '|', and if it does it replaces it with "\|".

**Author**

Dražen Bandić

**Returns**

new attribute

#### 7.63.2.4 AK_check_redo_log_select()

```
int AK_check_redo_log_select (
            int command,
            struct list_node * condition,
            char * srcTable )
```

Function that checks redolog for select, works only with selection.c, not select.c.

**Author**

Danko Bukovac

**Returns**

0 if select was not found, otherwise 1

#### 7.63.2.5 AK_printout_redolog()

```
void AK_printout_redolog ( )
```

Function that prints out the content of redolog memory.

**Author**

Krunoslav Bilić updated by Dražen Bandić, second update by Tomislav Turek

**Returns**

No return value.

#### 7.63.2.6 AK_redolog_commit()

```
void AK_redolog_commit ( )
```

## 7.64 rec/redo_log.h File Reference

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

## Functions

- int AK_add_to_redolog (int command, struct list_node *row_root)

  *Function that adds a new element to redolog.*
- int AK_add_to_redolog_select (int command, struct list_node *condition, char *srcTable)

  *Function that adds a new select to redolog, commented code with the new select from select.c, current code works with selection.c.*
- int AK_check_redo_log_select (int command, struct list_node *condition, char *srcTable)

  *Function that checks redolog for select, works only with selection.c, not select.c.*
- void AK_printout_redolog ()

  *Function that prints out the content of redolog memory.*
- void AK_redolog_commit ()
- char * AK_check_attributes (char *attributes)

  *Function that checks if the attribute contains '|', and if it does it replaces it with "\|".*

### 7.64.1 Function Documentation

#### 7.64.1.1 AK_add_to_redolog()

```
int AK_add_to_redolog (
            int command,
            struct list_node * row_root )
```

Function that adds a 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

#### 7.64.1.2 AK_add_to_redolog_select()

```
int AK_add_to_redolog_select (
            int command,
            struct list_node * condition,
            char * srcTable )
```

Function that adds a new select to redolog, commented code with the new select from select.c, current code works with selection.c.

**Author**

Danko Bukovac

**Returns**

EXIT_FAILURE if not allocated memory for ispis, otherwise EXIT_SUCCESS

#### 7.64.1.3 AK_check_attributes()

```
char* AK_check_attributes (
            char * attributes )
```

Function that checks if the attribute contains '|', and if it does it replaces it with "\|".

**Author**

Dražen Bandić

**Returns**

new attribute

#### 7.64.1.4 AK_check_redo_log_select()

```
int AK_check_redo_log_select (
            int command,
            struct list_node * condition,
            char * srcTable )
```

Function that checks redolog for select, works only with selection.c, not select.c.

**Author**

Danko Bukovac

**Returns**

0 if select was not found, otherwise 1

#### 7.64.1.5 AK_printout_redolog()

```
void AK_printout_redolog ( )
```

Function that prints out the content of redolog memory.

**Author**

Krunoslav Bilić updated by Dražen Bandić, second update by Tomislav Turek

**Returns**

No return value.

#### 7.64.1.6 AK_redolog_commit()

```
void AK_redolog_commit ( )
```

## 7.65 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)

  *Function that 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.*
- int AK_header_size (AK_header *header)

  *Function that calculates how many attributes there are in the header with a while loop.*
- void AK_agg_input_init (AK_agg_input *input)

  *Function that 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 that 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 that adds a header with a task on the beginning of the input object for aggregation. With the use of for loop existing attributes and tasks are moved from one place forward in input object.*
- void AK_agg_input_fix (AK_agg_input *input)

  *function that handles AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with a value of -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 that 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 assignes 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.*
- TestResult AK_aggregation_test ()

### 7.65.1 Detailed Description

Provides functions for aggregation and grouping

### 7.65.2 Function Documentation

#### 7.65.2.1 AK_agg_input_add()

```
int AK_agg_input_add (
              AK_header header,
              int agg_task,
              AK_agg_input * input )
```

Function that adds a header with a task in input object for aggregation.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *header* | a header that is being aggregated |
| *agg_task* | the task which is to be done on the header |
| *input* | the input object |

**Returns**

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

#### 7.65.2.2 AK_agg_input_add_to_beginning()

```
int AK_agg_input_add_to_beginning (
              AK_header header,
              int agg_task,
              AK_agg_input * input )
```

Function that adds a header with a task on the beginning of the input object for aggregation. With the use of for loop existing attributes and tasks are moved from one place forward in input object.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *header* | a header that is being aggregated |
| *agg_task* | the task which is to be done on the header |
| *input* | the input object |

**Returns**

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

#### 7.65.2.3 AK_agg_input_fix()

```
void AK_agg_input_fix (
            AK_agg_input * input )
```

function that handles AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with a value of -1. While loop examines whether the task in array is equal to AGG_TASK_AVG. If so, AGG_TAS↩ K_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**

| | |
|---|---|
| *input* | the input object |

**Returns**

No return value

#### 7.65.2.4 AK_agg_input_init()

```
void AK_agg_input_init (
            AK_agg_input * input )
```

Function that initializes the input object for aggregation with init values.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *input* | the input object |

**Returns**

No return value

#### 7.65.2.5 AK_aggregation()

```
int AK_aggregation (
            AK_agg_input * input,
```

```
char * source_table,
char * agg_table )
```

Function that 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 assignes 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**

| | |
|---|---|
| *input* | input object with list of atributes by which we aggregate and types of aggregations |
| *source_table* | - table name for the source table |
| *agg_table* | table name for aggregated table |

**Returns**

EXIT_SUCCESS if continues succesfuly, 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!

#### 7.65.2.6 AK_aggregation_test()

```
TestResult AK_aggregation_test ( )
```

checking results

This variable was added to handle bug described in this file.

#### 7.65.2.7 AK_header_size()

```
int AK_header_size (
            AK_header * header )
```

Function that calculates how many attributes there are in the header with a while loop.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *header* | A header array |

**Returns**

Number of attributes defined in header array

#### 7.65.2.8 AK_search_unsorted()

```
search_result AK_search_unsorted (
            char * szRelation,
            search_params * aspParams,
            int iNum_search_params )
```

Function that 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**

| | |
|---|---|
| *szRelation* | relation name |
| *aspParams* | array of search parameters |
| *iNum_search_params* | 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

## 7.66 rel/aggregation.h File Reference

```
#include "../auxi/test.h"
#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 atribute name, date and aggregation task associated.*
- struct AK_agg_input

  *Structure that contains attributes from table header, tasks for this table and counter value.*
- struct rowroot_struct

  *Structure that defines a new row in table using list_node.*
- struct projection_att_struct

  *Structure that defines projection_att which is a new list_node.*

### 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 that calculates how many attributes there are in the header with a while loop.*
- void AK_agg_input_init (AK_agg_input *input)

  *Function that 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 that 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 that adds a header with a task on the beginning of the input object for aggregation. With the use of for loop existing attributes and tasks are moved from one place forward in input object.*
- void AK_agg_input_fix (AK_agg_input *input)

  *function that handles AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with a value of -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 that 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 assignes 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.*
- TestResult AK_aggregation_test ()

### 7.66.1 Detailed Description

Header file that provides data structures, functions and defines for aggregation and grouping

### 7.66.2 Macro Definition Documentation

#### 7.66.2.1 AGG_TASK_AVG

```
#define AGG_TASK_AVG 6
```

#### 7.66.2.2 AGG_TASK_AVG_COUNT

```
#define AGG_TASK_AVG_COUNT 10
```

#### 7.66.2.3 AGG_TASK_AVG_SUM

```
#define AGG_TASK_AVG_SUM 11
```

#### 7.66.2.4 AGG_TASK_COUNT

```
#define AGG_TASK_COUNT 2
```

#### 7.66.2.5 AGG_TASK_GROUP

```
#define AGG_TASK_GROUP 1
```

#### 7.66.2.6 AGG_TASK_MAX

```
#define AGG_TASK_MAX 4
```

#### 7.66.2.7 AGG_TASK_MIN

```
#define AGG_TASK_MIN 5
```

#### 7.66.2.8 AGG_TASK_SUM

```
#define AGG_TASK_SUM 3
```

### 7.66.3 Function Documentation

#### 7.66.3.1 AK_agg_input_add()

```
int AK_agg_input_add (
            AK_header header,
            int agg_task,
            AK_agg_input * input )
```

Function that adds a header with a task in input object for aggregation.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *header* | a header that is being aggregated |
| *agg_task* | the task which is to be done on the header |
| *input* | the input object |

**Returns**

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

### 7.66.3.2 AK_agg_input_add_to_beginning()

```
int AK_agg_input_add_to_beginning (
            AK_header header,
            int agg_task,
            AK_agg_input * input )
```

Function that adds a header with a task on the beginning of the input object for aggregation. With the use of for loop existing attributes and tasks are moved from one place forward in input object.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *header* | a header that is being aggregated |
| *agg_task* | the task which is to be done on the header |
| *input* | the input object |

**Returns**

On success, returns EXIT_SUCCESS, otherwise EXIT_FAILURE

### 7.66.3.3 AK_agg_input_fix()

```
void AK_agg_input_fix (
            AK_agg_input * input )
```

function that handles AVG (average) aggregation. It goes through array of tasks in input object until it comes to task with a value of -1. While loop examines whether the task in array is equal to AGG_TASK_AVG. If so, AGG_TAS↩K_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**

| | |
|---|---|
| *input* | the input object |

**Returns**

No return value

#### 7.66.3.4 AK_agg_input_init()

```
void AK_agg_input_init (
            AK_agg_input * input )
```

Function that initializes the input object for aggregation with init values.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *input* | the input object |

**Returns**

No return value

#### 7.66.3.5 AK_aggregation()

```
int AK_aggregation (
            AK_agg_input * input,
            char * source_table,
            char * agg_table )
```

Function that 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 assignes 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**

| | |
|---|---|
| *input* | input object with list of atributes by which we aggregate and types of aggregations |
| *source_table* | - table name for the source table |
| *agg_table* | table name for aggregated table |

**Returns**

EXIT_SUCCESS if continues succesfuly, 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!

#### 7.66.3.6 AK_aggregation_test()

```
TestResult AK_aggregation_test ( )
```

checking results

This variable was added to handle bug described in this file.

#### 7.66.3.7 AK_header_size()

```
int AK_header_size (
            AK_header * header )
```

Function that calculates how many attributes there are in the header with a while loop.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *header* | A header array |

**Returns**

Number of attributes defined in header array

## 7.67 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 that produces a 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 the first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.*
- TestResult AK_op_difference_test ()

  *Function for difference operator testing.*

### 7.67.1 Detailed Description

Provides functions for relational difference operation

### 7.67.2 Function Documentation

#### 7.67.2.1 AK_difference()

```
int AK_difference (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that produces a 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 the 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**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the new table |

**Returns**

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

#### 7.67.2.2 AK_op_difference_test()

```
TestResult AK_op_difference_test ( )
```

Function for difference operator testing.

**Author**

Dino Laktašić

## 7.68 rel/difference.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
#include "../sql/drop.h"
```

Include dependency graph for difference.h: This graph shows which files directly or indirectly include this file:

### Functions

- int AK_difference (char *srcTable1, char *srcTable2, char *dstTable)

  *Function that produces a 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 the first table with every block in extent of second table. If there is a difference between their rows, they are put in dstTable.*
- TestResult AK_op_difference_test ()

  *Function for difference operator testing.*

### 7.68.1 Detailed Description

Header file that provides functions and defines for relational difference operation

### 7.68.2 Function Documentation

#### 7.68.2.1 AK_difference()

```
int AK_difference (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that produces a 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 the 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**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the new table |

**Returns**

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

#### 7.68.2.2 AK_op_difference_test()

```
TestResult AK_op_difference_test ( )
```

Function for difference operator testing.

**Author**

Dino Laktašić

## 7.69 rel/expression_check.c File Reference

```
#include "expression_check.h"
```

Include dependency graph for expression_check.c:

## Functions

- int AK_check_arithmetic_statement (struct list_node ∗el, const char ∗op, const char ∗a, const char ∗b)

  *Function that 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 charachter 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 a given string input.*
- int AK_check_regex_operator_expression (const char ∗value, const char ∗expression)

  *Function that evaluates regex expression on a given string input.*
- int AK_check_if_row_satisfies_expression (struct list_node ∗row_root, struct list_node ∗expr)

  *Function that 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.*
- TestResult AK_expression_check_test ()

### 7.69.1 Detailed Description

Provides functions for constraint checking used in selection and theta-join

### 7.69.2 Function Documentation

#### 7.69.2.1 AK_check_arithmetic_statement()

```
int AK_check_arithmetic_statement (
            struct list_node * el,
            const char * op,
            const char * a,
            const char * b )
```

Function that 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.

Function that compares values according to their data type, checks arithmetic statement which is part of expression given in the function below.

**Author**

Dino Laktašić, abstracted by Tomislav Mikulček,updated by Nikola Miljancic

**Parameters**

| | |
|---|---|
| *el* | list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation |
| ∗*op* | comparison operator |
| ∗*a* | left operand |
| ∗*b* | right operand |

**Returns**

0 if arithmetic statement is false, 1 if arithmetic statement is true

#### 7.69.2.2 AK_check_if_row_satisfies_expression()

```
int AK_check_if_row_satisfies_expression (
            struct list_node * row_root,
            struct list_node * expr )
```

Function that 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.

Function that replaces charachter wildcard (%,_) ch in string s with repl characters.

**Author**

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic, abstracted by Tomislav Mikulček

**Parameters**

| | |
|---|---|
| *row_root* | beginning of the row that is to be evaluated |
| *∗expr* | list with the logical expression in postfix notation |

**Returns**

0 if row does not satisfy, 1 if row satisfies expression

#### 7.69.2.3 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 a given string input.

@Author Leon Palaić

**Parameters**

| | |
|---|---|
| *value* | string value that must match regex expression |
| *expression* | POSIX regex expression |
| *checkWildCard* | replaces SQL wildcard to correesponding POSIX regex charachter |
| *sensitive* | case insensitive indicator 1-case sensitive,0- case insensitive |

**Returns**

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

#### 7.69.2.4 AK_check_regex_operator_expression()

```
int AK_check_regex_operator_expression (
            const char * value,
            const char * expression )
```

Function that evaluates regex expression on a given string input.

@Author Leon Palaić

**Parameters**

| | |
|---|---|
| *value* | string value that must match regex expression |
| *expression* | POSIX regex expression |

**Returns**

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

#### 7.69.2.5 AK_expression_check_test()

```
TestResult AK_expression_check_test ( )
```

#### 7.69.2.6 AK_replace_wild_card()

```
char* AK_replace_wild_card (
            const char * s,
            char ch,
            const char * repl )
```

Function that replaces charachter wildcard (%,_) ch in string s with repl characters.

@Author Leon Palaić

**Parameters**

| | |
|---|---|
| *s* | input string |
| *ch* | charachter to be replaced |

**Returns**

new sequence of charachters

## 7.70 rel/expression_check.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
#include <regex.h>
```

Include dependency graph for expression_check.h: This graph shows which files directly or indirectly include this file:

### Functions

- int AK_check_arithmetic_statement (struct list_node ∗el, const char ∗op, const char ∗a, const char ∗b)

  *Function that compares values according to their data type, checks arithmetic statement which is part of expression given in the function below.*
- int AK_check_if_row_satisfies_expression (struct list_node ∗row_root, struct list_node ∗expr)

  *Function that replaces charachter 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 a given string input.*
- int AK_check_regex_operator_expression (const char ∗value, const char ∗expression)

  *Function that evaluates regex expression on a given string input.*
- TestResult AK_expression_check_test ()

### 7.70.1 Detailed Description

Header file that functions and defines for expression ckecking

### 7.70.2 Function Documentation

#### 7.70.2.1 AK_check_arithmetic_statement()

```
int AK_check_arithmetic_statement (
            struct list_node * el,
            const char * op,
            const char * a,
            const char * b )
```

Function that compares values according to their data type, checks arithmetic statement which is part of expression given in the function below.

**Author**

Dino Laktašić, abstracted by Tomislav Mikulček,updated by Nikola Miljancic

**Parameters**

| | |
|---|---|
| *el* | list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation |
| *∗op* | comparison operator |
| *∗a* | left operand |
| *∗b* | right operand |

**Returns**

0 if arithmetic statement is false, 1 if arithmetic statement is true

Function that compares values according to their data type, checks arithmetic statement which is part of expression given in the function below.

**Author**

Dino Laktašić, abstracted by Tomislav Mikulček,updated by Nikola Miljancic

**Parameters**

| | |
|---|---|
| *el* | list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation |
| *∗op* | comparison operator |
| *∗a* | left operand |
| *∗b* | right operand |

**Returns**

0 if arithmetic statement is false, 1 if arithmetic statement is true

### 7.70.2.2 AK_check_if_row_satisfies_expression()

```
int AK_check_if_row_satisfies_expression (
            struct list_node * row_root,
            struct list_node * expr )
```

Function that replaces charachter wildcard (%,_) ch in string s with repl characters.

@Author Leon Palaić

**Parameters**

| | |
|---|---|
| *s* | input string |
| *ch* | charachter to be replaced |

**Returns**

new sequence of charachters

Function that replaces charachter wildcard (%,_) ch in string s with repl characters.

**Author**

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic, abstracted by Tomislav Mikulček

**Parameters**

| | |
|---|---|
| *row_root* | beginning of the row that is to be evaluated |
| ∗*expr* | list with the logical expression in postfix notation |

**Returns**

0 if row does not satisfy, 1 if row satisfies expression

#### 7.70.2.3 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 a given string input.

@Author Leon Palaić

**Parameters**

| | |
|---|---|
| *value* | string value that must match regex expression |
| *expression* | POSIX regex expression |
| *checkWildCard* | replaces SQL wildcard to correesponding POSIX regex charachter |
| *sensitive* | case insensitive indicator 1-case sensitive,0- case insensitive |

**Returns**

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

#### 7.70.2.4 AK_check_regex_operator_expression()

```
int AK_check_regex_operator_expression (
            const char * value,
            const char * expression )
```

Function that evaluates regex expression on a given string input.

@Author Leon Palaić

**Parameters**

| | |
|---|---|
| *value* | string value that must match regex expression |
| *expression* | POSIX regex expression |

**Returns**

0 if regex didnt match or sytnax of regex is incorecct 1 if string matches coresponding regex expression

#### 7.70.2.5 AK_expression_check_test()

```
TestResult AK_expression_check_test ( )
```

## 7.71 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 that makes a 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)*
- TestResult AK_op_intersect_test ()

  *Function for intersect operator testing.*

### 7.71.1 Detailed Description

Provides functions for relational intersect operation

### 7.71.2 Function Documentation

#### 7.71.2.1 AK_intersect()

```
int AK_intersect (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes a 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**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the new table |

**Returns**

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

#### 7.71.2.2 AK_op_intersect_test()

```
TestResult AK_op_intersect_test ( )
```

Function for intersect operator testing.

**Author**

Dino Laktašić

**Returns**

No return value

## 7.72 rel/intersect.h File Reference

```
#include "../auxi/test.h"
#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 that makes a 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)*
- TestResult AK_op_intersect_test ()

  *Function for intersect operator testing.*

### 7.72.1 Detailed Description

Provides data structures, functions and defines for relational intersect operation

### 7.72.2 Function Documentation

#### 7.72.2.1 AK_intersect()

```
int AK_intersect (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes a 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**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the new table |

**Returns**

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

#### 7.72.2.2 AK_op_intersect_test()

```
TestResult AK_op_intersect_test ( )
```

Function for intersect operator testing.

**Author**

Dino Laktašić

**Returns**

No return value

## 7.73 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 that makes a 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 that searches the second block and when found matches with the first one makes a join and writes a row to join the tables.*
- void AK_copy_blocks_join (AK_block *tbl1_temp_block, AK_block *tbl2_temp_block, struct list_node *att, char *new_table)

  *Function that 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 that makes a nat_join betwen two tables on some attributes.*
- TestResult AK_op_join_test ()

  *Function for natural join testing.*

### 7.73.1 Detailed Description

Provides functions for relational natural join operation

### 7.73.2 Function Documentation

#### 7.73.2.1 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 that 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**

| | |
|---|---|
| *tbl1_temp_block* | block of the first table |
| *tbl2_temp_block* | block of the second join table |
| *att* | attributes on which we make nat_join |
| *new_table* | name of the nat_join table |

**Returns**

No return value

#### 7.73.2.2 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 that makes a 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**

| | |
|---|---|
| *table_address1* | address of the block of the first table |
| *table_address2* | address of the block of the second table |
| *new_table* | name of the join table |
| *att_root* | ttributes on which we make nat_join |

**Returns**

No return value

#### 7.73.2.3 AK_join()

```
int AK_join (
            char * srcTable1,
            char * srcTable2,
            char * dstTable,
            struct list_node * att )
```

Function that makes a nat_join betwen two tables on some attributes.

**Author**

Matija Novak, updated to work with AK_list and support cacheing by Dino Laktašić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table to join |
| *srcTable2* | name of the second table to join |
| *att* | attributes on which we make nat_join |
| *dstTable* | name of the nat_join table |

**Returns**

if success returns EXIT_SUCCESS

#### 7.73.2.4 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 that searches the second block and when found matches with the first one makes a join and writes a row to join the tables.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *row_root* | - list of values from the first table to be marged with table2 |
| *row_root_insert* | - list of values from the first table to be inserted into nat_join table |
| *temp_block* | - block from the second table to be merged |
| *new_table* | - name of the nat_join table |

**Returns**

No return value

#### 7.73.2.5 AK_op_join_test()

```
TestResult AK_op_join_test ( )
```

Function for natural join testing.

**Author**

Matija Novak

**Returns**

No return value

## 7.74 rel/nat_join.h File Reference

```
#include "../auxi/test.h"
#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 that makes a 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 that searches the second block and when found matches with the first one makes a join and writes a row to join the tables.*
- void AK_copy_blocks_join (AK_block *tbl1_temp_block, AK_block *tbl2_temp_block, struct list_node *att, char *new_table)

  *Function that 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 that makes a nat_join betwen two tables on some attributes.*
- TestResult AK_op_join_test ()

  *Function for natural join testing.*

### 7.74.1 Detailed Description

Header file that provides functions and defines for relational natural join operation

### 7.74.2 Function Documentation

#### 7.74.2.1 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 that 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**

| | |
|---|---|
| *tbl1_temp_block* | block of the first table |
| *tbl2_temp_block* | block of the second join table |
| *att* | attributes on which we make nat_join |
| *new_table* | name of the nat_join table |

**Returns**

No return value

#### 7.74.2.2 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 that makes a 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**

| | |
|---|---|
| *table_address1* | address of the block of the first table |
| *table_address2* | address of the block of the second table |
| *new_table* | name of the join table |
| *att_root* | ttributes on which we make nat_join |

**Returns**

No return value

#### 7.74.2.3 AK_join()

```
int AK_join (
             char * srcTable1,
             char * srcTable2,
             char * dstTable,
             struct list_node * att )
```

Function that makes a nat_join betwen two tables on some attributes.

**Author**

Matija Novak, updated to work with AK_list and support cacheing by Dino Laktašić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table to join |
| *srcTable2* | name of the second table to join |
| *att* | attributes on which we make nat_join |
| *dstTable* | name of the nat_join table |

**Returns**

if success returns EXIT_SUCCESS

#### 7.74.2.4 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 that searches the second block and when found matches with the first one makes a join and writes a row to join the tables.

**Author**

Matija Novak, updated by Dino Laktašić

**Parameters**

| | |
|---|---|
| *row_root* | - list of values from the first table to be marged with table2 |
| *row_root_insert* | - list of values from the first table to be inserted into nat_join table |
| *temp_block* | - block from the second table to be merged |
| *new_table* | - name of the nat_join table |

**Returns**

No return value

#### 7.74.2.5 AK_op_join_test()

```
TestResult AK_op_join_test ( )
```

Function for natural join testing.

**Author**

Matija Novak

**Returns**

No return value

## 7.75 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 that makes the structure of an empty destination table for product operation.*
- void AK_product_procedure (char ∗srcTable1, char ∗srcTable2, char ∗dstTable, AK_header header[MAX_ATTRIBUTES])

  *Functions that iterates trough both tables and concates rows comparing headers and their row values.*
- TestResult AK_op_product_test ()

  *Function for product operator testing, where it is given 2 source table on which product operations are managed.*

### 7.75.1 Detailed Description

Provides functions for relational product operation

## 7.75.2 Function Documentation

### 7.75.2.1 AK_op_product_test()

```
TestResult AK_op_product_test ( )
```

Function for product operator testing, where it is given 2 source table on which product operations are managed.

**Author**

Dino Laktašić, Fabijan Josip Kraljić

**Returns**

Product destination table and number od passed tests.

Test result - number of successful and unsuccessful tests.

### 7.75.2.2 AK_product()

```
int AK_product (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes the structure of an empty destination table for product operation.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the product table |

**Returns**

Created destination table as a result of product operation

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

#### 7.75.2.3 AK_product_procedure()

```
void AK_product_procedure (
            char * srcTable1,
            char * srcTable2,
            char * dstTable,
            AK_header header[MAX_ATTRIBUTES] )
```

Functions that iterates trough both tables and concates rows comparing headers and their row values.

Functions that iterates trough both tables and concates rows. The result is in destination table.

**Author**

Dino Laktašić, Fabijan Josip Kraljić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the product table |

**Returns**

destination table filled with data sized n(rows srcTable1)∗m(rows srcTable2)

**Parameters**

| | |
|---|---|
| *header* | header of product table |

Product procedure Going through one table, and for each row in it, going through another table, and joining rows that way!

## 7.76 rel/product.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../file/files.h"
#include "../auxi/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 that makes the structure of an empty destination table for product operation.*
- void AK_product_procedure (char ∗srcTable1, char ∗srcTable2, char ∗dstTable, AK_header header[MAX_ATTRIBUTES])

  *Functions that iterates trough both tables and concates rows. The result is in destination table.*
- TestResult AK_op_product_test ()

  *Function for product operator testing, where it is given 2 source table on which product operations are managed.*

### 7.76.1 Detailed Description

Header file that provides functions and defines for relational product operation

### 7.76.2 Function Documentation

#### 7.76.2.1 AK_op_product_test()

```
TestResult AK_op_product_test ( )
```

Function for product operator testing, where it is given 2 source table on which product operations are managed.

**Author**

Dino Laktašić, Fabijan Josip Kraljić

**Returns**

Product destination table and number od passed tests.

Test result - number of successful and unsuccessful tests.

#### 7.76.2.2 AK_product()

```
int AK_product (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes the structure of an empty destination table for product operation.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the product table |

**Returns**

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the product table |

**Returns**

Created destination table as a result of product operation

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

#### 7.76.2.3 AK_product_procedure()

```
void AK_product_procedure (
             char * srcTable1,
             char * srcTable2,
             char * dstTable,
             AK_header header[MAX_ATTRIBUTES] )
```

Functions that iterates trough both tables and concates rows. The result is in destination table.

**Author**

Dino Laktašić, Fabijan Josip Kraljić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the product table |
| *header* | header of product table |

Functions that iterates trough both tables and concates rows. The result is in destination table.

**Author**

Dino Laktašić, Fabijan Josip Kraljić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the product table |

**Returns**

destination table filled with data sized n(rows srcTable1)∗m(rows srcTable2)

**Parameters**

| | |
|---|---|
| *header* | header of product table |

Product procedure Going through one table, and for each row in it, going through another table, and joining rows that way!

## 7.77 rel/projection.c File Reference

`#include "projection.h"`
Include dependency graph for projection.c:

### Functions

- void AK_create_block_header (int old_block, char ∗dstTable, struct list_node ∗att)

  *Function that creates a new header for the projection table.*
- char ∗ AK_get_operator (char ∗exp)

  *Function that fetches arithmetic operator from given expression string, determinates given operator so it can be used for aritmetic operations.*
- void AK_remove_substring (char ∗s, const char ∗substring)

  *Function that iterates through given string and removes specified part of that string.*
- int AK_determine_header_type (int firstOperand, int secondOperand)

  *Function that determines the new header type.*
- char ∗ AK_create_header_name (char ∗first, char ∗second, char ∗operator)

  *Function that 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 that copies the data from old table block to the new projection table.*
- char ∗ AK_perform_operation (char ∗op, struct AK_operand ∗firstOperand, struct AK_operand ∗second↩ Operand, int type)

  *Function that performes arithmetics operation depended on given operator.*
- int AK_projection (char ∗srcTable, char ∗dstTable, struct list_node ∗att, struct list_node ∗expr)

  *Function that makes a projection of some table on given attributes.*
- TestResult AK_op_projection_test ()

  *Function for projection operation testing, tests usual projection functionality, projection when it is given aritmetic operation or expresson.*

### 7.77.1 Detailed Description

Provides functions for relational projection operation

### 7.77.2 Function Documentation

#### 7.77.2.1 AK_copy_block_projection()

```
void AK_copy_block_projection (
            AK_block * old_block,
            struct list_node * att,
            char * dstTable,
            struct list_node * expr )
```

Function that copies the data from old table block to the new projection table.

**Author**

Matija Novak, rewritten and optimized by Dino Laktašić to support AK_list

**Parameters**

| | |
|---|---|
| *old_block* | block from which we copy data |
| *dstTable* | name of the new table |
| *att* | list of the attributes which should the projection table contain |
| *expr* | given expression to check |

**Returns**

New projection table that contains all blocks from old table

No return value

#### 7.77.2.2 AK_create_block_header()

```
void AK_create_block_header (
            int old_block,
            char * dstTable,
            struct list_node * att )
```

Function that creates a new header for the projection table.

**Author**

Matija Novak, rewritten and optimized by Dino Laktašić to support AK_list

**Parameters**

| | |
|---|---|
| *old_block_add* | address of the block from which we copy headers we need |
| *dstTable* | name of the new table - destination table |
| *att* | list of the attributes which should the projection table contain |

**Returns**

Newly created header

No return value

#### 7.77.2.3 AK_create_header_name()

```
char* AK_create_header_name (
            char * first,
            char * second,
            char * operator )
```

Function that creates new header name from passed operand names and operator.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *first* | operand name |
| *second* | operand name |
| *operator* | given operator |

**Returns**

Function returns set of characters that represent new header name

Character - new name

#### 7.77.2.4 AK_determine_header_type()

```
int AK_determine_header_type (
            int firstOperand,
            int secondOperand )
```

Function that determines the new header type.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *firstOperand* | operand type |
| *secondOperand* | operand type |

**Returns**

Function returns determinated header type

Integer - type

#### 7.77.2.5 AK_get_operator()

```
char* AK_get_operator (
            char * exp )
```

Function that fetches arithmetic operator from given expression string, determinates given operator so it can be used for aritmetic operations.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *exp* | input expression string |

**Returns**

character - aritmetic operator

character

#### 7.77.2.6 AK_op_projection_test()

```
TestResult AK_op_projection_test ( )
```

Function for projection operation testing, tests usual projection functionality, projection when it is given aritmetic operation or expresson.

**Author**

Dino Laktašić, rewritten and optimized by Irena Ilišević to support ILIKE operator and perform usual projection

**Returns**

Projection tables and number od passed tests

Test result - number of successful and unsuccessful tests

#### 7.77.2.7 AK_perform_operation()

```
char* AK_perform_operation (
            char * op,
            struct AK_operand * firstOperand,
            struct AK_operand * secondOperand,
            int type )
```

Function that performes arithmetics operation depended on given operator.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *firstOperand* | first operand |
| *secondOperand* | second operand |
| *op* | aritmetic operator |
| *type* | type of operand |

**Returns**

result of arithmetic operation

character

#### 7.77.2.8 AK_projection()

```
int AK_projection (
            char * srcTable,
            char * dstTable,
            struct list_node * att,
            struct list_node * expr )
```

Function that makes a projection of some table on given attributes.

**Author**

Matija Novak, rewritten and optimized by Dino Laktašić, now support cacheing

**Parameters**

| | |
|---|---|
| *srcTable* | source table - table on which projection is made |
| *expr* | given expression to check while doing projection |
| *att* | list of atributes on which we make projection |
| *dstTable* | table name for projection table - new table - destination table |

**Returns**

Projection table on given attributes

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

#### 7.77.2.9 AK_remove_substring()

```
void AK_remove_substring (
            char * s,
            const char * substring )
```

Function that iterates through given string and removes specified part of that string.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *s* | input string |
| *substring* | string that needs to be removed |

**Returns**

Cleaned new string

No return value

## 7.78 rel/projection.h File Reference

```
#include "../auxi/test.h"
#include "expression_check.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for projection.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct AK_operand

### Functions

- void AK_create_block_header (int old_block, char *dstTable, struct list_node *att)

  *Function that creates a new header for the projection table.*
- char * AK_get_operator (char *exp)

*Function that fetches arithmetic operator from given expression string, determinates given operator so it can be used for aritmetic operations.*

- void AK_remove_substring (char *s, const char *substring)

  *Function that iterates through given string and removes specified part of that string.*
- int AK_determine_header_type (int firstOperand, int secondOperand)

  *Function that determines the new header type.*
- char * AK_create_header_name (char *first, char *operator, char *second)

  *Function that 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 that copies the data from old table block to the new projection table.*
- char * AK_perform_operation (char *op, struct AK_operand *firstOperand, struct AK_operand *secondOperand, int type)

  *Function that performes arithmetics operation depended on given operator.*
- int AK_projection (char *srcTable, char *dstTable, struct list_node *att, struct list_node *expr)

  *Function that makes a projection of some table on given attributes.*
- TestResult AK_op_projection_test ()

  *Function for projection operation testing, tests usual projection functionality, projection when it is given aritmetic operation or expresson.*

## 7.78.1 Detailed Description

Header file that provides data structures, functions and defines for relational projection operation

## 7.78.2 Function Documentation

### 7.78.2.1 AK_copy_block_projection()

```
void AK_copy_block_projection (
            AK_block * old_block,
            struct list_node * att,
            char * dstTable,
            struct list_node * expr )
```

Function that copies the data from old table block to the new projection table.

**Author**

Matija Novak, rewritten and optimized by Dino Laktašić to support AK_list

**Parameters**

| | |
|---|---|
| *old_block* | block from which we copy data |
| *dstTable* | name of the new table |
| *att* | list of the attributes which should the projection table contain |
| *expr* | given expression to check |

**Returns**

New projection table that contains all blocks from old table

No return value

#### 7.78.2.2 AK_create_block_header()

```
void AK_create_block_header (
            int old_block,
            char * dstTable,
            struct list_node * att )
```

Function that creates a new header for the projection table.

**Author**

Matija Novak, rewritten and optimized by Dino Laktašić to support AK_list

**Parameters**

| | |
|---|---|
| *old_block_add* | address of the block from which we copy headers we need |
| *dstTable* | name of the new table - destination table |
| *att* | list of the attributes which should the projection table contain |

**Returns**

Newly created header

No return value

#### 7.78.2.3 AK_create_header_name()

```
char* AK_create_header_name (
            char * first,
            char * second,
            char * operator )
```

Function that creates new header name from passed operand names and operator.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *first* | operand name |
| *second* | operand name |
| *operator* | given operator |

**Returns**

Function returns set of characters that represent new header name

Character - new name

#### 7.78.2.4 AK_determine_header_type()

```
int AK_determine_header_type (
            int firstOperand,
            int secondOperand )
```

Function that determines the new header type.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *firstOperand* | operand type |
| *secondOperand* | operand type |

**Returns**

Function returns determinated header type

Integer - type

#### 7.78.2.5 AK_get_operator()

```
char* AK_get_operator (
            char * exp )
```

Function that fetches arithmetic operator from given expression string, determinates given operator so it can be used for aritmetic operations.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *exp* | input expression string |

**Returns**

character - aritmetic operator

character

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *exp* | input expression string |

**Returns**

character - aritmetic operator

character

### 7.78.2.6 AK_op_projection_test()

```
TestResult AK_op_projection_test ( )
```

Function for projection operation testing, tests usual projection functionality, projection when it is given aritmetic operation or expresson.

**Author**

Dino Laktašić, rewritten and optimized by Irena Ilišević to support ILIKE operator and perform usual projection

**Returns**

Projection tables and number od passed tests

Test result - number of successful and unsuccessful tests

### 7.78.2.7 AK_perform_operation()

```
char* AK_perform_operation (
            char * op,
            struct AK_operand * firstOperand,
            struct AK_operand * secondOperand,
            int type )
```

Function that performes arithmetics operation depended on given operator.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *firstOperand* | first operand |
| *secondOperand* | second operand |
| *op* | aritmetic operator |
| *type* | type of operand |

**Returns**

result of arithmetic operation

character

### 7.78.2.8 AK_projection()

```
int AK_projection (
            char * srcTable,
            char * dstTable,
            struct list_node * att,
            struct list_node * expr )
```

Function that makes a projection of some table on given attributes.

**Author**

Matija Novak, rewritten and optimized by Dino Laktašić, now support cacheing

**Parameters**

| | |
|---|---|
| *srcTable* | source table - table on which projection is made |
| *expr* | given expression to check while doing projection |
| *att* | list of atributes on which we make projection |
| *dstTable* | table name for projection table - new table - destination table |

**Returns**

Projection table on given attributes

EXIT_SUCCESS if continues succesfuly, when not EXIT_ERROR

### 7.78.2.9 AK_remove_substring()

```
void AK_remove_substring (
            char * s,
            const char * substring )
```

Function that iterates through given string and removes specified part of that string.

**Author**

Leon Palaić

**Parameters**

| | |
|---|---|
| *s* | input string |
| *substring* | string that needs to be removed |

**Returns**

Cleaned new string

No return value

## 7.79 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 that which implements selection.*
- TestResult AK_op_selection_test ()
  *Function for selection operator testing using WHERE clause and operators BETWEEN, AND.*
- TestResult AK_op_selection_test_pattern ()
  *Function for selection operator testing using operators LIKE, ILIKE, SIMILAR TO.*
- int AK_selection_op_rename (char ∗srcTable, char ∗dstTable, struct list_node ∗expr)
  *Function that which implements selection rename operation test.*

### 7.79.1 Detailed Description

Provides functions for relational selection operation

### 7.79.2 Function Documentation

#### 7.79.2.1 AK_op_selection_test()

```
TestResult AK_op_selection_test ( )
```

Function for selection operator testing using WHERE clause and operators BETWEEN, AND.

**Author**

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic

#### 7.79.2.2 AK_op_selection_test_pattern()

```
TestResult AK_op_selection_test_pattern ( )
```

Function for selection operator testing using operators LIKE, ILIKE, SIMILAR TO.

**Author**

Krunoslav Bilić updated by Filip Belinić

#### 7.79.2.3 AK_selection()

```
int AK_selection (
            char * srcTable,
            char * dstTable,
            struct list_node * expr )
```

Function that which implements selection.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗srcTable* | source table name |
| *∗dstTable* | destination table name |
| *∗expr* | list with posfix notation of the logical expression |

**Returns**

EXIT_SUCCESS

#### 7.79.2.4 AK_selection_op_rename()

```
int AK_selection_op_rename (
            char * srcTable,
            char * dstTable,
            struct list_node * expr )
```

Function that which implements selection rename operation test.

**Author**

unknown

**Parameters**

| | |
|---|---|
| ∗*srcTable* | source table name |
| ∗*dstTable* | destination table name |
| ∗*expr* | list with posfix notation of the logical expression |

**Returns**

EXIT_SUCCESS

## 7.80 rel/selection.h File Reference

```
#include "../auxi/test.h"
#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 that which implements selection.*
- TestResult AK_op_selection_test ()

  *Function for selection operator testing using WHERE clause and operators BETWEEN, AND.*
- TestResult AK_op_selection_test_pattern ()

  *Function for selection operator testing using operators LIKE, ILIKE, SIMILAR TO.*

### 7.80.1 Detailed Description

Header file that provides functions and defines for relational selection operation

### 7.80.2 Function Documentation

#### 7.80.2.1 AK_op_selection_test()

```
TestResult AK_op_selection_test ( )
```

Function for selection operator testing using WHERE clause and operators BETWEEN, AND.

**Author**

Matija Šestak, updated by Dino Laktašić,Nikola Miljancic

#### 7.80.2.2 AK_op_selection_test_pattern()

```
TestResult AK_op_selection_test_pattern ( )
```

Function for selection operator testing using operators LIKE, ILIKE, SIMILAR TO.

**Author**

Krunoslav Bilić updated by Filip Belinić

#### 7.80.2.3 AK_selection()

```
int AK_selection (
            char * srcTable,
            char * dstTable,
            struct list_node * expr )
```

Function that which implements selection.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| ∗*srcTable* | source table name |
| ∗*dstTable* | destination table name |
| ∗*expr* | list with posfix notation of the logical expression |

**Returns**

EXIT_SUCCESS

## 7.81 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 that creates a 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 that 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 that creates a theta join betwen two tables on specified conditions. Names of the attibutes 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.*
- TestResult AK_op_theta_join_test ()

  *Function for testing the theta join.*

## 7.81.1 Detailed Description

Provides functions for relational theta join operation

## 7.81.2 Function Documentation

### 7.81.2.1 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 that iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.

**Author**

Tomislav Mikulček

**Parameters**

| | |
|---|---|
| *tbl1_temp_block* | block of the first table |
| *tbl2_temp_block* | block of the second join table |
| *tbl1_num_att* | number of attributes in the first table |
| *tbl2_num_att* | number of attributes in the second table |
| *constraints* | list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation |
| *new_table* | name of the theta_join table |

**Returns**

No return value

#### 7.81.2.2 AK_create_theta_join_header()

```
int AK_create_theta_join_header (
            char * srcTable1,
            char * srcTable2,
            char * new_table )
```

Function that creates a header of the new table for theta join.

**Author**

Tomislav Mikulček

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *new_table* | name of the destination table |

**Returns**

EXIT_SUCCESS if the header was successfully created and EXIT_ERROR if the renamed headers are too long

#### 7.81.2.3 AK_op_theta_join_test()

```
TestResult AK_op_theta_join_test ( )
```

Function for testing the theta join.

**Author**

Tomislav Mikulček

**Returns**

No return value

#### 7.81.2.4 AK_theta_join()

```
int AK_theta_join (
            char * srcTable1,
            char * srcTable2,
            char * dstTable,
            struct list_node * constraints )
```

Function that creates a theta join betwen two tables on specified conditions. Names of the attibutes 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.

Function that creates a theta join betwen two tables on specified conditions.

**Author**

Tomislav Mikulček,updated by Nikola Miljancic

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table to join |
| *srcTable2* | name of the second table to join |
| *constraints* | list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation |
| *dstTable* | name of the theta join table |

**Returns**

if successful returns EXIT_SUCCESS and EXIT_ERROR otherwise

## 7.82 rel/theta_join.h File Reference

```
#include "../auxi/test.h"
#include "expression_check.h"
#include "../file/fileio.h"
#include "../auxi/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 that creates a theta join betwen two tables on specified conditions.*
- int AK_create_theta_join_header (char *srcTable1, char *srcTable2, char *new_table)

  *Function that creates a 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 that iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.*
- TestResult AK_op_theta_join_test ()

  *Function for testing the theta join.*

### 7.82.1 Detailed Description

Header file that provides functions and defines for theta-join

### 7.82.2 Function Documentation

#### 7.82.2.1 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 that iterates through blocks of the two tables and copies the rows which pass the constraint check into the new table.

**Author**

Tomislav Mikulček

**Parameters**

| | |
|---|---|
| *tbl1_temp_block* | block of the first table |
| *tbl2_temp_block* | block of the second join table |
| *tbl1_num_att* | number of attributes in the first table |
| *tbl2_num_att* | number of attributes in the second table |
| *constraints* | list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation |
| *new_table* | name of the theta_join table |

**Returns**

No return value

#### 7.82.2.2 AK_create_theta_join_header()

```
int AK_create_theta_join_header (
            char * srcTable1,
            char * srcTable2,
            char * new_table )
```

Function that creates a header of the new table for theta join.

**Author**

Tomislav Mikulček

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *new_table* | name of the destination table |

**Returns**

EXIT_SUCCESS if the header was successfully created and EXIT_ERROR if the renamed headers are too long

#### 7.82.2.3 AK_op_theta_join_test()

```
TestResult AK_op_theta_join_test ( )
```

Function for testing the theta join.

**Author**

Tomislav Mikulček

**Returns**

No return value

#### 7.82.2.4 AK_theta_join()

```
int AK_theta_join (
            char * srcTable1,
            char * srcTable2,
            char * dstTable,
            struct list_node * constraints )
```

Function that creates a theta join betwen two tables on specified conditions.

**Author**

Tomislav Mikulček,updated by Nikola Miljancic

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table to join |
| *srcTable2* | name of the second table to join |
| *constraints* | list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation |
| *dstTable* | name of the theta join table |

**Returns**

if successful returns EXIT_SUCCESS and EXIT_ERROR otherwise

Function that creates a theta join betwen two tables on specified conditions.

**Author**

Tomislav Mikulček,updated by Nikola Miljancic

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table to join |
| *srcTable2* | name of the second table to join |
| *constraints* | list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation |
| *dstTable* | name of the theta join table |

**Returns**

if successful returns EXIT_SUCCESS and EXIT_ERROR otherwise

## 7.83 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 that makes a union of two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)*
- TestResult AK_op_union_test ()

  *Function for union operator testing.*

### 7.83.1 Detailed Description

Provides functions for relational union operation

### 7.83.2 Function Documentation

#### 7.83.2.1 AK_op_union_test()

```
TestResult AK_op_union_test ( )
```

Function for union operator testing.

**Author**

Dino Laktašić

**Returns**

No return value

#### 7.83.2.2 AK_union()

```
int AK_union (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes a union of two tables. Union is implemented for working with multiple sets of data, i.e. duplicate tuples can be written in same table (union)

Function that makes a union of two tables.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the new table |

**Returns**

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

## 7.84 rel/union.h File Reference

```
#include "../auxi/test.h"
```

```
#include "../file/table.h"
#include "../file/fileio.h"
#include "../auxi/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 that makes a union of two tables.*
- TestResult AK_op_union_test ()

  *Function for union operator testing.*

### 7.84.1 Detailed Description

Header file that provides functions and defines for relational union operation

### 7.84.2 Function Documentation

#### 7.84.2.1 AK_op_union_test()

```
TestResult AK_op_union_test ( )
```

Function for union operator testing.

**Author**

Dino Laktašić

**Returns**

No return value

#### 7.84.2.2 AK_union()

```
int AK_union (
            char * srcTable1,
            char * srcTable2,
            char * dstTable )
```

Function that makes a union of two tables.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the new table |

**Returns**

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

Function that makes a union of two tables.

**Author**

Dino Laktašić

**Parameters**

| | |
|---|---|
| *srcTable1* | name of the first table |
| *srcTable2* | name of the second table |
| *dstTable* | name of the new table |

**Returns**

if success returns EXIT_SUCCESS, else returns EXIT_ERROR

## 7.85 sql/command.c File Reference

`#include "command.h"`
Include dependency graph for command.c:

### Functions

- int AK_command (command *commands, int commandNum)

  *Function for executing given commands (SELECT, UPDATE, DELETE AND INSERT)*
- TestResult AK_test_command ()

  *Function for testing commands.*

### 7.85.1 Detailed Description

TODO: Description

### 7.85.2 Function Documentation

#### 7.85.2.1 AK_command()

```
int AK_command (
            command * commands,
            int commandNum )
```

Function for executing given commands (SELECT, UPDATE, DELETE AND INSERT)

**Author**

Mario Kolmacic updated by Ivan Pusic and Tomislav Ilisevic

**Parameters**

| | |
|---|---|
| *commands* | Commands array to execute |
| *commandNum* | Number of commands in array |

**Returns**

ERROR_EXIT only if command can't be executed returns EXIT_ERROR

#### 7.85.2.2 AK_test_command()

```
TestResult AK_test_command ( )
```

Function for testing commands.

**Author**

Unknown, updated by Tomislav Ilisevic

## 7.86 sql/command.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../rel/selection.h"
#include "../auxi/mempro.h"
```

Include dependency graph for command.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct AK_command_struct

### Typedefs

- typedef struct AK_command_struct command

## Functions

- int AK_command (command *komande, int brojkomandi)

  *Function for executing given commands (SELECT, UPDATE, DELETE AND INSERT)*
- TestResult AK_test_command ()

  *Function for testing commands.*

### 7.86.1 Detailed Description

Header file that provides data structures, functions and defines for command.c

### 7.86.2 Typedef Documentation

#### 7.86.2.1 command

```
typedef struct AK_command_struct command
```

### 7.86.3 Function Documentation

#### 7.86.3.1 AK_command()

```
int AK_command (
            command * commands,
            int commandNum )
```

Function for executing given commands (SELECT, UPDATE, DELETE AND INSERT)

**Author**

Mario Kolmacic updated by Ivan Pusic and Tomislav Ilisevic

**Parameters**

| | |
|---|---|
| *commands* | Commands array to execute |
| *commandNum* | Number of commands in array |

**Returns**

ERROR_EXIT only if command can't be executed returns EXIT_ERROR

#### 7.86.3.2 AK_test_command()

```
TestResult AK_test_command ( )
```

Function for testing commands.

**Author**

Unknown, updated by Tomislav Ilisevic

## 7.87 sql/cs/between.c File Reference

```
#include "between.h"
```

Include dependency graph for between.c:

### Functions

- int AK_find_table_address (char *_systemTableName)

  *Function that returns system tables addresses by name.*
- void AK_set_constraint_between (char *tableName, char *constraintName, char *attName, char *startValue, char *endValue)

  *Function that sets between constraints on particular attribute, string constraint should be writen 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.*
- int AK_read_constraint_between (char *tableName, char *newValue, char *attNamePar)

  *Function that checks if the given value is between lower and upper bounds of the "between" constraint.*
- void AK_print_constraints (char *tableName)

  *Function for printing tables.*
- int AK_delete_constraint_between (char *tableName, char *constraintNamePar)

  *Function for deleting specific between constraint.*
- TestResult AK_constraint_between_test ()

  *Function that tests the functionality of implemented between constraint.*

### 7.87.1 Detailed Description

Provides functions for between constaint

### 7.87.2 Function Documentation

#### 7.87.2.1 AK_constraint_between_test()

```
TestResult AK_constraint_between_test ( )
```

Function that tests the functionality of implemented between constraint.

**Author**

Saša Vukšić, updated by Mislav Jurinić, updated by Blaž Rajič

**Returns**

No return value

#### 7.87.2.2 AK_delete_constraint_between()

```
int AK_delete_constraint_between (
            char * tableName,
            char * constraintNamePar )
```

Function for deleting specific between constraint.

**Author**

Maja Vračan

**Parameters**

| | |
|---|---|
| *tableName* | name of table on which constraint refers |
| *attName* | name of attribute on which constraint is declared |
| *constraintName* | name of constraint |

**Returns**

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

#### 7.87.2.3 AK_find_table_address()

```
int AK_find_table_address (
            char * _systemTableName )
```

Function that returns system tables addresses by name.

**Author**

Mislav Jurinić

**Parameters**

| | |
|---|---|
| *_systemTableName* | table name |

**Returns**

int

#### 7.87.2.4 AK_print_constraints()

```
void AK_print_constraints (
             char * tableName )
```

Function for printing tables.

**Author**

Maja Vračan

**Parameters**

| | |
|---|---|
| *tableName* | name of table |

#### 7.87.2.5 AK_read_constraint_between()

```
int AK_read_constraint_between (
             char * tableName,
             char * newValue,
             char * attNamePar )
```

Function that checks if the given value is between lower and upper bounds of the "between" constraint.

**Author**

Saša Vukšić, updated by Mislav Jurinić, updated by Blaž Rajič

**Parameters**

| | |
|---|---|
| *tableName* | table name |
| *newValue* | value we want to insert |
| *attNamePar* | attribute name in table |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.87.2.6 AK_set_constraint_between()

```
void AK_set_constraint_between (
            char * tableName,
            char * constraintName,
            char * attName,
            char * startValue,
            char * endValue )
```

Function that sets between constraints on particular attribute, string constraint should be writen 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.

Function that sets between constraints on particular attribute, string constraint should be writen in lowercase.

**Author**

Saša Vukšić, updated by Mislav Jurinić, updated by Blaž Rajič

**Parameters**

| | |
|---|---|
| *tableName* | table name |
| *constraintName* | name of constraint |
| *attName* | name of attribute |
| *startValue* | initial constraint |
| *endValue* | final constraint |

**Returns**

No return value

## 7.88 sql/cs/between.h File Reference

```
#include "../../auxi/test.h"
#include "../../mm/memoman.h"
#include "../../file/id.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for between.h: This graph shows which files directly or indirectly include this file:

### Functions

- int AK_find_table_address (char *_systemTableName)

*Function that returns system tables addresses by name.*

- void AK_set_constraint_between (char *tableName, char *constraintName, char *attName, char *startValue, char *endValue)

  *Function that sets between constraints on particular attribute, string constraint should be writen in lowercase.*

- int AK_read_constraint_between (char *tableName, char *newValue, char *attNamePar)

  *Function that checks if the given value is between lower and upper bounds of the "between" constraint.*

- int AK_delete_constraint_between (char *tableName, char constraintName[])

  *Function for deleting specific between constraint.*

- TestResult AK_constraint_between_test ()

  *Function that tests the functionality of implemented between constraint.*

## 7.88.1 Detailed Description

Header file that provides functions and defines for between constaint

## 7.88.2 Function Documentation

### 7.88.2.1 AK_constraint_between_test()

```
TestResult AK_constraint_between_test ( )
```

Function that tests the functionality of implemented between constraint.

**Author**

Saša Vukšić, updated by Mislav Jurinić, updated by Blaž Rajič

**Returns**

No return value

### 7.88.2.2 AK_delete_constraint_between()

```
int AK_delete_constraint_between (
            char * tableName,
            char constraintName[] )
```

Function for deleting specific between constraint.

**Author**

Maja Vračan, updated by Blaž Rajič

**Parameters**

| | |
|---|---|
| *tableName* | name of table on which constraint refers |
| *attName* | name of attribute on which constraint is declared |
| *constraintName* | name of constraint |

**Returns**

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

#### 7.88.2.3 AK_find_table_address()

```
int AK_find_table_address (
            char * _systemTableName )
```

Function that returns system tables addresses by name.

**Author**

Mislav Jurinić

**Parameters**

| | |
|---|---|
| *_systemTableName* | table name |

**Returns**

int

#### 7.88.2.4 AK_read_constraint_between()

```
int AK_read_constraint_between (
            char * tableName,
            char * newValue,
            char * attNamePar )
```

Function that checks if the given value is between lower and upper bounds of the "between" constraint.

**Author**

Saša Vukšić, updated by Mislav Jurinić

**Parameters**

| | |
|---|---|
| *tableName* | table name |
| *newValue* | value we want to insert |
| *attNamePar* | attribute name |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

**Author**

Saša Vukšić, updated by Mislav Jurinić, updated by Blaž Rajič

**Parameters**

| | |
|---|---|
| *tableName* | table name |
| *newValue* | value we want to insert |
| *attNamePar* | attribute name in table |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

### 7.88.2.5 AK_set_constraint_between()

```
void AK_set_constraint_between (
             char * tableName,
             char * constraintName,
             char * attName,
             char * startValue,
             char * endValue )
```

Function that sets between constraints on particular attribute, string constraint should be writen in lowercase.

**Author**

Saša Vukšić, updated by Mislav Jurinić

**Parameters**

| | |
|---|---|
| *tableName* | table name |
| *constraintName* | name of constraint |
| *attName* | name of attribute |
| *startValue* | initial constraint |
| *endValue* | final constraint |

**Returns**

No return value

Function that sets between constraints on particular attribute, string constraint should be writen in lowercase.

**Author**

Saša Vukšić, updated by Mislav Jurinić, updated by Blaž Rajič

**Parameters**

| | |
|---|---|
| *tableName* | table name |
| *constraintName* | name of constraint |
| *attName* | name of attribute |
| *startValue* | initial constraint |
| *endValue* | final constraint |

**Returns**

No return value

## 7.89 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)

  *Function that for a 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)

  *Function that adds a new "check" constraint into the system table.*
- int AK_check_constraint (char ∗table, char ∗attribute, void ∗value)

  *Function that verifies if the value we want to insert satisfies the "check" constraint.*
- int AK_delete_check_constraint (char ∗tableName, char ∗constraintName)

  *Function that deletes existing check constraint.*
- TestResult AK_check_constraint_test ()

  *Test function for "check" constraint.*

### 7.89.1 Detailed Description

Check constraint implementation file.

## 7.89.2 Function Documentation

### 7.89.2.1 AK_check_constraint()

```
int AK_check_constraint (
            char * table,
            char * attribute,
            void * value )
```

Function that verifies if the value we want to insert satisfies the "check" constraint.

**Author**

Mislav Jurinić

**Parameters**

| | |
|---|---|
| *table* | target table name |
| *attribute* | target attribute name |
| *value* | data we want to insert |

**Returns**

1 - result, 0 - failure

### 7.89.2.2 AK_check_constraint_test()

```
TestResult AK_check_constraint_test ( )
```

Test function for "check" constraint.

**Author**

Mislav Jurinić, updated by Bruno Pilošta

**Returns**

void

#### 7.89.2.3 AK_delete_check_constraint()

```
int AK_delete_check_constraint (
             char * tableName,
             char * constraintName )
```

Function that deletes existing check constraint.

Function that verifies if the value we want to insert satisfies the "check" constraint.

**Author**

Bruno Pilošta

**Parameters**

| | |
|---|---|
| *tableName* | System table where constraint will be deleted from |
| *constraintName* | Name of the constraint that will be deleted |

**Returns**

1 - result, 0 - failure

#### 7.89.2.4 AK_set_check_constraint()

```
int AK_set_check_constraint (
             char * table_name,
             char * constraint_name,
             char * attribute_name,
             char * condition,
             int type,
             void * value )
```

Function that adds a new "check" constraint into the system table.

**Author**

Mislav Jurinić

**Parameters**

| | |
|---|---|
| *table_name* | target table for "check" constraint evaluation |
| *constraint_name* | new "check" constraint name that will be visible in the system table |
| *attribute_name* | target attribute for "check" constraint evaluation |
| *condition* | logical operator ['<', '>', '!=', ...] |
| *type* | data type [int, float, varchar, datetime, ...] |
| *value* | condition to be set |

**Returns**

1 - result, 0 - failure

#### 7.89.2.5 condition_passed()

```
int condition_passed (
            char * condition,
            int type,
            void * value,
            void * row_data )
```

Function that for a given value, checks if it satisfies the "check" constraint.

**Author**

Mislav Jurinić

**Parameters**

| | |
|---|---|
| *condition* | logical operator ['<', '>', '!=', ...] |
| *type* | data type [int, float, varchar, datetime, ...] |
| *value* | condition to be set |
| *row_data* | data in table |

**Returns**

1 - result, 0 - failure

## 7.90 sql/cs/check_constraint.h File Reference

```
#include "../../auxi/test.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../rel/expression_check.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for check_constraint.h: This graph shows which files directly or indirectly include this file:

### Functions

- int condition_passed (char ∗condition, int type, void ∗value, void ∗row_data)

  *Function that for a 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)

  *Function that adds a new "check" constraint into the system table.*
- int AK_delete_check_constraint (char ∗tableName, char ∗constraintName)

  *Function that verifies if the value we want to insert satisfies the "check" constraint.*
- TestResult AK_check_constraint_test ()

  *Test function for "check" constraint.*

### 7.90.1 Detailed Description

Header file that provides functions and defines for check constraint

### 7.90.2 Function Documentation

#### 7.90.2.1 AK_check_constraint_test()

```
TestResult AK_check_constraint_test ( )
```

Test function for "check" constraint.

**Author**

Mislav Jurinić, updated by Bruno Pilošta

**Returns**

void

#### 7.90.2.2 AK_delete_check_constraint()

```
int AK_delete_check_constraint (
            char * tableName,
            char * constraintName )
```

Function that verifies if the value we want to insert satisfies the "check" constraint.

**Author**

Mislav Jurinić

**Parameters**

| | |
|---|---|
| *table* | target table name |
| *attribute* | target attribute name |
| *value* | data we want to insert |

**Returns**

1 - result, 0 - failure

Function that verifies if the value we want to insert satisfies the "check" constraint.

**Author**

Bruno Pilošta

**Parameters**

| | |
|---|---|
| *tableName* | System table where constraint will be deleted from |
| *constraintName* | Name of the constraint that will be deleted |

**Returns**

1 - result, 0 - failure

#### 7.90.2.3 AK_set_check_constraint()

```
int AK_set_check_constraint (
            char * table_name,
            char * constraint_name,
            char * attribute_name,
            char * condition,
            int type,
            void * value )
```

Function that adds a new "check" constraint into the system table.

**Author**

Mislav Jurinić

**Parameters**

| | |
|---|---|
| *table_name* | target table for "check" constraint evaluation |
| *constraint_name* | new "check" constraint name that will be visible in the system table |
| *attribute_name* | target attribute for "check" constraint evaluation |
| *condition* | logical operator ['<', '>', '!=', ...] |
| *type* | data type [int, float, varchar, datetime, ...] |
| *value* | condition to be set |

**Returns**

1 - result, 0 - failure

#### 7.90.2.4 condition_passed()

```
int condition_passed (
            char * condition,
```

```
            int type,
            void * value,
            void * row_data )
```

Function that for a given value, checks if it satisfies the "check" constraint.

**Author**

Mislav Jurinić

**Parameters**

| | |
|---|---|
| *condition* | logical operator ['<', '>', '!=', ...] |
| *type* | data type [int, float, varchar, datetime, ...] |
| *value* | condition to be set |
| *row_data* | data in table |

**Returns**

1 - result, 0 - failure

## 7.91 sql/cs/constraint_names.c File Reference

`#include "constraint_names.h"`
Include dependency graph for constraint_names.c:

### Functions

- int AK_check_constraint_name (char ∗constraintName)
  *Function that checks if constraint name would be unique in database.*
- TestResult AK_constraint_names_test ()
  *Function that tests if constraint name would be unique in database.*

### 7.91.1 Detailed Description

Provides functions for checking if constraint name is unique in database

### 7.91.2 Function Documentation

#### 7.91.2.1 AK_check_constraint_name()

```
int AK_check_constraint_name (
            char * constraintName )
```

Function that checks if constraint name would be unique in database.

**Author**

Nenad Makar, updated by Matej Lipovača

**Parameters**

| | |
|---|---|
| *char* | constraintName name which you want to give to constraint which you are trying to create |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

Updated by Matej Lipovača Added other constraint names from catalog, aswell in "constants.h"

#### 7.91.2.2 AK_constraint_names_test()

```
TestResult AK_constraint_names_test ( )
```

Function that tests if constraint name would be unique in database.

**Author**

Nenad Makar

**Returns**

No return value

## 7.92 sql/cs/constraint_names.h File Reference

```
#include "../../auxi/test.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../auxi/mempro.h"
```

Include dependency graph for constraint_names.h: This graph shows which files directly or indirectly include this file:

### Functions

- int AK_check_constraint_name (char *constraintName)

  *Function that checks if constraint name would be unique in database.*
- TestResult AK_constraint_names_test ()

  *Function that tests if constraint name would be unique in database.*

### 7.92.1 Detailed Description

Header file that provides functions and defines for checking if constraint name is unique in database

### 7.92.2 Function Documentation

#### 7.92.2.1 AK_check_constraint_name()

```
int AK_check_constraint_name (
            char * constraintName )
```

Function that checks if constraint name would be unique in database.

**Author**

Nenad Makar, updated by Mislav Jurinić

**Parameters**

| | |
|---|---|
| *char* | constraintName name which you want to give to constraint which you are trying to create |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

**Author**

Nenad Makar, updated by Matej Lipovača

**Parameters**

| | |
|---|---|
| *char* | constraintName name which you want to give to constraint which you are trying to create |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

Updated by Matej Lipovača Added other constraint names from catalog, aswell in "constants.h"

#### 7.92.2.2 AK_constraint_names_test()

```
TestResult AK_constraint_names_test ( )
```

Function that tests if constraint name would be unique in database.

**Author**

Nenad Makar

**Returns**

No return value

# 7.93 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 an attribute.*
- int AK_check_constraint_not_null (char *tableName, char *attName, char *constraintName)
  *Function that checks if constraint name is unique and in violation of NOT NULL constraint.*
- int AK_read_constraint_not_null (char *tableName, char *attName, char *newValue)
  *Function checks if NOT NULL constraint is already set.*
- int AK_delete_constraint_not_null (char *tableName, char *constraintName)
  *Function for deleting not null constraints.*
- TestResult AK_nnull_constraint_test ()
  *Function for testing NOT NULL constraint.*

### 7.93.1 Detailed Description

Provides functions for not null constraint

### 7.93.2 Function Documentation

#### 7.93.2.1 AK_check_constraint_not_null()

```
int AK_check_constraint_not_null (
            char * tableName,
            char * attName,
            char * constraintName )
```

Function that checks if constraint name is unique and in violation of NOT NULL constraint.

**Author**

Saša Vukšić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char** | tableName name of table |
| *char** | attName name of attribute |
| *char** | constraintName name of constraint |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.93.2.2 AK_delete_constraint_not_null()

```
int AK_delete_constraint_not_null (
            char * tableName,
            char * constraintName )
```

Function for deleting not null constraints.

**Author**

Bruno Pilošta

**Parameters**

| | |
|---|---|
| *tableName* | System table where constraint will be deleted from |
| *constraintName* | Name of constraint that will be deleted |

**Returns**

EXIT_SUCCESS if the constraint is deleted, EXIT_ERROR otherwise

#### 7.93.2.3 AK_nnull_constraint_test()

```
TestResult AK_nnull_constraint_test ( )
```

Function for testing NOT NULL constraint.

**Author**

Saša Vukšić, updated by Nenad Makar

**Returns**

No return value

#### 7.93.2.4 AK_read_constraint_not_null()

```
int AK_read_constraint_not_null (
            char * tableName,
            char * attName,
            char * newValue )
```

Function checks if NOT NULL constraint is already set.

**Author**

Saša Vukšić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char∗* | tableName name of table |
| *char∗* | attName name of attribute |
| *char∗* | newValue new value |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.93.2.5 AK_set_constraint_not_null()

```
int AK_set_constraint_not_null (
            char * tableName,
            char * attName,
            char * constraintName )
```

Function that sets NOT NULL constraint on an attribute.

**Author**

Saša Vukšić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char∗* | tableName name of table |
| *char∗* | attName name of attribute |
| *char∗* | constraintName name of constraint |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

## 7.94 sql/cs/nnull.h File Reference

```
#include "../../auxi/test.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../auxi/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 an attribute.*

- int AK_read_constraint_not_null (char *tableName, char *attName, char *newValue)

  *Function checks if NOT NULL constraint is already set.*
- int AK_check_constraint_not_null (char *tableName, char *attName, char *newValue)

  *Function that checks if constraint name is unique and in violation of NOT NULL constraint.*
- int AK_delete_constraint_not_null (char *tableName, char constraintName[ ])

  *Function for deleting specific not null constraint.*
- TestResult AK_nnull_constraint_test ()

  *Function for testing NOT NULL constraint.*

## 7.94.1 Detailed Description

Header file that provides functions and defines for not null constraint

## 7.94.2 Function Documentation

### 7.94.2.1 AK_check_constraint_not_null()

```
int AK_check_constraint_not_null (
            char * tableName,
            char * attName,
            char * constraintName )
```

Function that checks if constraint name is unique and in violation of NOT NULL constraint.

**Author**

Saša Vukšić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char** | tableName name of table |
| *char** | attName name of attribute |
| *char** | constraintName name of constraint |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

### 7.94.2.2 AK_delete_constraint_not_null()

```
int AK_delete_constraint_not_null (
            char * tableName,
            char constraintName[] )
```

Function for deleting specific not null constraint.

**Author**

Maja Vračan

**Parameters**

| | |
|---|---|
| *tableName* | name of table on which constraint refers |
| *attName* | name of attribute on which constraint is declared |
| *constraintName* | name of constraint |

**Returns**

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

#### 7.94.2.3 AK_nnull_constraint_test()

```
TestResult AK_nnull_constraint_test ( )
```

Function for testing NOT NULL constraint.

**Author**

Saša Vukšić, updated by Nenad Makar

**Returns**

No return value

#### 7.94.2.4 AK_read_constraint_not_null()

```
int AK_read_constraint_not_null (
            char * tableName,
            char * attName,
            char * newValue )
```

Function checks if NOT NULL constraint is already set.

**Author**

Saša Vukšić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char∗* | tableName name of table |
| *char∗* | attName name of attribute |
| *char∗* | newValue new value |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.94.2.5 AK_set_constraint_not_null()

```
int AK_set_constraint_not_null (
            char * tableName,
            char * attName,
            char * constraintName )
```

Function that sets NOT NULL constraint on an attribute.

**Author**

Saša Vukšić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char∗* | tableName name of table |
| *char∗* | attName name of attribute |
| *char∗* | constraintName name of constraint |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

## 7.95 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 that 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 that reads a reference entry from system table.*
- int AK_reference_check_attribute (char ∗tableName, char ∗attribute, char ∗value)

  *Function that checks referential integrity for one attribute.*
- int AK_reference_check_if_update_needed (struct list_node ∗lista, int action)

  *Funcction 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 that checks for a REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.*

- int AK_reference_update (struct list_node *lista, int action)

  *Function that updates child table entries according to ongoing update of parent table entries.*
- int AK_reference_check_entry (struct list_node *lista)

  *Function that checks a new entry for referential integrity.*
- TestResult AK_reference_test ()

  *Function for testing referential integrity.*

## 7.95.1 Detailed Description

Provides functions for referential integrity

## 7.95.2 Function Documentation

### 7.95.2.1 AK_add_reference()

```
int AK_add_reference (
            char * childTable,
            char * childAttNames[],
            char * parentTable,
            char * parentAttNames[],
            int attNum,
            char * constraintName,
            int type )
```

Function that 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**

| | |
|---|---|
| *name* | of the child table |
| *array* | of child table attribute names (foreign key attributes) |
| *name* | of the parent table |
| *array* | of parent table attribute names (primary key attributes) |
| *number* | of attributes in foreign key |
| *name* | of the constraint |
| *type* | of the constraint, constants defined in 'reference.h' |

**Returns**

EXIT_SUCCESS

#### 7.95.2.2 AK_get_reference()

```
AK_ref_item AK_get_reference (
             char * tableName,
             char * constraintName )
```

Function that reads a reference entry from system table.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *name* | of the table with reference (with foreign key) |
| *name* | of the reference constraint |

**Returns**

AK_ref_item object with all neccessary information about the reference

#### 7.95.2.3 AK_reference_check_attribute()

```
int AK_reference_check_attribute (
             char * tableName,
             char * attribute,
             char * value )
```

Function that checks referential integrity for one attribute.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *child* | table name |
| *attribute* | name (foreign key attribute) |
| *value* | of the attribute we're checking |

**Returns**

EXIT ERROR if check failed, EXIT_SUCCESS if referential integrity is ok

#### 7.95.2.4 AK_reference_check_entry()

```
int AK_reference_check_entry (
            struct list_node * lista )
```

Function that checks a new entry for referential integrity.

**Author**

Dejan Franković

**Parameters**

| | |
|---|---|
| *list* | of elements for insert row |

**Returns**

EXIT_SUCCESS if referential integrity is ok, EXIT_ERROR if it is compromised

#### 7.95.2.5 AK_reference_check_if_update_needed()

```
int AK_reference_check_if_update_needed (
            struct list_node * lista,
            int action )
```

Funcction that quickly checks if there are any referential constraints that should be applied on a given list of changes.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *list* | of elements for update |
| *is* | action UPDATE or DELETE ? |

**Returns**

EXIT_SUCCESS if update is needed, EXIT_ERROR if not

#### 7.95.2.6 AK_reference_check_restricion()

```
int AK_reference_check_restricion (
            struct list_node * lista,
            int action )
```

Function that checks for a REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.

**Author**

Dejan Franković

**Parameters**

| | |
|---|---|
| *list* | of elements for update |
| *is* | action UPDATE or DELETE? |

**Returns**

EXIT_SUCCESS if there is no restriction on this action, EXIT_ERROR if there is

#### 7.95.2.7 AK_reference_test()

```
TestResult AK_reference_test ( )
```

Function for testing referential integrity.

**Author**

Dejan Franković

**Returns**

No return value

#### 7.95.2.8 AK_reference_update()

```
int AK_reference_update (
            struct list_node * lista,
            int action )
```

Function that updates child table entries according to ongoing update of parent table entries.

**Author**

Dejan Franković

**Parameters**

| | |
|---|---|
| *list* | of elements for update |
| *is* | action UPDATE or DELETE ? |

**Returns**

EXIT_SUCCESS

## 7.96 sql/cs/reference.h File Reference

```
#include "../../auxi/test.h"
#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 that 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 that reads a reference entry from system table.*
- int AK_reference_check_attribute (char *tableName, char *attribute, char *value)

  *Function that checks referential integrity for one attribute.*
- int AK_reference_check_if_update_needed (struct list_node *lista, int action)

  *Funcction 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 that checks for a REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.*
- int AK_reference_update (struct list_node *lista, int action)

  *Function that updates child table entries according to ongoing update of parent table entries.*
- int AK_reference_check_entry (struct list_node *lista)

  *Function that checks a new entry for referential integrity.*
- TestResult 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)

  *Used to add a new element after some element, to insert on first place give list as before element. It calls function AK_Insert_New_Element_For_Update.*
- void AK_Update_Existing_Element (int newtype, void *data, char *table, char *attribute_name, struct list_node *ElementBefore)

  *Used to add a constraint attribute which will define what element gets updated when the operation is executed.*
- 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.*
- int AK_selection (char *srcTable, char *dstTable, struct list_node *expr)

  *Function that 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)

  *!! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. 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.*
- 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 that initializes a new segment and writes its start and finish address in system catalog table. For creting new table, index, temporary table, etc. call this function.*

### 7.96.1 Detailed Description

đ Provides data structures, functions and defines for referential integrity

## 7.96.2 Macro Definition Documentation

### 7.96.2.1 MAX_CHILD_CONSTRAINTS

```
#define MAX_CHILD_CONSTRAINTS 20
```

Constant declaring maximum number of child constraints.

### 7.96.2.2 MAX_REFERENCE_ATTRIBUTES

```
#define MAX_REFERENCE_ATTRIBUTES 10
```

Constant declaring maximum number of reference attributes.

### 7.96.2.3 REF_TYPE_CASCADE

```
#define REF_TYPE_CASCADE 3
```

### 7.96.2.4 REF_TYPE_NO_ACTION

```
#define REF_TYPE_NO_ACTION 2
```

Constant declaring no action reference type.

Constant declaring cascade reference type.

### 7.96.2.5 REF_TYPE_NONE

```
#define REF_TYPE_NONE -1
```

Constant declaring none reference type.

### 7.96.2.6 REF_TYPE_RESTRICT

```
#define REF_TYPE_RESTRICT 4
```

Constant declaring restrict reference type.

#### 7.96.2.7 REF_TYPE_SET_DEFAULT

```
#define REF_TYPE_SET_DEFAULT 5
```

Constant declaring set default reference type.

#### 7.96.2.8 REF_TYPE_SET_NULL

```
#define REF_TYPE_SET_NULL 1
```

Constant declaring set null reference type.

### 7.96.3 Function Documentation

#### 7.96.3.1 AK_add_reference()

```
int AK_add_reference (
            char * childTable,
            char * childAttNames[],
            char * parentTable,
            char * parentAttNames[],
            int attNum,
            char * constraintName,
            int type )
```

Function that 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**

| | |
|---|---|
| *name* | of the child table |
| *array* | of child table attribute names (foreign key attributes) |
| *name* | of the parent table |
| *array* | of parent table attribute names (primary key attributes) |
| *number* | of attributes in foreign key |
| *name* | of the constraint |
| *type* | of the constraint, constants defined in 'reference.h' |

**Returns**

EXIT_SUCCESS

#### 7.96.3.2 AK_delete_row()

```
int AK_delete_row (
            struct list_node * row_root )
```

Function deletes rows.

**Author**

Matija Novak, Dejan Frankovic (added referential integrity)

**Parameters**

| | |
|---|---|
| *row_root* | elements of one row @returs EXIT_SUCCESS if success |

#### 7.96.3.3 AK_get_reference()

```
AK_ref_item AK_get_reference (
            char * tableName,
            char * constraintName )
```

Function that reads a reference entry from system table.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *name* | of the table with reference (with foreign key) |
| *name* | of the reference constraint |

**Returns**

AK_ref_item object with all neccessary information about the reference

#### 7.96.3.4 AK_initialize_new_segment()

```
int AK_initialize_new_segment (
            char * name,
```

```
            int type,
            AK_header * header )
```

Function that initializes a 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**

| | |
|---|---|
| *name* | segment name |
| *type* | segment type |
| *header* | pointer to header that should be written to the new extent (all blocks) |

**Returns**

start address of new segment

#### 7.96.3.5 AK_Insert_New_Element()

```
void AK_Insert_New_Element (
            int newtype,
            void * data,
            char * table,
            char * attribute_name,
            struct list_node * ElementBefore )
```

Used to add a 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**

| | |
|---|---|
| *newtype* | type of the data |
| *data* | the data |
| *table* | table name |
| *attribute_name* | attribute name |
| *element* | element after we which insert the new element |
| *constraint* | is NEW_VALUE |

**Returns**

No return value

#### 7.96.3.6 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 )
```

!! YOU PROBABLY DON'T WANT TO USE THIS FUNCTION !! - Use AK_Update_Existing_Element or AK_Insert↩ _New_Element instead. 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**

| | |
|---|---|
| *newtype* | type of the data |
| *data* | the data |
| *table* | table name |
| *attribute_name* | attribute name |
| *element* | element after we which insert the new element |
| *constraint* | NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for |

**Returns**

No return value

#### 7.96.3.7 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 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**

| | |
|---|---|
| *row_root* | list of elements which contain data of one row |

**Returns**

EXIT_SUCCESS if success else EXIT_ERROR

#### 7.96.3.8 AK_reference_check_attribute()

```
int AK_reference_check_attribute (
            char * tableName,
            char * attribute,
            char * value )
```

Function that checks referential integrity for one attribute.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *child* | table name |
| *attribute* | name (foreign key attribute) |
| *value* | of the attribute we're checking |

**Returns**

EXIT ERROR if check failed, EXIT_SUCCESS if referential integrity is ok

#### 7.96.3.9 AK_reference_check_entry()

```
int AK_reference_check_entry (
            struct list_node * lista )
```

Function that checks a new entry for referential integrity.

**Author**

Dejan Franković

**Parameters**

| | |
|---|---|
| *list* | of elements for insert row |

**Returns**

EXIT_SUCCESS if referential integrity is ok, EXIT_ERROR if it is compromised

#### 7.96.3.10 AK_reference_check_if_update_needed()

```
int AK_reference_check_if_update_needed (
            struct list_node * lista,
            int action )
```

Funcction that quickly checks if there are any referential constraints that should be applied on a given list of changes.

**Author**

Dejan Frankovic

**Parameters**

| | |
|---|---|
| *list* | of elements for update |
| *is* | action UPDATE or DELETE ? |

**Returns**

EXIT_SUCCESS if update is needed, EXIT_ERROR if not

#### 7.96.3.11 AK_reference_check_restricion()

```
int AK_reference_check_restricion (
            struct list_node * lista,
            int action )
```

Function that checks for a REF_TYPE_RESTRICT references appliable to the operation of updating or deleting a row in a table.

**Author**

Dejan Franković

**Parameters**

| | |
|---|---|
| *list* | of elements for update |
| *is* | action UPDATE or DELETE? |

**Returns**

EXIT_SUCCESS if there is no restriction on this action, EXIT_ERROR if there is

#### 7.96.3.12 AK_reference_test()

```
TestResult AK_reference_test ( )
```

Function for testing referential integrity.

**Author**

Dejan Franković

**Returns**

No return value

#### 7.96.3.13 AK_reference_update()

```
int AK_reference_update (
            struct list_node * lista,
            int action )
```

Function that updates child table entries according to ongoing update of parent table entries.

**Author**

Dejan Franković

**Parameters**

| | |
|---|---|
| *list* | of elements for update |
| *is* | action UPDATE or DELETE ? |

**Returns**

EXIT_SUCCESS

#### 7.96.3.14 AK_selection()

```
int AK_selection (
              char * srcTable,
              char * dstTable,
              struct list_node * expr )
```

Function that which implements selection.

**Author**

Matija Šestak.

**Parameters**

| | |
|---|---|
| *∗srcTable* | source table name |
| *∗dstTable* | destination table name |
| *∗expr* | list with posfix notation of the logical expression |

**Returns**

EXIT_SUCCESS

#### 7.96.3.15 AK_Update_Existing_Element()

```
void AK_Update_Existing_Element (
              int newtype,
              void * data,
              char * table,
              char * attribute_name,
              struct list_node * ElementBefore )
```

Used to add a constraint attribute which will define what element gets updated when the operation is executed.

**Author**

Igor Rinkovec

**Parameters**

| | |
|---|---|
| *newtype* | type of the data |
| *data* | the data |
| *table* | table name |
| *attribute_name* | attribute name |
| *element* | element after we which insert the new element |
| *constraint* | is NEW_VALUE |

**Returns**

No return value

#### 7.96.3.16 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**

| | |
|---|---|
| *row_root* | elements of one row |

**Returns**

EXIT_SUCCESS if success

## 7.97 sql/cs/unique.c File Reference

```
#include "unique.h"
```
Include dependency graph for unique.c:

### Functions

- int AK_set_constraint_unique (char *tableName, char attName[], char constraintName[])

  *Function that sets unique constraint on attribute(s)*
- int AK_read_constraint_unique (char *tableName, char attName[], char newValue[])

  *Function that checks if the insertion of some value(s) would violate the UNIQUE constraint.*
- int AK_delete_constraint_unique (char *tableName, char *constraintName)

  *Function for deleting specific unique constraint.*
- TestResult AK_unique_test ()

  *Function for testing UNIQUE constraint.*

### 7.97.1 Detailed Description

Provides functions for unique constraint

## 7.97.2 Function Documentation

### 7.97.2.1 AK_delete_constraint_unique()

```
int AK_delete_constraint_unique (
            char * tableName,
            char * constraintName )
```

Function for deleting specific unique constraint.

**Author**

Blaž Rajič, updated by Bruno Pilošta

**Parameters**

| | |
|---|---|
| *tableName* | name of table on which constraint refers |
| *constraintName* | name of constraint |

**Returns**

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

### 7.97.2.2 AK_read_constraint_unique()

```
int AK_read_constraint_unique (
            char * tableName,
            char attName[],
            char newValue[] )
```

Function that checks if the insertion of some value(s) would violate the UNIQUE constraint.

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char∗* | tableName name of table |
| *char* | attName[] name(s) of attribute(s), if you want to check combination of values of more attributes seperate names of attributes with constant SEPARATOR (see test) |
| *char* | newValue[] new value(s), if you want to check combination of values of more attributes seperate 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 AK_tuple_to_string(struct list_node ∗tuple) or with sprintf in a similiar 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

#### 7.97.2.3 AK_set_constraint_unique()

```
int AK_set_constraint_unique (
            char * tableName,
            char attName[],
            char constraintName[] )
```

Function that sets unique constraint on attribute(s)

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char** | tableName name of table |
| *char* | 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) |
| *char* | constraintName[] name of constraint |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.97.2.4 AK_unique_test()

```
TestResult AK_unique_test ( )
```

Function for testing UNIQUE constraint.

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Returns**

No return value

# 7.98 sql/cs/unique.h File Reference

```
#include "../../auxi/test.h"
#include "../../file/table.h"
#include "../../file/fileio.h"
#include "../../auxi/mempro.h"
#include "../../auxi/dictionary.h"
#include "constraint_names.h"
```

Include dependency graph for unique.h: This graph shows which files directly or indirectly include this file:

## Functions

- int AK_set_constraint_unique (char *tableName, char attName[], char constraintName[])

  *Function that sets unique constraint on attribute(s)*
- int AK_read_constraint_unique (char *tableName, char attName[], char newValue[])

  *Function that checks if the insertion of some value(s) would violate the UNIQUE constraint.*
- int AK_delete_constraint_unique (char *tableName, char constraintName[])

  *Function for deleting specific unique constraint.*
- TestResult AK_unique_test ()

  *Function for testing UNIQUE constraint.*

## 7.98.1 Detailed Description

Header file that provides functions and defines for unique constraint

## 7.98.2 Function Documentation

### 7.98.2.1 AK_delete_constraint_unique()

```
int AK_delete_constraint_unique (
            char * tableName,
            char constraintName[] )
```

Function for deleting specific unique constraint.

**Author**

Maja Vračan, updated by Blaž Rajič

**Parameters**

| | |
|---|---|
| *tableName* | name of table on which constraint refers |
| *constraintName* | name of constraint |

**Returns**

EXIT_SUCCESS when constraint is deleted, else EXIT_ERROR

#### 7.98.2.2 AK_read_constraint_unique()

```
int AK_read_constraint_unique (
            char * tableName,
            char attName[],
            char newValue[] )
```

Function that checks if the insertion of some value(s) would violate the UNIQUE constraint.

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char** | tableName name of table |
| *char* | attName[] name(s) of attribute(s), if you want to check combination of values of more attributes seperate names of attributes with constant SEPARATOR (see test) |
| *char* | newValue[] new value(s) |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char** | tableName name of table |
| *char* | attName[] name(s) of attribute(s), if you want to check combination of values of more attributes seperate names of attributes with constant SEPARATOR (see test) |
| *char* | newValue[] new value(s), if you want to check combination of values of more attributes seperate 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 AK_tuple_to_string(struct list_node *tuple) or with sprintf in a similiar 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

#### 7.98.2.3 AK_set_constraint_unique()

```
int AK_set_constraint_unique (
            char * tableName,
            char attName[],
            char constraintName[] )
```

Function that sets unique constraint on attribute(s)

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Parameters**

| | |
|---|---|
| *char** | tableName name of table |
| *char* | 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) |
| *char* | constraintName[] name of constraint |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.98.2.4 AK_unique_test()

```
TestResult AK_unique_test ( )
```

Function for testing UNIQUE constraint.

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Returns**

No return value

## 7.99 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 the 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 checking if the element(view, function, sequence, user ...) exist in system catalog table.*
- TestResult AK_drop_test ()

  *Function for testing all DROP functions.*

## Variables

- char * system_catalog [NUM_SYS_TABLES]

### 7.99.1 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

### 7.99.2 Function Documentation

#### 7.99.2.1 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**

| | |
|---|---|
| *type* | drop type |
| *drop_arguments* | arguments of DROP command |

#### 7.99.2.2 AK_drop_help_function()

```
void AK_drop_help_function (
            char * tblName,
            char * sys_table )
```

Help function for the 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**

| | |
|---|---|
| *tblName* | name of table or index |
| *sys_table* | name of system catalog table |

#### 7.99.2.3 AK_drop_test()

```
TestResult 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

#### 7.99.2.4 AK_if_exist()

```
int AK_if_exist (
            char * tblName,
            char * sys_table )
```

Help function for checking if the element(view, function, sequence, user ...) exist in system catalog table.

**Author**

Jurica Hlevnjak, updated by Tomislav Ilisevic

**Parameters**

| | |
|---|---|
| *tblName* | name of table, index view, function, trigger, sequence, user, group or constraint |
| *sys_table* | name of system catalog table |

**Returns**

if element exist in system catalog returns 1, if not returns 0

### 7.99.3 Variable Documentation

#### 7.99.3.1 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"
}
```

## 7.100 sql/drop.h File Reference

```
#include "../auxi/test.h"
#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 "../auxi/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.*
- TestResult AK_drop_test ()

  *Function for testing all DROP functions.*
- int AK_if_exist (char *tblName, char *sys_table)

  *Help function for checking if the element(view, function, sequence, user ...) exist in system catalog table.*

### 7.100.1 Detailed Description

Header file that provides data structures, functions and defines for unique constraint

### 7.100.2 Typedef Documentation

#### 7.100.2.1 AK_drop_arguments

```
typedef struct drop_arguments AK_drop_arguments
```

### 7.100.3 Function Documentation

#### 7.100.3.1 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**

| | |
|---|---|
| *type* | drop type |
| *drop_arguments* | arguments of DROP command |

#### 7.100.3.2 AK_drop_test()

```
TestResult 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

#### 7.100.3.3 AK_if_exist()

```
int AK_if_exist (
            char * tblName,
            char * sys_table )
```

Help function for checking if the element(view, function, sequence, user ...) exist in system catalog table.

**Author**

Jurica Hlevnjak, updated by Tomislav Ilisevic

**Parameters**

| | |
|---|---|
| *tblName* | name of table, index view, function, trigger, sequence, user, group or constraint |
| *sys_table* | name of system catalog table |

**Returns**

if element exist in system catalog returns 1, if not returns 0

## 7.101 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 belongs to a function.*
- int AK_check_function_arguments_type (int function_id, struct list_node *args)

  *Function that checks whether arguments belongs to a 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 that removes a function by its obj_id.*
- int AK_function_arguments_remove_by_obj_id (int *obj_id)

  *Function that 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.*
- TestResult AK_function_test ()

  *Function for functions testing.*

### 7.101.1 Detailed Description

Provides functions for functions

### 7.101.2 Function Documentation

#### 7.101.2.1 AK_check_function_arguments()

```
int AK_check_function_arguments (
            int function_id,
            struct list_node * arguments_list )
```

Function that checks whether arguments belongs to a function.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| ∗*function_id* | id of the function |
| ∗*arguments_list* | list of arguments |

**Returns**

EXIT_SUCCESS of the function or EXIT_ERROR

#### 7.101.2.2 AK_check_function_arguments_type()

```
int AK_check_function_arguments_type (
            int function_id,
            struct list_node * args )
```

Function that checks whether arguments belongs to a function but only checks argument type (not name). Used for drop function.

**Author**

Jurica Hlevnjak updated by Aleksandra Polak

**Parameters**

| | |
|---|---|
| *function↩ _id* | id of the function |
| *args* | function arguments |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.101.2.3 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**

| | |
|---|---|
| ∗*name* | name of the function |
| ∗*return_type* | data type returned from a function - values from 0 to 13 - defined in constants.h |
| ∗*arguments_list* | list of function arguments |

**Returns**

function id or EXIT_ERROR

#### 7.101.2.4 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**

| | |
|---|---|
| ∗*function_id* | id of the function to which the argument belongs |
| ∗*arg_number* | number of the argument |
| ∗*arg_type* | data type of the argument |
| ∗*argname* | name of the argument |

**Returns**

function argument id or EXIT_ERROR

#### 7.101.2.5 AK_function_arguments_remove_by_obj_id()

```
int AK_function_arguments_remove_by_obj_id (
              int * obj_id )
```

Function that removes function arguments by function id.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *obj↩ _id* | obj_id of the function |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

### 7.101.2.6 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**

| | |
|---|---|
| *∗name* | name of the function to be modified |
| *∗arguments_list* | list of function arguments |
| *∗new_return_type* | new return type |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

### 7.101.2.7 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**

| | |
|---|---|
| ∗*name* | name of the function |
| ∗*arguments_list* | list of arguments |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.101.2.8 AK_function_remove_by_obj_id()

```
int AK_function_remove_by_obj_id (
            int obj_id )
```

Function that removes a function by its obj_id.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *obj↩ _id* | obj_id of the function |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.101.2.9 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**

| | |
|---|---|
| ∗*name* | name of the function to be modified |
| ∗*arguments_list* | list of arguments to be modified |
| ∗*new_name* | new name of the function |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.101.2.10 AK_function_test()

```
TestResult AK_function_test ( )
```

Function for functions testing.

**Author**

Boris Kišić, updated by Tomislav Ilisevic

**Returns**

No return value

#### 7.101.2.11 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**

| | |
|---|---|
| *∗function* | name of the function |
| *∗arguments_list* | list of arguments |

**Returns**

obj_id of the function or EXIT_ERROR

## 7.102 sql/function.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
```

```
#include "../file/fileio.h"
#include "../auxi/mempro.h"
#include "../auxi/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 belongs to a function.*
- int AK_check_function_arguments_type (int function_id, struct list_node *args)

  *Function that checks whether arguments belongs to a 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 that removes a function by its obj_id.*
- int AK_function_arguments_remove_by_obj_id (int *obj_id)

  *Function that 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.*
- TestResult AK_function_test ()

  *Function for functions testing.*

### 7.102.1 Detailed Description

Header file that provides functions and defines for functions

Header file that provides functions and defines for view.c

### 7.102.2 Function Documentation

#### 7.102.2.1 AK_check_function_arguments()

```
int AK_check_function_arguments (
            int function_id,
            struct list_node * arguments_list )
```

Function that checks whether arguments belongs to a function.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| ∗*function_id* | id of the function |
| ∗*arguments_list* | list of arguments |

**Returns**

EXIT_SUCCESS of the function or EXIT_ERROR

#### 7.102.2.2 AK_check_function_arguments_type()

```
int AK_check_function_arguments_type (
            int function_id,
            struct list_node * args )
```

Function that checks whether arguments belongs to a function but only checks argument type (not name). Used for drop function.

**Author**

Jurica Hlevnjak updated by Aleksandra Polak

**Parameters**

| | |
|---|---|
| *function↩ _id* | id of the function |
| *args* | function arguments |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.102.2.3 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**

| | |
|---|---|
| *∗name* | name of the function |
| *∗return_type* | data type returned from a function - values from 0 to 13 - defined in constants.h |
| *∗arguments_list* | list of function arguments |

**Returns**

function id or EXIT_ERROR

#### 7.102.2.4 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**

| | |
|---|---|
| *∗function_id* | id of the function to which the argument belongs |
| *∗arg_number* | number of the argument |
| *∗arg_type* | data type of the argument |
| *∗argname* | name of the argument |

**Returns**

function argument id or EXIT_ERROR

#### 7.102.2.5 AK_function_arguments_remove_by_obj_id()

```
int AK_function_arguments_remove_by_obj_id (
              int * obj_id )
```

Function that removes function arguments by function id.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *obj↩ _id* | obj_id of the function |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

### 7.102.2.6 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**

| | |
|---|---|
| *∗name* | name of the function to be modified |
| *∗arguments_list* | list of function arguments |
| *∗new_return_type* | new return type |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

### 7.102.2.7 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**

| | |
|---|---|
| *∗name* | name of the function |
| *∗arguments_list* | list of arguments |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.102.2.8 AK_function_remove_by_obj_id()

```
int AK_function_remove_by_obj_id (
            int obj_id )
```

Function that removes a function by its obj_id.

**Author**

Boris Kišić

**Parameters**

| | |
|---|---|
| *obj↩ _id* | obj_id of the function |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.102.2.9 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**

| | |
|---|---|
| *∗name* | name of the function to be modified |
| *∗arguments_list* | list of arguments to be modified |
| *∗new_name* | new name of the function |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.102.2.10 AK_function_test()

```
TestResult AK_function_test ( )
```

Function for functions testing.

**Author**

Boris Kišić, updated by Tomislav Ilisevic

**Returns**

No return value

#### 7.102.2.11 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**

| | |
|---|---|
| *∗function* | name of the function |
| *∗arguments_list* | list of arguments |

**Returns**

obj_id of the function or EXIT_ERROR

## 7.103 sql/insert.c File Reference

```
#include "insert.h"
```

Include dependency graph for insert.c:

## Functions

- AK_header * AK_get_insert_header (int *size, char *tblName, struct list_node *columns)

  *Function creates headers based on entered columns in SQL command. If no columns are entered it will use table header.*
- int AK_insert (char *tblName, struct list_node *columns, struct list_node *values)

  *Function that implements SQL insert command.*
- TestResult AK_insert_test ()

### 7.103.1 Function Documentation

#### 7.103.1.1 AK_get_insert_header()

```
AK_header* AK_get_insert_header (
            int * size,
            char * tblName,
            struct list_node * columns )
```

Function creates headers based on entered columns in SQL command. If no columns are entered it will use table header.

**Author**

Filip Žmuk

**Parameters**

| | |
|---|---|
| *size* | pointer to integer in which size of header will be saved |
| *tblName* | table in which rows will be inserted |
| *columns* | list of columns in SQL command |

**Returns**

header for values to be inserted or EXIT_ERROR

#### 7.103.1.2 AK_insert()

```
int AK_insert (
            char * tblName,
            struct list_node * columns,
            struct list_node * values )
```

Function that implements SQL insert command.

**Author**

Filip Žmuk

**Parameters**

| | |
|---|---|
| *tableName* | table in which rows will be inserted |
| *columns* | list of columns |
| *values* | values to be inserted |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.103.1.3 AK_insert_test()

```
TestResult AK_insert_test ( )
```

## 7.104 sql/insert.h File Reference

```
#include "../auxi/mempro.h"
#include "../auxi/test.h"
#include "../file/fileio.h"
#include "../auxi/constants.h"
#include "../file/table.h"
#include "drop.h"
```

Include dependency graph for insert.h: This graph shows which files directly or indirectly include this file:

### Functions

- AK_header * AK_get_insert_header (int *size, char *tblName, struct list_node *columns)

  *Function creates headers based on entered columns in SQL command. If no columns are entered it will use table header.*
- int AK_insert (char *tableName, struct list_node *columns, struct list_node *values)

  *Function that implements SQL insert command.*
- TestResult AK_insert_test ()

### 7.104.1 Detailed Description

Implementation of SQL insert command.

Header file SQL insert command.

### 7.104.2 Function Documentation

#### 7.104.2.1 AK_get_insert_header()

```
AK_header* AK_get_insert_header (
             int * size,
             char * tblName,
             struct list_node * columns )
```

Function creates headers based on entered columns in SQL command. If no columns are entered it will use table header.

**Author**

Filip Žmuk

**Parameters**

| | |
|---|---|
| *size* | pointer to integer in which size of header will be saved |
| *tblName* | table in which rows will be inserted |
| *columns* | list of columns in SQL command |

**Returns**

header for values to be inserted or EXIT_ERROR

#### 7.104.2.2 AK_insert()

```
int AK_insert (
             char * tblName,
             struct list_node * columns,
             struct list_node * values )
```

Function that implements SQL insert command.

**Author**

Filip Žmuk

**Parameters**

| | |
|---|---|
| *tableName* | table in which rows will be inserted |
| *columns* | list of columns |
| *values* | values to be inserted |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.104.2.3 AK_insert_test()

```
TestResult AK_insert_test ( )
```

## 7.105 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)

  *Function that returns an ID of the given user.*
- int AK_user_check_pass (char *username, int *password)

  *Function that checks if there is user with given password.*
- int AK_user_remove_by_name (char *name)

  *Function that removes the given user.*
- int AK_user_rename (char *old_name, char *new_name, int *password)

  *Function that renames a given user.*
- int AK_group_add (char *name, int set_id)

  *Function that adds a new group.*
- int AK_group_get_id (char *name)

  *Function that returns the ID from the given group name.*
- int AK_group_remove_by_name (char *name)

  *Function that removes the given group.*
- int AK_group_rename (char *old_name, char *new_name)

  *Function that renames the given group.*
- int AK_grant_privilege_user (char *username, char *table, char *right)

  *Function that grants a specific privilege to the desired user on a given table.*
- int AK_revoke_privilege_user (char *username, char *table, char *right)

  *Function that revokes users privilege on the given table.*
- int AK_revoke_all_privileges_user (char *username)

  *Function that revokes ALL user's privileges on ALL tables (for DROP user)*
- int AK_grant_privilege_group (char *groupname, char *table, char *right)

  *Function that grants a privilege to a given group on a given table.*
- int AK_revoke_privilege_group (char *groupname, char *table, char *right)

  *Function that revokes a groups privilege on the given table.*
- int AK_revoke_all_privileges_group (char *groupname)

  *Function that revokes ALL privileges from the desired group on ALL tables (needed for DROP group)*
- int AK_add_user_to_group (char *user, char *group)

  *Function that puts the desired user in the given group.*
- int AK_remove_user_from_all_groups (char *user)

  *Function that removes user from all groups. Used for DROP user.*
- int AK_remove_all_users_from_group (char *group)

  *Function that removes all users from a group. Used for DROP group.*
- int AK_check_privilege (char *username, char *table, char *privilege)

*Function that checks whether the given user has a right for the given operation on the given table.*

- int AK_check_user_privilege (char ∗user)

  *Function that 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)

  *Function that checks if the group has any privileges. Used in drop group for restriction.*

- TestResult AK_privileges_test ()

  *Function that tests all the previous functions.*

## 7.105.1 Detailed Description

Provides functions for privileges

## 7.105.2 Function Documentation

### 7.105.2.1 AK_add_user_to_group()

```
int AK_add_user_to_group (
            char * user,
            char * group )
```

Function that 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**

| | |
|---|---|
| ∗*user* | username of user which will be put in group |
| ∗*group* | name of group in which user will be put |

**Returns**

EXIT_SUCCESS or EXIT_ERROR if the user is already in the group

### 7.105.2.2 AK_check_group_privilege()

```
int AK_check_group_privilege (
            char * group )
```

Function that 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**

| | |
|---|---|
| *group* | name of group |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.105.2.3 AK_check_privilege()

```
int AK_check_privilege (
            char * username,
            char * table,
            char * privilege )
```

Function that checks whether the given user has a right for the given operation on the given table.

**Author**

Kristina Takač, updated by Marko Flajšek

**Parameters**

| | |
|---|---|
| ∗*user* | username for which we want check privileges |
| ∗*table* | name of table for which we want to check whether user has right on |
| ∗*privilege* | 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

#### 7.105.2.4 AK_check_user_privilege()

```
int AK_check_user_privilege (
            char * user )
```

Function that 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**

| | |
|---|---|
| *user* | name of user |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

### 7.105.2.5 AK_grant_privilege_group()

```
int AK_grant_privilege_group (
            char * groupname,
            char * table,
            char * right )
```

Function that grants a privilege to a given group on a given table.

**Author**

Kristina Takač.

**Parameters**

| | |
|---|---|
| ∗*groupname* | name of group to which we want to grant privilege |
| ∗*table* | name of table on which privilege will be granted to user |
| ∗*right* | 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

### 7.105.2.6 AK_grant_privilege_user()

```
int AK_grant_privilege_user (
            char * username,
            char * table,
            char * right )
```

Function that grants a 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**

| | |
|---|---|
| ∗*username* | username of user to whom we want to grant privilege |
| ∗*table* | name of table on which privilege will be granted to user |
| ∗*right* | 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

#### 7.105.2.7 AK_group_add()

```
int AK_group_add (
            char * name,
            int set_id )
```

Function that adds a new group.

**Author**

Kristina Takač, edited by Ljubo Barać

**Parameters**

| | |
|---|---|
| ∗*name* | name of group to be added |
| *set_id* | non default id to be passed |

**Returns**

id of group

#### 7.105.2.8 AK_group_get_id()

```
int AK_group_get_id (
            char * name )
```

Function that returns the ID from the given group name.

**Author**

Kristina Takač.

**Parameters**

| | |
|---|---|
| *∗name* | name of group whose id we are looking for |

**Returns**

id of group, otherwise EXIT_ERROR

#### 7.105.2.9 AK_group_remove_by_name()

```
int AK_group_remove_by_name (
            char * name )
```

Function that removes the given group.

**Author**

Ljubo Barać

**Parameters**

| | |
|---|---|
| *name* | Name of the group to be removed |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.105.2.10 AK_group_rename()

```
int AK_group_rename (
            char * old_name,
            char * new_name )
```

Function that renames the given group.

**Author**

Ljubo Barać, update by Lidija Lastavec

**Parameters**

| | |
|---|---|
| *old_name* | Name of the group to be renamed |
| *new_name* | New name of the group |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.105.2.11 AK_privileges_test()

```
TestResult 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

#### 7.105.2.12 AK_remove_all_users_from_group()

```
int AK_remove_all_users_from_group (
            char * group )
```

Function that removes all users from a group. Used for DROP group.

**Author**

Jurica Hlevnjak, update by Lidija Lastavec

**Parameters**

| | |
|---|---|
| *group* | name of group |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.105.2.13 AK_remove_user_from_all_groups()

```
int AK_remove_user_from_all_groups (
            char * user )
```

Function that removes user from all groups. Used for DROP user.

**Author**

Jurica Hlevnjak, update by Lidija Lastavec

**Parameters**

| | |
|---|---|
| *user* | name of user |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.105.2.14 AK_revoke_all_privileges_group()

```
int AK_revoke_all_privileges_group (
            char * groupname )
```

Function that revokes ALL privileges from the desired group on ALL tables (needed for DROP group)

**Author**

Jurica Hlevnjak

**Parameters**

| | |
|---|---|
| *groupname* | name of group from which we want to revoke all privileges |

**Returns**

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

#### 7.105.2.15 AK_revoke_all_privileges_user()

```
int AK_revoke_all_privileges_user (
            char * username )
```

Function that revokes ALL user's privileges on ALL tables (for DROP user)

**Author**

Jurica Hlevnjak, updated by Marko Flajšek

**Parameters**

| | |
|---|---|
| *username* | name of user from whom we want to revoke all privileges |

**Returns**

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

#### 7.105.2.16 AK_revoke_privilege_group()

```
int AK_revoke_privilege_group (
            char * groupname,
            char * table,
            char * right )
```

Function that revokes a groups 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**

| | |
|---|---|
| ∗*grounamep* | name of group which user belongs to |
| ∗*table* | name of table on which privilege will be granted to group |
| ∗*right* | type of privilege which will be granted to group on a given table |

**Returns**

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

#### 7.105.2.17 AK_revoke_privilege_user()

```
int AK_revoke_privilege_user (
            char * username,
            char * table,
            char * right )
```

Function that revokes users 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**

| | |
|---|---|
| ∗*username* | username of user to whom we want to grant privilege |
| ∗*table* | name of table on which privilege will be revoked from user |
| ∗*right* | 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

### 7.105.2.18 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**

| | |
|---|---|
| ∗*username* | username of user to be added |
| ∗*password* | password of user to be added |
| *set_id* | obj_id of the new user |

**Returns**

user_id

### 7.105.2.19 AK_user_check_pass()

```
int AK_user_check_pass (
            char * username,
            int * password )
```

Function that checks if there is user with given password.

**Author**

Fran MIkolić.

**Parameters**

| | |
|---|---|
| ∗*username* | username of user whose password we are checking |
| ∗*password* | password of given username whom we will check |

**Returns**

check 0 if false or 1 if true

#### 7.105.2.20 AK_user_get_id()

```
int AK_user_get_id (
            char * username )
```

Function that returns an ID of the given user.

**Author**

Kristina Takač.

**Parameters**

| | |
|---|---|
| ∗*username* | username of user whose id we are looking for |

**Returns**

user_id, otherwise EXIT_ERROR

#### 7.105.2.21 AK_user_remove_by_name()

```
int AK_user_remove_by_name (
            char * name )
```

Function that removes the given user.

**Author**

Ljubo Barać

**Parameters**

| | |
|---|---|
| *name* | Name of the user to be removed |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.105.2.22 AK_user_rename()

```
int AK_user_rename (
            char * old_name,
            char * new_name,
            int * password )
```

Function that renames a given user.

**Author**

Ljubo Barać, update by Lidija Lastavec, update by Marko Flajšek

**Parameters**

| | |
|---|---|
| *old_name* | Name of the user to be renamed |
| *new_name* | New name of the user |
| *password* | Password of the user to be renamed (should be provided) |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

## 7.106 sql/privileges.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../file/fileio.h"
#include "../file/id.h"
#include "../rec/archive_log.h"
#include "../auxi/mempro.h"
```

Include dependency graph for privileges.h: This graph shows which files directly or indirectly include this file:

### 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)

  *Function that returns an ID of the given user.*
- int AK_user_check_pass (char ∗username, int ∗password)

  *Function that checks if there is user with given password.*
- int AK_group_add (char ∗name, int set_id)

*Function that adds a new group.*

- int AK_group_get_id (char *name)

  *Function that returns the ID from the given group name.*

- int AK_grant_privilege_user (char *username, char *table, char *right)

  *Function that grants a specific privilege to the desired user on a given table.*

- int AK_revoke_privilege_user (char *username, char *table, char *right)

  *Function that revokes users privilege on the given table.*

- int AK_revoke_all_privileges_user (char *username)

  *Function that revokes ALL user's privileges on ALL tables (for DROP user)*

- int AK_grant_privilege_group (char *groupname, char *table, char *right)

  *Function that grants a privilege to a given group on a given table.*

- int AK_revoke_privilege_group (char *groupname, char *table, char *right)

  *Function that revokes a groups privilege on the given table.*

- int AK_revoke_all_privileges_group (char *groupname)

  *Function that revokes ALL privileges from the desired group on ALL tables (needed for DROP group)*

- int AK_add_user_to_group (char *user, char *group)

  *Function that puts the desired user in the given group.*

- int AK_remove_user_from_all_groups (char *user)

  *Function that removes user from all groups. Used for DROP user.*

- int AK_remove_all_users_from_group (char *group)

  *Function that removes all users from a group. Used for DROP group.*

- int AK_check_privilege (char *username, char *table, char *privilege)

  *Function that checks whether the given user has a right for the given operation on the given table.*

- int AK_check_user_privilege (char *user)

  *Function that 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)

  *Function that checks if the group has any privileges. Used in drop group for restriction.*

- int AK_group_remove_by_name (char *name)

  *Function that removes the given group.*

- int AK_user_rename (char *old_name, char *new_name, int *password)

  *Function that renames a given user.*

- int AK_group_rename (char *old_name, char *new_name)

  *Function that renames the given group.*

- TestResult AK_privileges_test ()

  *Function that tests all the previous functions.*

### 7.106.1 Detailed Description

Header file that provides functions and defines for privileges.c

### 7.106.2 Function Documentation

#### 7.106.2.1 AK_add_user_to_group()

```
int AK_add_user_to_group (
            char * user,
            char * group )
```

Function that 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**

| | |
|---|---|
| *∗user* | username of user which will be put in group |
| *∗group* | name of group in which user will be put |

**Returns**

EXIT_SUCCESS or EXIT_ERROR if the user is already in the group

#### 7.106.2.2 AK_check_group_privilege()

```
int AK_check_group_privilege (
            char * group )
```

Function that 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**

| | |
|---|---|
| *group* | name of group |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.106.2.3 AK_check_privilege()

```
int AK_check_privilege (
            char * username,
```

```
            char * table,
            char * privilege )
```

Function that checks whether the given user has a right for the given operation on the given table.

**Author**

Kristina Takač, updated by Marko Flajšek

**Parameters**

| | |
|---|---|
| *∗user* | username for which we want check privileges |
| *∗table* | name of table for which we want to check whether user has right on |
| *∗privilege* | 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

#### 7.106.2.4 AK_check_user_privilege()

```
int AK_check_user_privilege (
            char * user )
```

Function that 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**

| | |
|---|---|
| *user* | name of user |

**Returns**

EXIT_ERROR or EXIT_SUCCESS

#### 7.106.2.5 AK_grant_privilege_group()

```
int AK_grant_privilege_group (
            char * groupname,
            char * table,
            char * right )
```

Function that grants a privilege to a given group on a given table.

**Author**

Kristina Takač.

**Parameters**

| | |
|---|---|
| ∗*groupname* | name of group to which we want to grant privilege |
| ∗*table* | name of table on which privilege will be granted to user |
| ∗*right* | 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

### 7.106.2.6 AK_grant_privilege_user()

```
int AK_grant_privilege_user (
            char * username,
            char * table,
            char * right )
```

Function that grants a 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**

| | |
|---|---|
| ∗*username* | username of user to whom we want to grant privilege |
| ∗*table* | name of table on which privilege will be granted to user |
| ∗*right* | 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

### 7.106.2.7 AK_group_add()

```
int AK_group_add (
            char * name,
            int set_id )
```

Function that adds a new group.

**Author**

Kristina Takač, edited by Ljubo Barać

**Parameters**

| | |
|---|---|
| *∗name* | name of group to be added |
| *set_id* | non default id to be passed |

**Returns**

id of group

#### 7.106.2.8 AK_group_get_id()

```
int AK_group_get_id (
            char * name )
```

Function that returns the ID from the given group name.

**Author**

Kristina Takač.

**Parameters**

| | |
|---|---|
| *∗name* | name of group whose id we are looking for |

**Returns**

id of group, otherwise EXIT_ERROR

#### 7.106.2.9 AK_group_remove_by_name()

```
int AK_group_remove_by_name (
            char * name )
```

Function that removes the given group.

**Author**

Ljubo Barać

**Parameters**

| | |
|---|---|
| *name* | Name of the group to be removed |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

### 7.106.2.10 AK_group_rename()

```
int AK_group_rename (
            char * old_name,
            char * new_name )
```

Function that renames the given group.

**Author**

Ljubo Barać, update by Lidija Lastavec

**Parameters**

| | |
|---|---|
| *old_name* | Name of the group to be renamed |
| *new_name* | New name of the group |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

### 7.106.2.11 AK_privileges_test()

```
TestResult 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

#### 7.106.2.12 AK_remove_all_users_from_group()

```
int AK_remove_all_users_from_group (
            char * group )
```

Function that removes all users from a group. Used for DROP group.

**Author**

Jurica Hlevnjak, update by Lidija Lastavec

**Parameters**

| | |
|---|---|
| *group* | name of group |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.106.2.13 AK_remove_user_from_all_groups()

```
int AK_remove_user_from_all_groups (
            char * user )
```

Function that removes user from all groups. Used for DROP user.

**Author**

Jurica Hlevnjak, update by Lidija Lastavec

**Parameters**

| | |
|---|---|
| *user* | name of user |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.106.2.14 AK_revoke_all_privileges_group()

```
int AK_revoke_all_privileges_group (
            char * groupname )
```

Function that revokes ALL privileges from the desired group on ALL tables (needed for DROP group)

**Author**

Jurica Hlevnjak

**Parameters**

| | |
|---|---|
| *groupname* | name of group from which we want to revoke all privileges |

**Returns**

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

#### 7.106.2.15 AK_revoke_all_privileges_user()

```
int AK_revoke_all_privileges_user (
            char * username )
```

Function that revokes ALL user's privileges on ALL tables (for DROP user)

**Author**

Jurica Hlevnjak, updated by Marko Flajšek

**Parameters**

| | |
|---|---|
| *username* | name of user from whom we want to revoke all privileges |

**Returns**

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

#### 7.106.2.16 AK_revoke_privilege_group()

```
int AK_revoke_privilege_group (
            char * groupname,
            char * table,
            char * right )
```

Function that revokes a groups privilege on the given table.

**Author**

Kristina Takač, updated by Mario Peroković - added comparing by table id

**Parameters**

| | |
|---|---|
| ∗*grounamep* | name of group which user belongs to |
| ∗*table* | name of table on which privilege will be granted to group |
| ∗*right* | type of privilege which will be granted to group on a given table |

**Returns**

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

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**

| | |
|---|---|
| ∗*grounamep* | name of group which user belongs to |
| ∗*table* | name of table on which privilege will be granted to group |
| ∗*right* | type of privilege which will be granted to group on a given table |

**Returns**

EXIT_SUCCESS if privilege is revoked, EXIT_ERROR if it isn't

### 7.106.2.17 AK_revoke_privilege_user()

```
int AK_revoke_privilege_user (
            char * username,
            char * table,
            char * right )
```

Function that revokes users privilege on the given table.

**Author**

Kristina Takač, updated by Mario Peroković - added comparing by table id, and use of user_id in AK_user_right

**Parameters**

| | |
|---|---|
| ∗*username* | username of user to whom we want to grant privilege |
| ∗*table* | name of table on which privilege will be revoked from user |
| ∗*right* | 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

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**

| | |
|---|---|
| ∗*username* | username of user to whom we want to grant privilege |
| ∗*table* | name of table on which privilege will be revoked from user |
| ∗*right* | 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

#### 7.106.2.18 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**

| | |
|---|---|
| ∗*username* | username of user to be added |
| ∗*password* | password of user to be added |
| *set_id* | obj_id of the new user |

**Returns**

user_id

#### 7.106.2.19 AK_user_check_pass()

```
int AK_user_check_pass (
            char * username,
            int * password )
```

Function that checks if there is user with given password.

**Author**

Fran MIkolić.

**Parameters**

| | |
|---|---|
| *∗username* | username of user whose password we are checking |
| *∗password* | password of given username whom we will check |

**Returns**

check 0 if false or 1 if true

#### 7.106.2.20 AK_user_get_id()

```
int AK_user_get_id (
            char * username )
```

Function that returns an ID of the given user.

**Author**

Kristina Takač.

**Parameters**

| | |
|---|---|
| *∗username* | username of user whose id we are looking for |

**Returns**

user_id, otherwise EXIT_ERROR

#### 7.106.2.21 AK_user_rename()

```
int AK_user_rename (
            char * old_name,
```

```
char * new_name,
int * password )
```

Function that renames a given user.

**Author**

Ljubo Barać, update by Lidija Lastavec, update by Marko Flajšek

**Parameters**

| | |
|---|---|
| *old_name* | Name of the user to be renamed |
| *new_name* | New name of the user |
| *password* | Password of the user to be renamed (should be provided) |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

## 7.107 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, struct list_node *ordering)

  *Function that implements SELECT relational operator.*
- TestResult AK_select_test ()

  *Function for testing the implementation.*

### 7.107.1 Detailed Description

Provides functions for SELECT relational operator

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

### 7.107.2 Function Documentation

#### 7.107.2.1 AK_select()

```
int AK_select (
            char * srcTable,
            char * destTable,
            struct list_node * attributes,
            struct list_node * condition,
            struct list_node * ordering )
```

Function that implements SELECT relational operator.

**Author**

Filip Žmuk

**Parameters**

| | |
|---|---|
| *srcTable* | - original table that is used for selection |
| *destTable* | - table that contains the result |
| *condition* | - condition for selection |
| *attributes* | - atributes to be selected |
| *ordering* | - atributes for result sorting |

**Returns**

EXIT_SUCCESS if cache result in memory and print table else break

#### 7.107.2.2 AK_select_test()

```
TestResult AK_select_test ( )
```

Function for testing the implementation.

**Author**

Renata Mesaros, updated by Filip Žmuk and Josip Susnjara

## 7.108 sql/select.h File Reference

```
#include "../file/table.h"
#include "../auxi/test.h"
#include "../file/fileio.h"
#include "../rel/selection.h"
#include "../rel/projection.h"
#include "../auxi/auxiliary.h"
#include "../auxi/mempro.h"
#include "../file/filesort.h"
```

Include dependency graph for select.h: This graph shows which files directly or indirectly include this file:

## Functions

- int AK_select (char *srcTable, char *destTable, struct list_node *attributes, struct list_node *condition, struct list_node *ordering)

  *Function that implements SELECT relational operator.*
- TestResult AK_select_test ()

  *Function for testing the implementation.*

## 7.108.1 Detailed Description

Header file that provides functions for select.h

## 7.108.2 Function Documentation

### 7.108.2.1 AK_select()

```
int AK_select (
            char * srcTable,
            char * destTable,
            struct list_node * attributes,
            struct list_node * condition,
            struct list_node * ordering )
```

Function that implements SELECT relational operator.

**Author**

Filip Žmuk

**Parameters**

| | |
|---|---|
| *srcTable* | - original table that is used for selection |
| *destTable* | - table that contains the result |
| *condition* | - condition for selection |
| *attributes* | - atributes to be selected |
| *ordering* | - atributes for result sorting |

**Returns**

EXIT_SUCCESS if cache result in memory and print table else break

### 7.108.2.2 AK_select_test()

```
TestResult AK_select_test ( )
```

Function for testing the implementation.

**Author**

Renata Mesaros, updated by Filip Žmuk and Josip Susnjara

## 7.109 sql/trigger.c File Reference

`#include "trigger.h"`
Include dependency graph for trigger.c:

### Functions

- int AK_trigger_save_conditions (int trigger, struct list_node *condition)

  *Function that saves conditions for a trigger.*
- int AK_trigger_add (char *name, char *event, struct list_node *condition, char *table, char *function, struct list_node *arguments_list)

  *Function that adds a trigger to the 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 the system table by name.*
- int AK_trigger_remove_by_obj_id (int obj_id)

  *Function that removes a trigger by its obj_id.*
- int AK_trigger_edit (char *name, char *event, struct list_node *condition, char *table, char *function)

  *Function that 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 that fetches postfix list of conditions for the trigger (compatible with selection)*
- int AK_trigger_rename (char *old_name, char *new_name, char *table)

  *Function that renames the trigger.*
- TestResult AK_trigger_test ()

  *Function for trigger testing.*

### 7.109.1 Detailed Description

Provides functions for triggers

### 7.109.2 Function Documentation

### 7.109.2.1 AK_trigger_add()

```
int AK_trigger_add (
             char * name,
             char * event,
             struct list_node * condition,
             char * table,
             char * function,
             struct list_node * arguments_list )
```

Function that adds a trigger to the system table.

#### Author

Unknown updated by Aleksandra Polak

#### Parameters

| | |
|---|---|
| *∗name* | name of the trigger |
| *∗event* | event that calls the trigger - this should perhaps be an integer with defined constants... |
| *∗condition* | AK_list list of conditions in postfix |
| *∗table* | name of the table trigger is hooked on |
| *∗function* | function that is being called by the trigger |

#### Returns

trigger id or EXIT_ERROR

### 7.109.2.2 AK_trigger_edit()

```
int AK_trigger_edit (
             char * name,
             char * event,
             struct list_node * condition,
             char * table,
             char * function )
```

Function that 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.

Function that edits information about the trigger in system table.

#### Author

Unknown

**Parameters**

| | |
|---|---|
| ∗*name* | name of the trigger (or NULL if using obj_id) |
| ∗*event* | event of the trigger (or NULL if it isn't changing) |
| ∗*condition* | list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed) |
| ∗*table* | name of the connected table (or NULL id using obj_id) |
| ∗*function* | name of the connected function (or NULL if it isn't changing) |

**Returns**

EXIT_SUCCESS or EXIT_ERROR @IMPORTANT: ∗AK_dbg_messg.. need to be fixed then we could uncomment lines with this function, by then we use ∗printf.

#### 7.109.2.3 AK_trigger_get_conditions()

```
struct list_node* AK_trigger_get_conditions (
            int trigger )
```

Function that fetches postfix list of conditions for the trigger (compatible with selection)

**Author**

Unknown, updated by Mario Peroković

**Parameters**

| | |
|---|---|
| *trigger* | obj_id of the trigger |

**Returns**

list of conditions for the trigger

#### 7.109.2.4 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**

| | |
|---|---|
| *∗name* | name of the trigger |
| *∗table* | name of the table on which the trigger is hooked |

**Returns**

obj_id of the trigger or EXIT_ERROR

### 7.109.2.5 AK_trigger_remove_by_name()

```
int AK_trigger_remove_by_name (
            char * name,
            char * table )
```

Function that removes a trigger from the system table by name.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *∗name* | name of the trigger |
| *∗table* | name of the table |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

### 7.109.2.6 AK_trigger_remove_by_obj_id()

```
int AK_trigger_remove_by_obj_id (
            int obj_id )
```

Function that removes a trigger by its obj_id.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *obj_id* | obj_id of the trigger |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.109.2.7 AK_trigger_rename()

```
int AK_trigger_rename (
            char * old_name,
            char * new_name,
            char * table )
```

Function that renames the trigger.

**Author**

Ljubo Barać

**Parameters**

| | |
|---|---|
| *old_name* | Name of the trigger to be renamed |
| *new_name* | New name of the trigger |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.109.2.8 AK_trigger_save_conditions()

```
int AK_trigger_save_conditions (
            int trigger,
            struct list_node * condition )
```

Function that saves conditions for a trigger.

**Author**

Unknown, updated by Mario Peroković, check if data is TYPE_INT

**Parameters**

| | |
|---|---|
| *trigger* | obj_id of the trigger in question |
| *∗condition* | AK_list list of conditions |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.109.2.9 AK_trigger_test()

```
TestResult AK_trigger_test ( )
```

Function for trigger testing.

**Author**

Unknown updated by Aleksandra Polak

## 7.110 sql/trigger.h File Reference

```
#include "../auxi/test.h"
#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)

  *Function that saves conditions for a trigger.*
- int AK_trigger_add (char ∗name, char ∗event, struct list_node ∗condition, char ∗table, char ∗function, struct list_node ∗arguments_list)

  *Function that adds a trigger to the 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 the system table by name.*
- int AK_trigger_remove_by_obj_id (int obj_id)

  *Function that removes a trigger by its obj_id.*
- int AK_trigger_edit (char ∗name, char ∗event, struct list_node ∗condition, char ∗table, char ∗function)

  *Function that edits information about the trigger in system table.*
- struct list_node ∗ AK_trigger_get_conditions (int trigger)

  *Function that fetches postfix list of conditions for the trigger (compatible with selection)*
- int AK_trigger_rename (char ∗old_name, char ∗new_name, char ∗table)

  *Function that renames the trigger.*
- TestResult AK_trigger_test ()

  *Function for trigger testing.*

### 7.110.1 Detailed Description

Header file that provides functions and defines for trigger.c

### 7.110.2 Function Documentation

#### 7.110.2.1 AK_trigger_add()

```
int AK_trigger_add (
            char * name,
            char * event,
            struct list_node * condition,
            char * table,
            char * function,
            struct list_node * arguments_list )
```

Function that adds a trigger to the system table.

**Author**

Unknown updated by Aleksandra Polak

**Parameters**

| | |
|---|---|
| *∗name* | name of the trigger |
| *∗event* | event that calls the trigger - this should perhaps be an integer with defined constants... |
| *∗condition* | AK_list list of conditions in postfix |
| *∗table* | name of the table trigger is hooked on |
| *∗function* | function that is being called by the trigger |

**Returns**

trigger id or EXIT_ERROR

#### 7.110.2.2 AK_trigger_edit()

```
int AK_trigger_edit (
            char * name,
            char * event,
            struct list_node * condition,
            char * table,
            char * function )
```

Function that edits information about the trigger in system table.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| ∗*name* | name of the trigger (or NULL if using obj_id) |
| ∗*event* | event of the trigger (or NULL if it isn't changing) |
| ∗*condition* | list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed) |
| ∗*table* | name of the connected table (or NULL id using obj_id) |
| ∗*function* | name of the connected function (or NULL if it isn't changing) |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

Function that edits information about the trigger in system table.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| ∗*name* | name of the trigger (or NULL if using obj_id) |
| ∗*event* | event of the trigger (or NULL if it isn't changing) |
| ∗*condition* | list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed) |
| ∗*table* | name of the connected table (or NULL id using obj_id) |
| ∗*function* | name of the connected function (or NULL if it isn't changing) |

**Returns**

EXIT_SUCCESS or EXIT_ERROR @IMPORTANT: ∗AK_dbg_messg.. need to be fixed then we could uncomment lines with this function, by then we use ∗printf.

### 7.110.2.3 AK_trigger_get_conditions()

```
struct list_node* AK_trigger_get_conditions (
            int trigger )
```

Function that fetches postfix list of conditions for the trigger (compatible with selection)

**Author**

Unknown, updated by Mario Peroković

**Parameters**

| | |
|---|---|
| *trigger* | obj_id of the trigger |

**Returns**

list of conditions for the trigger

#### 7.110.2.4 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**

| | |
|---|---|
| *∗name* | name of the trigger |
| *∗table* | name of the table on which the trigger is hooked |

**Returns**

obj_id of the trigger or EXIT_ERROR

#### 7.110.2.5 AK_trigger_remove_by_name()

```
int AK_trigger_remove_by_name (
            char * name,
            char * table )
```

Function that removes a trigger from the system table by name.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *∗name* | name of the trigger |
| *∗table* | name of the table |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.110.2.6 AK_trigger_remove_by_obj_id()

```
int AK_trigger_remove_by_obj_id (
            int obj_id )
```

Function that removes a trigger by its obj_id.

**Author**

Unknown

**Parameters**

| | |
|---|---|
| *obj_id* | obj_id of the trigger |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.110.2.7 AK_trigger_rename()

```
int AK_trigger_rename (
            char * old_name,
            char * new_name,
            char * table )
```

Function that renames the trigger.

**Author**

Ljubo Barać

**Parameters**

| | |
|---|---|
| *old_name* | Name of the trigger to be renamed |
| *new_name* | New name of the trigger |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.110.2.8 AK_trigger_save_conditions()

```
int AK_trigger_save_conditions (
            int trigger,
            struct list_node * condition )
```

Function that saves conditions for a trigger.

**Author**

Unknown, updated by Mario Peroković, check if data is TYPE_INT

**Parameters**

| | |
|---|---|
| *trigger* | obj_id of the trigger in question |
| *∗condition* | AK_list list of conditions |

**Returns**

EXIT_SUCCESS or EXIT_ERROR

#### 7.110.2.9 AK_trigger_test()

```
TestResult AK_trigger_test ( )
```

Function for trigger testing.

**Author**

Unknown updated by Aleksandra Polak

## 7.111 sql/view.c File Reference

```
#include "view.h"
```

Include dependency graph for view.c:

## Functions

- char * AK_check_view_name (char *name)

  *Function that checks if the name of the view already exists in AK_view table.*
- int AK_get_view_object_id (char *name)

  *Function that finds an object's id by its name.*
- char * AK_get_view_query (char *name)

  *Function that returns a query by its name.*
- char * AK_get_relation_expression (char *name)

  *Function that 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)

  *Function that 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_object_id (int obj_id)

  *Function that removes the view by its object id.*
- int AK_view_rename (char *name, char *new_name)

  *Function that renames a view (based on it's name) from "name" to "new_name".*
- int AK_view_remove_by_name (char *name)

  *Function that removes the view by its name by identifying the view's id and passing id to AK_view_remove_by_↩ object_id.*
- int AK_view_change_query (char *name, char *query, char *rel_exp)

  *Function that changes the query from a view (determined by it's name) to "query".*
- int AK_test_get_view_data (char *rel_exp)

  *Function that shows the data from test view query. Only for test purpose.*
- TestResult AK_view_test ()

  *A testing function for view.c functions.*

### 7.111.1 Detailed Description

Provides functions for views

### 7.111.2 Function Documentation

#### 7.111.2.1 AK_check_view_name()

```
char* AK_check_view_name (
            char * name )
```

Function that checks if the name of the view already exists in AK_view table.

**Author**

Sara Kisic

**Parameters**

| | |
|---|---|
| *name* | Name of the view |

**Returns**

EXIT_ERROR if the name already exists or name

#### 7.111.2.2 AK_get_relation_expression()

```
char* AK_get_relation_expression (
            char * name )
```

Function that returns a relation expression by its name param name name of the view.

**Author**

Danko Sačer

**Returns**

rel_exp string or EXIT_ERROR

#### 7.111.2.3 AK_get_view_object_id()

```
int AK_get_view_object_id (
            char * name )
```

Function that finds an object's id by its name.

**Author**

Kresimir Ivkovic

**Parameters**

| | |
|---|---|
| *name* | name of the view |

**Returns**

View's id or EXIT_ERROR

#### 7.111.2.4 AK_get_view_query()

```
char* AK_get_view_query (
            char * name )
```

Function that returns a query by its name.

**Author**

Danko Sačer

**Parameters**

| | |
|---|---|
| *name* | name of the view |

**Returns**

query string or EXIT_ERROR

#### 7.111.2.5 AK_test_get_view_data()

```
int AK_test_get_view_data (
            char * rel_exp )
```

Function that shows the data from test view query. Only for test purpose.

**Author**

Darko Hranic

**Parameters**

| | |
|---|---|
| *rel_exp* | conditions as string |

#### 7.111.2.6 AK_view_add()

```
int AK_view_add (
            char * name,
            char * query,
            char * rel_exp,
            int set_id )
```

Function that 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**

| | |
|---|---|
| *name* | name og the view |
| *query* | query of the view |
| *rel_exp* | relation expression of the view |
| *set_id* | id of view |

**Returns**

Id of the newly inserted view

#### 7.111.2.7 AK_view_change_query()

```
int AK_view_change_query (
            char * name,
            char * query,
            char * rel_exp )
```

Function that changes the query from a view (determined by it's name) to "query".

**Author**

Kresimir Ivkovic

**Parameters**

| | |
|---|---|
| *name* | of the query |
| *query* | new query of the view |
| *rel_exp* | relation expression of the view |

**Returns**

error or success

#### 7.111.2.8 AK_view_remove_by_name()

```
int AK_view_remove_by_name (
            char * name )
```

Function that removes the view by its name by identifying the view's id and passing id to AK_view_remove_by_↩ object_id.

Function that 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**

| | |
|---|---|
| *name* | name of the view |

**Returns**

Result of AK_view_remove_by_object_id or EXIT_ERROR if no id is found

#### 7.111.2.9 AK_view_remove_by_object_id()

```
int AK_view_remove_by_object_id (
            int obj_id )
```

Function that removes the view by its object id.

**Author**

Kresimir Ivkovic

**Parameters**

| | |
|---|---|
| *obj↩ _id* | object id of the view |

**Returns**

Result of AK_delete_row for the view (success or error)

#### 7.111.2.10 AK_view_rename()

```
int AK_view_rename (
            char * name,
            char * new_name )
```

Function that renames a view (based on it's name) from "name" to "new_name".

**Author**

Kresimir Ivkovic

**Parameters**

| | |
|---|---|
| *name* | name of the view |
| *new_name* | new name of the view |

**Returns**

error or success

#### 7.111.2.11 AK_view_test()

```
TestResult AK_view_test ( )
```

A testing function for view.c functions.

**Author**

Kresimir Ivkovic, updated by Lidija Lastavec

## 7.112 sql/view.h File Reference

```
#include "../auxi/test.h"
#include "../file/table.h"
#include "../file/id.h"
#include "../file/fileio.h"
#include "../auxi/mempro.h"
```

Include dependency graph for view.h: This graph shows which files directly or indirectly include this file:

### Functions

- char * AK_check_view_name (char *name)

  *Function that checks if the name of the view already exists in AK_view table.*
- int AK_view_add (char *name, char *query, char *rel_exp, int set_id)

  *Function that 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_name (char *name)

  *Function that 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)

  *Function that 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)

  *Function that changes the query from a view (determined by it's name) to "query".*
- TestResult AK_view_test ()

  *A testing function for view.c functions.*
- char * AK_get_view_query (char *name)

  *Function that returns a query by its name.*

## 7.112.1 Function Documentation

### 7.112.1.1 AK_check_view_name()

```
char* AK_check_view_name (
            char * name )
```

Function that checks if the name of the view already exists in AK_view table.

**Author**

Sara Kisic

**Parameters**

| | |
|---|---|
| *name* | Name of the view |

**Returns**

EXIT_ERROR if the name already exists or name

### 7.112.1.2 AK_get_view_query()

```
char* AK_get_view_query (
            char * name )
```

Function that returns a query by its name.

**Author**

Danko Sačer

**Parameters**

| | |
|---|---|
| *name* | name of the view |

**Returns**

query string or EXIT_ERROR

#### 7.112.1.3 AK_view_add()

```
int AK_view_add (
             char * name,
             char * query,
             char * rel_exp,
             int set_id )
```

Function that 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**

| | |
|---|---|
| *name* | name og the view |
| *query* | query of the view |
| *rel_exp* | relation expression of the view |
| *set_id* | id of view |

**Returns**

Id of the newly inserted view

#### 7.112.1.4 AK_view_change_query()

```
int AK_view_change_query (
             char * name,
             char * query,
             char * rel_exp )
```

Function that changes the query from a view (determined by it's name) to "query".

**Author**

Kresimir Ivkovic

**Parameters**

| | |
|---|---|
| *name* | of the query |
| *query* | new query of the view |
| *rel_exp* | relation expression of the view |

**Returns**

error or success

#### 7.112.1.5 AK_view_remove_by_name()

```
int AK_view_remove_by_name (
            char * name )
```

Function that 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**

| | |
|---|---|
| *name* | name of the view |

**Returns**

Result of AK_view_remove_by_obj_id or EXIT_ERROR if no id is found

Function that 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**

| | |
|---|---|
| *name* | name of the view |

**Returns**

Result of AK_view_remove_by_object_id or EXIT_ERROR if no id is found

#### 7.112.1.6 AK_view_rename()

```
int AK_view_rename (
            char * name,
            char * new_name )
```

Function that renames a view (based on it's name) from "name" to "new_name".

**Author**

Kresimir Ivkovic

**Parameters**

| | |
|---|---|
| *name* | name of the view |
| *new_name* | new name of the view |

**Returns**

error or success

#### 7.112.1.7 AK_view_test()

```
TestResult AK_view_test ( )
```

A testing function for view.c functions.

**Author**

Kresimir Ivkovic, updated by Lidija Lastavec

## 7.113 tools/comments.py File Reference

### Namespaces

- comments

### 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 = [ ]

### 7.113.1 Detailed Description

//!

## 7.114 tools/getFiles.sh File Reference

### 7.114.1 Detailed Description

Finding all files that ends with extension .py or .c and storing them into file.txt

## 7.115 tools/parseC.sh File Reference

### 7.115.1 Detailed Description

Parsing every C file

## 7.116 tools/parsePy.sh File Reference

### 7.116.1 Detailed Description

Parsing every Py file

## 7.117 tools/updateVersion.sh File Reference

### 7.117.1 Detailed Description

Updating project version

## 7.118 trans/transaction.c File Reference

```
#include "transaction.h"
```

Include dependency graph for transaction.c:

## Functions

- int AK_memory_block_hash (int blockMemoryAddress)

  *Function that calculates the 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)

  *Function that searches for a existing entry in hash list of active blocks.*
- AK_transaction_elem_P AK_search_empty_link_for_hook (int blockAddress)

  *Function that 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)

  *Function that adds an element to the doubly linked list.*
- int AK_delete_hash_entry_list (int blockAddress)

  *Function that 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)

  *Function that 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)

  *Function that 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)

  *Function that, 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)

  *Function that 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)

  *Function that appends all addresses affected by the transaction.*
- int AK_execute_commands (command *commandArray, int lengthOfArray)

  *Function 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)

  *Function that is the thread start point all relevant functions. 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.*
- int AK_transaction_manager (command *commandArray, int lengthOfArray)

  *Function 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_transction 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 the lock is released.*
- void AK_transaction_finished ()

  *Function that is called when some transaction is finished.*
- void AK_all_transactions_finished ()

  *Function that 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.*
- TestResult 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

### 7.118.1 Detailed Description

Defines functions for transaction execution

### 7.118.2 Function Documentation

#### 7.118.2.1 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**

| | |
|---|---|
| *memoryAddress* | integer representation of memory address. |
| *type* | of lock issued to the provided memory address. |
| *transactionId* | integer representation of transaction id. |

**Returns**

OK or NOT_OK based on the success of the function.

**########\n# Lock Granted after wait#\n#----------—#\n# Lock ID:lu TYPE:i #\n#----------—#\n# LockedAddress:i #\n############################\n\n", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress);** */

**##################\n# Lock Granted #\n#----------—#\n# Lock ID:lu TYPE:i #\n#----------—#\n# LockedAddress:i #\n############################\n\n", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress);** */

#### 7.118.2.2 AK_add_hash_entry_list()

```
AK_transaction_elem_P AK_add_hash_entry_list (
            int blockAddress,
            int type )
```

Function that adds an element to the doubly linked list.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |
| *type* | of lock issued to the provided memory address. |

**Returns**

pointer to the newly created doubly linked element.

### 7.118.2.3 AK_add_lock()

```
AK_transaction_lock_elem_P AK_add_lock (
            AK_transaction_elem_P HashList,
            int type,
            pthread_t transactionId )
```

Function that adds an element to the locks doubly linked list.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *memoryAddress* | integer representation of memory address. |
| *type* | of lock issued to the provided memory address. |
| *transactionId* | integer representation of transaction id. |

**Returns**

pointer to the newly created Locks doubly linked element.

### 7.118.2.4 AK_all_transactions_finished()

```
void AK_all_transactions_finished ( )
```

Function that is called when all transactions are finished.

**Author**

Ivan Pusic

### 7.118.2.5 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**

| | |
|---|---|
| *memoryAddress* | integer representation of memory address. |
| *type* | of lock issued to the provided memory address. |
| *transactionId* | integer representation of transaction id. |

**Returns**

pointer to the newly created Locks doubly linked element.

#### 7.118.2.6 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**

| | |
|---|---|
| *transaction_data* | Data for executing transaction |

**Returns**

Exit status (OK or NOT_OK)

#### 7.118.2.7 AK_delete_hash_entry_list()

```
int AK_delete_hash_entry_list (
            int blockAddress )
```

Function that deletes a specific element in the lockTable doubly linked list.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |

**Returns**

integer OK or NOT_OK based on success of finding the specific element in the list.

### 7.118.2.8 AK_delete_lock_entry_list()

```
int AK_delete_lock_entry_list (
            int blockAddress,
            pthread_t id )
```

Function that deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |
| *id* | integer representation of transaction id. |

**Returns**

int OK or NOT_OK based on success of finding the specific element in the list.

### 7.118.2.9 AK_execute_commands()

```
int AK_execute_commands (
            command * commandArray,
            int lengthOfArray )
```

Function 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**

| | |
|---|---|
| *commandArray* | array filled with commands that need to be secured using transactions |
| *lengthOfArray* | length of commandArray |
| *transactionId* | associated with the transaction |

**Returns**

ABORT or COMMIT based on the success of the function.

#### 7.118.2.10 AK_execute_transaction()

```
void* AK_execute_transaction (
            void * params )
```

Function that is the thread start point all relevant functions. It acts as an intermediary between the main thread and other threads.

**Author**

Frane Jakelić updated by Ivan Pusic

**Parameters**

| | |
|---|---|
| *data* | transmitted to the thread from the main thread |

#### 7.118.2.11 AK_get_memory_blocks()

```
int AK_get_memory_blocks (
            char * tblName,
            AK_memoryAddresses_link addressList )
```

Function that appends all addresses affected by the transaction.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *addressList* | pointer to the linked list where the addresses are stored. |
| *tblName* | table name used in the transaction |

**Returns**

OK or NOT_OK based on the success of the function.

#### 7.118.2.12 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**

| | |
|---|---|
| *noticeType* | Type of action (event) |

#### 7.118.2.13 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

#### 7.118.2.14 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

#### 7.118.2.15 AK_isLock_waiting()

```
int AK_isLock_waiting (
            AK_transaction_elem_P lockHolder,
            int type,
            pthread_t transactionId,
            AK_transaction_lock_elem_P lock )
```

Function that, 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**

| | |
|---|---|
| *lockHolder* | pointer to the hash list entry that is entitled to the specific memory address. |
| *type* | of lock issued to the provided memory address. |
| *transaction↩ Id* | integer representation of transaction id. |
| *lock* | 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.

#### 7.118.2.16 AK_lock_released()

```
void AK_lock_released ( )
```

Function which is called when the lock is released.

**Author**

Ivan Pusic

#### 7.118.2.17 AK_memory_block_hash()

```
int AK_memory_block_hash (
            int blockMemoryAddress )
```

Function that calculates the 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**

| | |
|---|---|
| *blockMemoryAddress* | integer representation of memory address, the hash value is calculated from this parameter. |

**Returns**

integer containing the hash value of the passed memory address

#### 7.118.2.18 AK_on_all_transactions_end()

```
void AK_on_all_transactions_end ( )
```

Function for handling event when all transactions are finished.

**Author**

Ivan Pusic

#### 7.118.2.19 AK_on_lock_release()

```
void AK_on_lock_release ( )
```

Function for handling event when one of lock is released.

**Author**

Ivan Pusic

#### 7.118.2.20 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**

| | |
|---|---|
| *observer* | Observer type |
| *observable* | Observable type |
| *type* | Type of observable who sent some notice |

#### 7.118.2.21 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**

| | |
|---|---|
| *transaction_thread* | Thread ID of transaction which is finished |

#### 7.118.2.22 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**

| | |
|---|---|
| *adresses* | linked list of memory addresses locked by the transaction. |
| *transaction↩ Id* | integer representation of transaction id. |

#### 7.118.2.23 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**

| | |
|---|---|
| *transaction_thread* | Active thread to delete |

**Returns**

Exit status (OK or NOT_OK)

#### 7.118.2.24 AK_search_empty_link_for_hook()

```
AK_transaction_elem_P AK_search_empty_link_for_hook (
            int blockAddress )
```

Function that searches for a empty link for new active block, helper method in case of address collision.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |

**Returns**

pointer to empty location to store new active address

#### 7.118.2.25 AK_search_existing_link_for_hook()

```
AK_transaction_elem_P AK_search_existing_link_for_hook (
            int blockAddress )
```

Function that searches for a existing entry in hash list of active blocks.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |

**Returns**

pointer to the existing hash list entry

### 7.118.2.26 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 )
```

Function that searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *memoryAddress* | integer representation of memory address. |
| *id* | integer representation of transaction id. |

**Returns**

NULL pointer if the element is not found otherwise it returns a pointer to the found element

### 7.118.2.27 AK_test_Transaction()

```
TestResult AK_test_Transaction ( )
```

### 7.118.2.28 AK_transaction_finished()

```
void AK_transaction_finished ( )
```

Function that is called when some transaction is finished.

**Author**

Ivan Pusic

#### 7.118.2.29 AK_transaction_manager()

```
int AK_transaction_manager (
            command * commandArray,
            int lengthOfArray )
```

Function 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**

| | |
|---|---|
| *commandArray* | array filled with commands that need to be secured using transactions |
| *lengthOfArray* | length of commandArray |

#### 7.118.2.30 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**

| | |
|---|---|
| *observable_transaction* | Observable type instance |
| *observer* | Observer instance |

**Returns**

Exit status (OK or NOT_OK)

#### 7.118.2.31 AK_transaction_unregister_observer()

```
int AK_transaction_unregister_observer (
            AK_observable_transaction * observable_transaction,
            AK_observer * observer )
```

Function for unregistering observer from AK_observable_transction type.

**Author**

Ivan Pusic

**Parameters**

| | |
|---|---|
| *observable_transaction* | Observable type instance |
| *observer* | Observer instance |

**Returns**

Exit status (OK or NOT_OK)

#### 7.118.2.32 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**

| | |
|---|---|
| *observer_lock* | Observer type instance |

### 7.118.3 Variable Documentation

#### 7.118.3.1 accessLockMutex

```
pthread_mutex_t accessLockMutex = PTHREAD_MUTEX_INITIALIZER
```

#### 7.118.3.2 acquireLockMutex

```
pthread_mutex_t acquireLockMutex = PTHREAD_MUTEX_INITIALIZER
```

#### 7.118.3.3 activeThreads

```
pthread_t activeThreads[MAX_ACTIVE_TRANSACTIONS_COUNT]
```

#### 7.118.3.4 activeTransactionsCount

```
int activeTransactionsCount = 0
```

#### 7.118.3.5 cond_lock

```
pthread_cond_t cond_lock = PTHREAD_COND_INITIALIZER
```

#### 7.118.3.6 endTransationTestLockMutex

```
pthread_mutex_t endTransationTestLockMutex = PTHREAD_MUTEX_INITIALIZER
```

#### 7.118.3.7 LockTable

```
AK_transaction_list LockTable[NUMBER_OF_KEYS]
```

#### 7.118.3.8 newTransactionLockMutex

```
pthread_mutex_t newTransactionLockMutex = PTHREAD_MUTEX_INITIALIZER
```

#### 7.118.3.9 observable_transaction

```
AK_observable_transaction* observable_transaction
```

#### 7.118.3.10 transactionsCount

```
int transactionsCount = 0
```

## 7.119 trans/transaction.h File Reference

```
#include <pthread.h>
#include "../auxi/test.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)

  *Function that calculates the 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)

  *Function that searches for a existing entry in hash list of active blocks.*
- AK_transaction_elem_P AK_search_empty_link_for_hook (int)

  *Function that 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)

  *Function that adds an element to the doubly linked list.*
- int AK_delete_hash_entry_list (int)

  *Function that 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)

  *Function that 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)

  *Function that 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)

  *Function that, 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)

  *Function that 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)

  *Function that appends all addresses affected by the transaction.*
- int AK_execute_commands (command *, int)

  *Function 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 *)

  *Function that is the thread start point all relevant functions. It acts as an intermediary between the main thread and other threads.*
- int AK_transaction_manager (command *, int)

  *Function that receives all the data and gives an id to that data and starts a thread that executes the transaction.*
- TestResult 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 the lock is released.*
- void AK_transaction_finished ()

  *Function that is called when some transaction is finished.*
- void AK_all_transactions_finished ()

  *Function that 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_transction 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.*

## 7.119.1 Detailed Description

Header file that contains data structures,functions and defines for the transaction execution

## 7.119.2 Typedef Documentation

### 7.119.2.1 AK_memoryAddresses

```
typedef struct memoryAddresses AK_memoryAddresses
```

### 7.119.2.2 AK_memoryAddresses_link

```
typedef struct memoryAddresses* AK_memoryAddresses_link
```

#### 7.119.2.3 AK_observable_transaction

```
typedef struct observable_transaction_struct AK_observable_transaction
```

#### 7.119.2.4 AK_observer_lock

```
typedef struct observer_lock AK_observer_lock
```

#### 7.119.2.5 AK_thread_Container

```
typedef struct threadContainer AK_thread_Container
```

#### 7.119.2.6 AK_thread_elem

```
typedef struct threadContainer* AK_thread_elem
```

#### 7.119.2.7 AK_transaction_data

```
typedef struct transactionData AK_transaction_data
```

#### 7.119.2.8 AK_transaction_elem

```
typedef struct transaction_list_elem AK_transaction_elem
```

#### 7.119.2.9 AK_transaction_elem_P

```
typedef struct transaction_list_elem* AK_transaction_elem_P
```

#### 7.119.2.10 AK_transaction_list

```
typedef struct transaction_list_head AK_transaction_list
```

#### 7.119.2.11 AK_transaction_lock_elem

```
typedef struct transaction_locks_list_elem AK_transaction_lock_elem
```

#### 7.119.2.12 AK_transaction_lock_elem_P

```
typedef struct transaction_locks_list_elem* AK_transaction_lock_elem_P
```

### 7.119.3 Enumeration Type Documentation

#### 7.119.3.1 NoticeType

```
enum NoticeType
```

Enumeration which define notice types for transactions.

**Author**

Ivan Pusic

**Enumerator**

| | |
|---|---|
| AK_LOCK_RELEASED | |
| AK_TRANSACTION_FINISHED | |
| AK_ALL_TRANSACTION_FINISHED | |

### 7.119.4 Function Documentation

#### 7.119.4.1 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**

| | |
|---|---|
| *memoryAddress* | integer representation of memory address. |
| *type* | of lock issued to the provided memory address. |
| *transactionId* | integer representation of transaction id. |

**Returns**

OK or NOT_OK based on the success of the function.

**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**

| | |
|---|---|
| *memoryAddress* | integer representation of memory address. |
| *type* | of lock issued to the provided memory address. |
| *transactionId* | integer representation of transaction id. |

**Returns**

OK or NOT_OK based on the success of the function.

**########\n# Lock Granted after wait#\n#----------—#\n# Lock ID:lu TYPE:i #\n#----------—#\n# LockedAddress:i #\n############################\n\n", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress);** ∗/

**##################\n# Lock Granted #\n#----------—#\n# Lock ID:lu TYPE:i #\n#----------—#\n# LockedAddress:i #\n############################\n\n", (unsigned long)lock->TransactionId, lock->lock_type, memoryAddress);** ∗/

### 7.119.4.2 AK_add_hash_entry_list()

```
AK_transaction_elem_P AK_add_hash_entry_list (
            int blockAddress,
            int type )
```

Function that adds an element to the doubly linked list.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |
| *type* | of lock issued to the provided memory address. |

**Returns**

pointer to the newly created doubly linked element.

#### 7.119.4.3 AK_add_lock()

```
AK_transaction_lock_elem_P AK_add_lock (
            AK_transaction_elem_P HashList,
            int type,
            pthread_t transactionId )
```

Function that adds an element to the locks doubly linked list.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *memoryAddress* | integer representation of memory address. |
| *type* | of lock issued to the provided memory address. |
| *transactionId* | integer representation of transaction id. |

**Returns**

pointer to the newly created Locks doubly linked element.

#### 7.119.4.4 AK_all_transactions_finished()

```
void AK_all_transactions_finished ( )
```

Function that is called when all transactions are finished.

**Author**

Ivan Pusic

### 7.119.4.5 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**

| | |
|---|---|
| *memoryAddress* | integer representation of memory address. |
| *type* | of lock issued to the provided memory address. |
| *transactionId* | integer representation of transaction id. |

**Returns**

pointer to the newly created Locks doubly linked element.

### 7.119.4.6 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**

| | |
|---|---|
| *transaction_data* | Data for executing transaction |

**Returns**

Exit status (OK or NOT_OK)

#### 7.119.4.7 AK_delete_hash_entry_list()

```
int AK_delete_hash_entry_list (
            int blockAddress )
```

Function that deletes a specific element in the lockTable doubly linked list.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |

**Returns**

integer OK or NOT_OK based on success of finding the specific element in the list.

#### 7.119.4.8 AK_delete_lock_entry_list()

```
int AK_delete_lock_entry_list (
            int blockAddress,
            pthread_t id )
```

Function that deletes a specific entry in the Locks doubly linked list using the transaction id as it's key.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |
| *id* | integer representation of transaction id. |

**Returns**

int OK or NOT_OK based on success of finding the specific element in the list.

#### 7.119.4.9 AK_execute_commands()

```
int AK_execute_commands (
            command * commandArray,
            int lengthOfArray )
```

Function 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**

| | |
|---|---|
| *commandArray* | array filled with commands that need to be secured using transactions |
| *lengthOfArray* | length of commandArray |
| *transactionId* | associated with the transaction |

**Returns**

ABORT or COMMIT based on the success of the function.

**Author**

Frane Jakelić updated by Ivan Pusic

**Todo** Check multithreading, check if it's working correctly

**Parameters**

| | |
|---|---|
| *commandArray* | array filled with commands that need to be secured using transactions |
| *lengthOfArray* | length of commandArray |
| *transactionId* | associated with the transaction |

**Returns**

ABORT or COMMIT based on the success of the function.

#### 7.119.4.10 AK_execute_transaction()

```
void* AK_execute_transaction (
            void * params )
```

Function that is the thread start point all relevant functions. It acts as an intermediary between the main thread and other threads.

**Author**

Frane Jakelić updated by Ivan Pusic

**Parameters**

| | |
|---|---|
| *data* | transmitted to the thread from the main thread |

#### 7.119.4.11 AK_get_memory_blocks()

```
int AK_get_memory_blocks (
            char * tblName,
            AK_memoryAddresses_link addressList )
```

Function that appends all addresses affected by the transaction.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *addressList* | pointer to the linked list where the addresses are stored. |
| *tblName* | table name used in the transaction |

**Returns**

OK or NOT_OK based on the success of the function.

#### 7.119.4.12 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**

| | |
|---|---|
| *noticeType* | Type of action (event) |

#### 7.119.4.13 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

#### 7.119.4.14 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

#### 7.119.4.15 AK_isLock_waiting()

```
int AK_isLock_waiting (
            AK_transaction_elem_P lockHolder,
            int type,
            pthread_t transactionId,
            AK_transaction_lock_elem_P lock )
```

Function that, 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**

| | |
|---|---|
| *lockHolder* | pointer to the hash list entry that is entitled to the specific memory address. |
| *type* | of lock issued to the provided memory address. |
| *transaction↩ Id* | integer representation of transaction id. |
| *lock* | 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.

#### 7.119.4.16 AK_lock_released()

```
void AK_lock_released ( )
```

Function which is called when the lock is released.

**Author**

Ivan Pusic

#### 7.119.4.17 AK_memory_block_hash()

```
int AK_memory_block_hash (
            int blockMemoryAddress )
```

Function that calculates the 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**

| | |
|---|---|
| *blockMemoryAddress* | integer representation of memory address, the hash value is calculated from this parameter. |

**Returns**

integer containing the hash value of the passed memory address

**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**

| | |
|---|---|
| *blockMemoryAddress* | integer representation of memory address, the hash value is calculated from this parameter. |

**Returns**

integer containing the hash value of the passed memory address

#### 7.119.4.18 AK_on_all_transactions_end()

```
void AK_on_all_transactions_end ( )
```

Function for handling event when all transactions are finished.

**Author**

Ivan Pusic

#### 7.119.4.19 AK_on_lock_release()

```
void AK_on_lock_release ( )
```

Function for handling event when one of lock is released.

**Author**

Ivan Pusic

#### 7.119.4.20 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**

| | |
|---|---|
| *observer* | Observer type |
| *observable* | Observable type |
| *type* | Type of observable who sent some notice |

#### 7.119.4.21 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**

| | |
|---|---|
| *transaction_thread* | Thread ID of transaction which is finished |

#### 7.119.4.22 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**

| | |
|---|---|
| *adresses* | linked list of memory addresses locked by the transaction. |
| *transaction↩ Id* | integer representation of transaction id. |

#### 7.119.4.23 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**

| | |
|---|---|
| *transaction_thread* | Active thread to delete |

**Returns**

Exit status (OK or NOT_OK)

#### 7.119.4.24 AK_search_empty_link_for_hook()

```
AK_transaction_elem_P AK_search_empty_link_for_hook (
            int blockAddress )
```

Function that searches for a empty link for new active block, helper method in case of address collision.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |

**Returns**

pointer to empty location to store new active address

#### 7.119.4.25 AK_search_existing_link_for_hook()

```
AK_transaction_elem_P AK_search_existing_link_for_hook (
            int blockAddress )
```

Function that searches for a existing entry in hash list of active blocks.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *blockAddress* | integer representation of memory address. |

**Returns**

pointer to the existing hash list entry

#### 7.119.4.26 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 )
```

Function that searches for a specific entry in the Locks doubly linked list using the transaction id as it's key.

**Author**

Frane Jakelić

**Parameters**

| | |
|---|---|
| *memoryAddress* | integer representation of memory address. |
| *id* | integer representation of transaction id. |

**Returns**

NULL pointer if the element is not found otherwise it returns a pointer to the found element

#### 7.119.4.27 AK_test_Transaction()

```
TestResult AK_test_Transaction ( )
```

#### 7.119.4.28 AK_transaction_finished()

```
void AK_transaction_finished ( )
```

Function that is called when some transaction is finished.

**Author**

Ivan Pusic

#### 7.119.4.29 AK_transaction_manager()

```
int AK_transaction_manager (
            command * commandArray,
            int lengthOfArray )
```

Function 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**

| | |
|---|---|
| *commandArray* | array filled with commands that need to be secured using transactions |
| *lengthOfArray* | length of commandArray |

#### 7.119.4.30 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**

| | |
|---|---|
| *observable_transaction* | Observable type instance |
| *observer* | Observer instance |

**Returns**

Exit status (OK or NOT_OK)

#### 7.119.4.31 AK_transaction_unregister_observer()

```
int AK_transaction_unregister_observer (
            AK_observable_transaction * observable_transaction,
            AK_observer * observer )
```

Function for unregistering observer from AK_observable_transction type.

**Author**

Ivan Pusic

**Parameters**

| | |
|---|---|
| *observable_transaction* | Observable type instance |
| *observer* | Observer instance |

**Returns**

Exit status (OK or NOT_OK)

#### 7.119.4.32 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**

| | |
|---|---|
| *observer_lock* | Observer type instance |

# Index


_dictionary_, 15
    hash, 15
    key, 15
    n, 16
    size, 16
    val, 16
_file_metadata, 16
    checksum, 16
    new_name, 17
    new_path, 17
    old_name, 17
    old_path, 17
_line_status_
    iniparser.c, 144
_notifyDetails, 17
    message, 17
    type, 18

ABORT
    constants.h, 113
accessLockMutex
    transaction.c, 684
acquireLockMutex
    transaction.c, 684
activeThreads
    transaction.c, 684
activeTransactionsCount
    transaction.c, 685
add
    bucket_elem, 47
    list_structure_ad, 56
addBlock
    struct_add, 70
address
    AK_block, 21
    AK_tuple_dict, 45
    transaction_list_elem, 75
address_from
    table_addresses, 72
address_to
    table_addresses, 72
adresa
    memoryAddresses, 58
agg_task
    AK_agg_value, 19
AGG_TASK_AVG
    aggregation.h, 505
AGG_TASK_AVG_COUNT
    aggregation.h, 505
AGG_TASK_AVG_SUM
    aggregation.h, 505
AGG_TASK_COUNT
    aggregation.h, 505
AGG_TASK_GROUP
    aggregation.h, 506
AGG_TASK_MAX
    aggregation.h, 506
AGG_TASK_MIN
    aggregation.h, 506
AGG_TASK_SUM
    aggregation.h, 506
aggregation.c
    AK_agg_input_add, 499
    AK_agg_input_add_to_beginning, 500
    AK_agg_input_fix, 500
    AK_agg_input_init, 501
    AK_aggregation, 501
    AK_aggregation_test, 502
    AK_header_size, 502
    AK_search_unsorted, 503
aggregation.h
    AGG_TASK_AVG, 505
    AGG_TASK_AVG_COUNT, 505
    AGG_TASK_AVG_SUM, 505
    AGG_TASK_COUNT, 505
    AGG_TASK_GROUP, 506
    AGG_TASK_MAX, 506
    AGG_TASK_MIN, 506
    AGG_TASK_SUM, 506
    AK_agg_input_add, 506
    AK_agg_input_add_to_beginning, 507
    AK_agg_input_fix, 507
    AK_agg_input_init, 508
    AK_aggregation, 508
    AK_aggregation_test, 509
    AK_header_size, 509
aiBlocks
    search_result, 68
aiSearch_attributes
    search_result, 68
aiTuple_addresses
    search_result, 68
AK_acquire_lock
    transaction.c, 671
    transaction.h, 690
AK_add_hash_entry_list
    transaction.c, 672
    transaction.h, 691
AK_add_lock

transaction.c, 673
transaction.h, 692
AK_add_reference
reference.c, 581
reference.h, 588
AK_add_succesor
auxiliary.h, 87
AK_add_to_bitmap_index
bitmap.c, 301
bitmap.h, 308
AK_add_to_redolog
redo_log.c, 495
redo_log.h, 497
AK_add_to_redolog_select
redo_log.c, 495
redo_log.h, 497
AK_add_user_to_group
privileges.c, 623
privileges.h, 634
AK_add_vertex
auxiliary.h, 87
AK_agg_input, 18
attributes, 18
counter, 18
tasks, 19
AK_agg_input_add
aggregation.c, 499
aggregation.h, 506
AK_agg_input_add_to_beginning
aggregation.c, 500
aggregation.h, 507
AK_agg_input_fix
aggregation.c, 500
aggregation.h, 507
AK_agg_input_init
aggregation.c, 501
aggregation.h, 508
AK_agg_value, 19
agg_task, 19
att_name, 19
data, 20
AK_aggregation
aggregation.c, 501
aggregation.h, 508
AK_aggregation_test
aggregation.c, 502
aggregation.h, 509
AK_ALL_TRANSACTION_FINISHED
transaction.h, 690
AK_all_transactions_finished
observable_transaction_struct, 61
transaction.c, 673
transaction.h, 692
AK_allocate_block_activity_modes
dbman.c, 219
AK_allocate_blocks
dbman.c, 219
dbman.h, 241
AK_allocation_set_mode
dbman.h, 241
AK_ALLOCATION_TABLE_SIZE
dbman.h, 239
AK_allocationbit
dbman.h, 257
AK_allocationbit_test
dbman.c, 220
dbman.h, 241
AK_allocationtable_dump
dbman.c, 220
dbman.h, 242
AK_allocationtable_test
dbman.c, 220
dbman.h, 242
AK_archive_log
archive_log.c, 483
archive_log.h, 484
AK_bitmap_test
bitmap.c, 302
bitmap.h, 309
AK_BLOBS_PATH
configuration.h, 106
AK_block, 20
address, 21
AK_free_space, 21
chained_with, 21
data, 21
header, 21
last_tuple_dict_id, 21
tuple_dict, 21
type, 22
AK_block_activity, 22
block_lock, 23
locked_for_reading, 23
locked_for_writing, 23
reading_done, 23
thread_holding_lock, 23
writing_done, 24
AK_block_activity_info
dbman.h, 257
AK_block_sort
filesort.c, 291
filesort.h, 295
AK_blocktable, 24
allocationtable, 24
bittable, 24
last_allocated, 24
last_initialized, 25
ltime, 25
prepared, 25
AK_blocktable_dump
dbman.c, 220
dbman.h, 242
AK_blocktable_flush
dbman.c, 221
dbman.h, 242
AK_blocktable_get

dbman.c, 221
dbman.h, 243
AK_btree_create
btree.c, 316
btree.h, 324
AK_btree_delete
btree.c, 316
btree.h, 324
AK_btree_insert
btree.c, 317
btree.h, 324
AK_btree_search_delete
btree.c, 317
btree.h, 325
AK_btree_test
btree.c, 318
btree.h, 325
AK_cache_AK_malloc
memoman.c, 423
memoman.h, 434
AK_cache_block
memoman.c, 423
memoman.h, 434
AK_cache_result
memoman.c, 424
memoman.h, 435
AK_calloc
mempro.c, 164
mempro.h, 181
AK_change_hash_info
hash.c, 330
hash.h, 337
AK_chars_num_from_number
auxiliary.h, 88
AK_check_arithmetic_statement
expression_check.c, 513
expression_check.h, 516
AK_check_attributes
redo_log.c, 495
redo_log.h, 497
AK_check_constraint
check_constraint.c, 567
AK_check_constraint_name
constraint_names.c, 572
constraint_names.h, 573
AK_check_constraint_not_null
nnull.c, 575
nnull.h, 578
AK_check_constraint_test
check_constraint.c, 567
check_constraint.h, 570
AK_check_constraints
theta_join.c, 548
theta_join.h, 551
AK_check_folder_archivelog
archive_log.c, 483
AK_check_folder_blobs
blobs.c, 259
blobs.h, 265
AK_check_for_writes
mempro.c, 164
mempro.h, 181
AK_check_function_arguments
function.c, 607
function.h, 613
AK_check_function_arguments_type
function.c, 608
function.h, 614
AK_check_group_privilege
privileges.c, 623
privileges.h, 635
AK_check_if_row_satisfies_expression
expression_check.c, 514
expression_check.h, 517
AK_check_privilege
privileges.c, 624
privileges.h, 635
AK_check_redo_log_select
redo_log.c, 495
redo_log.h, 498
AK_check_regex_expression
expression_check.c, 514
expression_check.h, 518
AK_check_regex_operator_expression
expression_check.c, 515
expression_check.h, 518
AK_check_tables_scheme
table.c, 370
table.h, 384
tableOld.c, 397
tableOld.h, 410
AK_check_user_privilege
privileges.c, 624
privileges.h, 636
AK_check_view_name
view.c, 659
view.h, 665
AK_clear_all_newline
blobs.c, 259
blobs.h, 265
AK_command
command.c, 556
command.h, 558
AK_command_recovery_struct, 25
arguments, 26
condition, 26
finished, 26
operation, 26
table_name, 26
AK_command_struct, 26
id_command, 27
parameters, 27
tblName, 27
AK_compare
rel_eq_assoc.c, 450
rel_eq_assoc.h, 452

AK_concat
blobs.c, 259
blobs.h, 265
AK_config
iniparser.c, 152
iniparser.h, 162
AK_constraint_between_test
between.c, 559
between.h, 563
AK_constraint_names_test
constraint_names.c, 573
constraint_names.h, 574
AK_CONSTRAINTS_BEWTEEN
constants.h, 113
AK_CONSTRAINTS_CHECK_CONSTRAINT
constants.h, 113
AK_CONSTRAINTS_DEFAULT
constants.h, 113
AK_CONSTRAINTS_FOREIGN_KEY
constants.h, 113
AK_CONSTRAINTS_INDEX
constants.h, 113
AK_CONSTRAINTS_NOT_NULL
constants.h, 114
AK_CONSTRAINTS_PRIMARY_KEY
constants.h, 114
AK_CONSTRAINTS_UNIQUE
constants.h, 114
AK_convert_type
auxiliary.h, 88
AK_copy
blobs.c, 259
blobs.h, 265
AK_copy_block_projection
projection.c, 533
projection.h, 539
AK_copy_blocks_join
nat_join.c, 522
nat_join.h, 525
AK_copy_header
dbman.c, 221
dbman.h, 243
AK_create_block_header
projection.c, 534
projection.h, 540
AK_create_create_table_parameter
table.c, 371
table.h, 385
tableOld.c, 397
tableOld.h, 411
AK_create_hash_index
hash.c, 331
hash.h, 338
AK_create_header
dbman.c, 222
dbman.h, 244
AK_create_header_name
projection.c, 535
projection.h, 540
AK_create_Index
bitmap.c, 303
bitmap.h, 310
AK_create_Index_Table
bitmap.c, 303
bitmap.h, 310
AK_create_join_block_header
nat_join.c, 523
nat_join.h, 526
AK_create_List_Address_Test
bitmap.h, 311
AK_create_lock
transaction.c, 673
transaction.h, 692
AK_create_new_transaction_thread
transaction.c, 674
transaction.h, 693
AK_create_table
table.c, 371
table.h, 385
tableOld.c, 397
tableOld.h, 411
AK_create_table_parameter
table.h, 384
tableOld.h, 410
AK_create_table_struct, 27
name, 27
type, 27
AK_create_test_table_assistant
test.c, 203
AK_create_test_table_course
test.c, 203
AK_create_test_table_department
test.c, 203
AK_create_test_table_employee
test.c, 204
AK_create_test_table_professor
test.c, 204
AK_create_test_table_professor2
test.c, 204
AK_create_test_table_student
test.c, 205
AK_create_test_tables
test.c, 205
test.h, 213
AK_create_theta_join_header
theta_join.c, 548
theta_join.h, 551
AK_custom_action
observable.c, 195
AK_CUSTOM_FIRST
observable.h, 200
AK_custom_register_observer
observable.c, 196
TypeObservable, 80
AK_CUSTOM_SECOND
observable.h, 200

AK_custom_unregister_observer
observable.c, 196
TypeObservable, 80
AK_db_cache, 28
cache, 28
next_replace, 28
AK_dbg_messg
debug.c, 130
debug.h, 133
AK_deallocate_search_result
filesearch.c, 286
filesearch.h, 289
AK_debmod_calloc
mempro.c, 164
mempro.h, 182
AK_debmod_d
mempro.c, 165
mempro.h, 182
AK_debmod_die
mempro.c, 165
mempro.h, 183
AK_debmod_dv
mempro.c, 166
mempro.h, 183
AK_debmod_enter_critical_sec
mempro.c, 166
mempro.h, 183
AK_debmod_free
mempro.c, 166
mempro.h, 184
AK_debmod_fstack_pop
mempro.c, 167
mempro.h, 184
AK_debmod_fstack_push
mempro.c, 167
mempro.h, 185
AK_debmod_func_add
mempro.c, 168
mempro.h, 185
AK_debmod_func_get_name
mempro.c, 168
mempro.h, 186
AK_debmod_func_id
mempro.c, 169
mempro.h, 186
AK_debmod_function_current
mempro.c, 169
mempro.h, 187
AK_debmod_function_epilogue
mempro.c, 170
mempro.h, 187
AK_debmod_function_prologue
mempro.c, 170
mempro.h, 188
AK_debmod_init
mempro.c, 171
mempro.h, 188
AK_debmod_leave_critical_sec
mempro.c, 171
mempro.h, 188
AK_debmod_log_memory_alloc
mempro.c, 171
mempro.h, 189
AK_DEBMOD_MAX_FUNC_NAME
mempro.h, 179
AK_DEBMOD_MAX_FUNCTIONS
mempro.h, 179
AK_DEBMOD_MAX_WRITE_DETECTIONS
mempro.h, 179
AK_DEBMOD_ON
mempro.h, 179
AK_DEBMOD_PAGES_NUM
mempro.h, 180
AK_DEBMOD_PRINT
mempro.h, 180
AK_debmod_print_function_use
mempro.c, 172
mempro.h, 189
AK_DEBMOD_STACKSIZE
mempro.h, 180
AK_DEBMOD_STATE
mempro.h, 193
AK_debmod_state, 29
alloc_owner, 29
dirty, 30
free_owner, 30
fstack_items, 30
fstack_size, 30
func_used_by, 30
function, 30
init, 30
last_function_id, 30
nomi, 31
page, 31
page_size, 31
print, 31
ready, 31
real, 31
used, 31
AK_define_tarjan_graph
auxiliary.h, 89
AK_Delete_All_elementsAd
index.c, 344
index.h, 353
AK_delete_bitmap_index
bitmap.c, 304
bitmap.h, 311
AK_delete_block
dbman.c, 223
dbman.h, 244
AK_delete_check_constraint
check_constraint.c, 567
check_constraint.h, 570
AK_delete_constraint_between
between.c, 560
between.h, 563

AK_delete_constraint_not_null
nnull.c, 576
nnull.h, 578
AK_delete_constraint_unique
unique.c, 597
unique.h, 599
AK_Delete_elementAd
index.c, 344
index.h, 353
AK_delete_extent
dbman.c, 223
dbman.h, 245
AK_delete_hash_entry_list
transaction.c, 674
transaction.h, 693
AK_delete_hash_index
hash.c, 331
hash.h, 338
AK_delete_in_hash_index
hash.c, 331
hash.h, 338
AK_Delete_L3
auxiliary.h, 89
AK_delete_lock_entry_list
transaction.c, 675
transaction.h, 694
AK_delete_row
fileio.c, 270
fileio.h, 276
reference.h, 589
AK_delete_row_by_id
fileio.c, 270
fileio.h, 277
AK_delete_row_from_block
fileio.c, 270
fileio.h, 277
AK_delete_segment
dbman.c, 224
dbman.h, 245
AK_delete_update_segment
fileio.c, 271
fileio.h, 277
AK_DeleteAll_L3
auxiliary.h, 90
AK_destroy_critical_section
auxiliary.h, 90
AK_destroy_observable
Observable, 59
AK_destroy_observer
Observer, 62
AK_determine_header_type
projection.c, 535
projection.h, 541
AK_difference
difference.c, 510
difference.h, 512
AK_drop
drop.c, 602
drop.h, 605
AK_drop_arguments
drop.h, 605
AK_drop_help_function
drop.c, 603
AK_drop_test
drop.c, 603
drop.h, 606
AK_elem_hash_value
hash.c, 332
hash.h, 339
AK_End_L2
auxiliary.h, 91
AK_enter_critical_section
auxiliary.h, 91
AK_EPI
mempro.h, 180
AK_execute_commands
transaction.c, 675
transaction.h, 694
AK_execute_rel_eq
query_optimization.c, 444
query_optimization.h, 447
AK_execute_transaction
transaction.c, 676
transaction.h, 695
AK_expression_check_test
expression_check.c, 515
expression_check.h, 519
AK_File_Metadata
blobs.h, 264
AK_File_Metadata_malloc
blobs.c, 260
blobs.h, 266
AK_fileio_test
fileio.c, 271
fileio.h, 278
AK_files_test
files.c, 282
files.h, 284
AK_filesearch_test
filesearch.c, 286
filesearch.h, 289
AK_filesort_test
filesort.c, 292
filesort.h, 295
AK_find_AK_free_space
memoman.c, 424
memoman.h, 435
AK_find_available_result_block
memoman.c, 425
memoman.h, 436
AK_find_delete_in_hash_index
hash.c, 332
hash.h, 339
AK_find_in_hash_index
hash.c, 333
hash.h, 340

AK_find_table_address
between.c, 560
between.h, 564
AK_First_L2
auxiliary.h, 91
AK_flush_cache
memoman.c, 425
memoman.h, 436
AK_folder_exists
blobs.c, 260
blobs.h, 266
AK_fread
mempro.c, 172
AK_free
mempro.c, 173
mempro.h, 190
AK_free_space
AK_block, 21
AK_function_add
function.c, 608
function.h, 614
AK_function_arguments_add
function.c, 609
function.h, 615
AK_function_arguments_remove_by_obj_id
function.c, 609
function.h, 615
AK_function_change_return_type
function.c, 610
function.h, 616
AK_function_remove_by_name
function.c, 610
function.h, 616
AK_function_remove_by_obj_id
function.c, 611
function.h, 617
AK_function_rename
function.c, 611
function.h, 617
AK_function_test
function.c, 612
function.h, 618
AK_fwrite
mempro.c, 173
AK_generate_result_id
memoman.c, 425
memoman.h, 436
AK_get_allocation_set
dbman.c, 224
dbman.h, 246
AK_get_array_perms
auxiliary.h, 92
AK_get_attr_index
table.c, 372
table.h, 386
tableOld.c, 398
tableOld.h, 412
AK_get_attr_name
table.c, 372
table.h, 386
tableOld.c, 398
tableOld.h, 412
AK_get_Attribute
bitmap.c, 305
bitmap.h, 312
AK_get_attribute
bitmap.c, 304
bitmap.h, 311
AK_get_block
memoman.c, 425
memoman.h, 436
AK_get_column
table.c, 373
table.h, 387
tableOld.c, 399
tableOld.h, 413
AK_get_extent
dbman.c, 225
dbman.h, 246
AK_Get_First_elementAd
index.c, 345
index.h, 354
AK_get_function_obj_id
function.c, 612
function.h, 618
AK_get_hash_info
hash.c, 333
hash.h, 340
AK_get_header
table.c, 373
table.h, 387
tableOld.c, 399
tableOld.h, 413
AK_get_header_number
filesort.c, 292
filesort.h, 296
AK_get_id
id.c, 298
id.h, 300
AK_get_index_addresses
memoman.c, 426
memoman.h, 437
AK_get_index_header
index.c, 345
AK_get_index_num_records
index.c, 346
index.h, 354
AK_get_index_segment_addresses
memoman.c, 427
memoman.h, 438
AK_get_index_tuple
index.c, 346
index.h, 355
AK_get_insert_header
insert.c, 619
insert.h, 620

AK_Get_Last_elementAd
index.c, 347
index.h, 355
AK_get_memory_blocks
transaction.c, 676
transaction.h, 696
AK_get_message
observable.c, 196
TypeObservable, 80
AK_Get_Next_elementAd
index.c, 347
index.h, 356
AK_get_nth_main_bucket_add
hash.c, 334
hash.h, 341
AK_get_num_of_tuples
filesort.c, 292
filesort.h, 296
AK_get_num_records
table.c, 374
table.h, 388
tableOld.c, 400
tableOld.h, 414
AK_get_observer_by_id
Observable, 59
AK_get_operator
projection.c, 536
projection.h, 541
AK_Get_Position_Of_elementAd
index.c, 348
index.h, 356
AK_Get_Previous_elementAd
index.c, 348
index.h, 357
AK_get_reference
reference.c, 581
reference.h, 589
AK_get_relation_expression
view.c, 660
AK_get_row
table.c, 374
table.h, 388
tableOld.c, 400
tableOld.h, 414
AK_get_segment_addresses
memoman.c, 427
memoman.h, 438
AK_get_segment_addresses_internal
memoman.c, 427
memoman.h, 439
AK_get_system_table_address
memoman.c, 428
AK_get_table_addresses
memoman.c, 428
memoman.h, 439
AK_get_table_atribute_types
test.c, 205
test.h, 214
AK_get_table_id
id.c, 298
AK_get_table_obj_id
table.c, 375
table.h, 389
tableOld.c, 401
tableOld.h, 415
AK_get_timestamp
archive_log.c, 483
archive_log.h, 485
AK_get_total_headers
filesort.c, 292
filesort.h, 296
AK_get_tuple
table.c, 375
table.h, 389
tableOld.c, 401
tableOld.h, 415
AK_get_view_object_id
view.c, 660
AK_get_view_query
view.c, 660
view.h, 665
AK_GetNth_L2
auxiliary.h, 93
AK_grant_privilege_group
privileges.c, 625
privileges.h, 636
AK_grant_privilege_user
privileges.c, 625
privileges.h, 637
AK_graph
auxiliary.h, 86
AK_group_add
privileges.c, 626
privileges.h, 637
AK_group_get_id
privileges.c, 626
privileges.h, 638
AK_group_remove_by_name
privileges.c, 627
privileges.h, 638
AK_group_rename
privileges.c, 627
privileges.h, 639
AK_GUID
blobs.c, 260
blobs.h, 266
AK_handle_observable_transaction_action
transaction.c, 676
transaction.h, 696
AK_hash_test
hash.c, 334
hash.h, 341
AK_header, 32
att_name, 32
constr_code, 32
constr_name, 33

integrity, 33
type, 33
AK_header_size
aggregation.c, 502
aggregation.h, 509
AK_id_test
id.c, 299
id.h, 300
AK_if_exist
drop.c, 603
drop.h, 606
AK_If_ExistOp
bitmap.c, 305
bitmap.h, 312
AK_increase_extent
dbman.c, 225
dbman.h, 247
AK_index_table_exist
index.c, 349
index.h, 357
AK_index_test
index.c, 349
index.h, 358
AK_inflate_config
iniparser.c, 145
iniparser.h, 154
AK_init_allocation_table
dbman.c, 226
dbman.h, 248
AK_init_block
dbman.c, 226
dbman.h, 248
AK_init_critical_section
auxiliary.h, 94
AK_init_db_file
dbman.c, 226
dbman.h, 248
AK_init_disk_manager
dbman.c, 227
dbman.h, 249
AK_Init_L3
auxiliary.h, 95
AK_init_new_extent
memoman.c, 429
memoman.h, 440
AK_init_observable
observable.c, 196
observable.h, 200
AK_init_observable_transaction
transaction.c, 677
transaction.h, 696
AK_init_observer
observable.c, 196
observable.h, 200
AK_init_observer_lock
transaction.c, 677
transaction.h, 697
AK_init_system_catalog
dbman.c, 227
dbman.h, 249
AK_init_system_tables_catalog
dbman.c, 227
dbman.h, 249
AK_initialize_new_index_segment
files.c, 282
files.h, 284
AK_initialize_new_segment
files.c, 283
files.h, 285
reference.h, 589
AK_InitializelistAd
index.c, 349
index.h, 358
AK_INLINE
mempro.h, 180
AK_insert
insert.c, 619
insert.h, 621
AK_insert_bucket_to_block
hash.c, 334
hash.h, 341
AK_insert_entry
dbman.c, 229
dbman.h, 251
AK_insert_in_hash_index
hash.c, 335
hash.h, 342
AK_Insert_New_Element
fileio.c, 271
fileio.h, 278
reference.h, 590
AK_Insert_New_Element_For_Update
fileio.c, 272
fileio.h, 279
reference.h, 591
AK_Insert_NewelementAd
index.c, 350
index.h, 358
AK_insert_row
fileio.c, 273
fileio.h, 279
reference.h, 591
AK_insert_row_to_block
fileio.c, 273
fileio.h, 280
AK_insert_test
insert.c, 620
insert.h, 621
AK_InsertAfter_L2
auxiliary.h, 95
AK_InsertAtBegin_L3
auxiliary.h, 96
AK_InsertAtEnd_L3
auxiliary.h, 96
AK_InsertBefore_L2
auxiliary.h, 97

AK_intersect
intersect.c, 519
intersect.h, 521
AK_IsEmpty_L2
auxiliary.h, 97
AK_isLock_waiting
transaction.c, 677
transaction.h, 697
AK_join
nat_join.c, 523
nat_join.h, 527
AK_leave_critical_section
auxiliary.h, 98
AK_list
auxiliary.h, 86
AK_list_elem
auxiliary.h, 86
AK_lo_export
blobs.c, 260
blobs.h, 266
AK_lo_import
blobs.c, 261
blobs.h, 267
AK_lo_test
blobs.c, 261
blobs.h, 267
AK_lo_unlink
blobs.c, 261
blobs.h, 267
AK_load_chosen_log
recovery.c, 486
recovery.h, 491
AK_load_latest_log
recovery.c, 487
recovery.h, 491
AK_LOCK_RELEASED
transaction.h, 690
AK_lock_released
observable_transaction_struct, 61
transaction.c, 678
transaction.h, 698
AK_malloc
mempro.c, 174
mempro.h, 190
AK_mem_block, 33
block, 34
dirty, 34
timestamp_last_change, 34
timestamp_read, 34
AK_mem_block_modify
memoman.c, 429
memoman.h, 440
AK_memoman_init
memoman.c, 429
memoman.h, 440
AK_memoman_test
memoman.c, 430
memoman.h, 441
AK_memoman_test2
memoman.c, 430
memoman.h, 441
AK_memory_block_hash
transaction.c, 678
transaction.h, 698
AK_memoryAddresses
transaction.h, 688
AK_memoryAddresses_link
transaction.h, 688
AK_mempro_test
mempro.c, 174
mempro.h, 191
AK_memset_int
dbman.c, 229
dbman.h, 251
AK_merge_block_join
nat_join.c, 524
nat_join.h, 527
AK_Metadata
blobs.h, 264
AK_mkdir
blobs.c, 262
blobs.h, 268
AK_new_extent
dbman.c, 230
dbman.h, 252
AK_new_segment
dbman.c, 231
dbman.h, 253
AK_Next_L2
auxiliary.h, 98
AK_nnull_constraint_test
nnull.c, 576
nnull.h, 579
AK_notify
Observer, 62
AK_notify_observer
Observable, 59
AK_notify_observers
Observable, 59
AK_num_attr
table.c, 376
table.h, 390
tableOld.c, 402
tableOld.h, 416
AK_num_index_attr
index.c, 350
index.h, 359
AK_observable
observable.h, 199
AK_observable_pattern
observable.c, 197
observable.h, 200
AK_observable_test
observable.c, 197
observable.h, 201
AK_observable_transaction

transaction.h, 688
AK_observable_type
Observable, 59
AK_ObservableType_Def
Observable, 59
AK_ObservableType_Enum
observable.h, 199
AK_observer
observable.h, 199
AK_observer_lock
transaction.h, 689
AK_observer_type
Observer, 63
AK_observer_type_event_handler
Observer, 63
AK_on_all_transactions_end
transaction.c, 679
transaction.h, 699
AK_on_lock_release
transaction.c, 679
transaction.h, 699
AK_on_observable_notify
transaction.c, 679
transaction.h, 699
AK_on_transaction_end
transaction.c, 680
transaction.h, 700
AK_op_difference_test
difference.c, 511
difference.h, 512
AK_op_intersect_test
intersect.c, 520
intersect.h, 521
AK_op_join_test
nat_join.c, 524
nat_join.h, 528
AK_op_product_test
product.c, 529
product.h, 531
AK_op_projection_test
projection.c, 536
projection.h, 542
AK_op_rename_test
table.c, 376
table.h, 390
tableOld.c, 402
tableOld.h, 416
AK_op_selection_test
selection.c, 544
selection.h, 546
AK_op_selection_test_pattern
selection.c, 544
selection.h, 546
AK_op_theta_join_test
theta_join.c, 549
theta_join.h, 552
AK_op_union_test
union.c, 554
union.h, 555
AK_operand, 35
type, 35
value, 35
AK_perform_operation
projection.c, 536
projection.h, 542
AK_pop_from_stack
auxiliary.h, 98
AK_Previous_L2
auxiliary.h, 99
AK_print_active_functions
mempro.c, 174
mempro.h, 191
AK_print_Att_Test
bitmap.c, 306
bitmap.h, 313
AK_print_block
dbman.c, 231
dbman.h, 253
AK_print_constraints
between.c, 561
AK_print_function_use
mempro.c, 174
mempro.h, 191
AK_print_function_uses
mempro.c, 175
mempro.h, 192
AK_print_Header_Test
bitmap.c, 306
bitmap.h, 313
AK_print_index_table
index.c, 351
index.h, 359
AK_print_optimized_query
query_optimization.c, 445
query_optimization.h, 448
AK_print_rel_eq_assoc
rel_eq_assoc.c, 450
rel_eq_assoc.h, 453
AK_print_rel_eq_comut
rel_eq_comut.c, 454
rel_eq_comut.h, 457
AK_print_rel_eq_projection
rel_eq_projection.c, 459
rel_eq_projection.h, 465
AK_print_rel_eq_selection
rel_eq_selection.c, 471
rel_eq_selection.h, 476
AK_print_row
table.c, 376
table.h, 390
tableOld.c, 403
tableOld.h, 416
AK_print_row_spacer
table.c, 377
table.h, 391
tableOld.c, 403

tableOld.h, 417
AK_print_row_spacer_to_file
table.c, 377
table.h, 391
tableOld.c, 404
tableOld.h, 417
AK_print_row_to_file
table.c, 378
table.h, 392
tableOld.c, 404
tableOld.h, 418
AK_print_table
table.c, 378
table.h, 392
tableOld.c, 405
tableOld.h, 418
AK_print_table_to_file
table.c, 379
table.h, 393
tableOld.c, 405
tableOld.h, 419
AK_printout_redolog
redo_log.c, 496
redo_log.h, 498
AK_privileges_test
privileges.c, 628
privileges.h, 639
AK_PRO
mempro.h, 180
AK_product
product.c, 529
product.h, 531
AK_product_procedure
product.c, 529
product.h, 532
AK_projection
projection.c, 537
projection.h, 543
AK_push_to_stack
auxiliary.h, 99
AK_query_mem, 35
dictionary, 36
parsed, 36
result, 36
AK_query_mem_AK_free
memoman.c, 430
memoman.h, 441
AK_query_mem_AK_malloc
memoman.c, 430
memoman.h, 441
AK_query_mem_dict, 37
dictionary, 37
next_replace, 37
AK_query_mem_lib, 38
next_replace, 38
parsed, 38
AK_query_mem_result, 39
next_replace, 39
results, 39
AK_query_optimization
query_optimization.c, 445
query_optimization.h, 448
AK_query_optimization_test
query_optimization.c, 446
query_optimization.h, 449
AK_read_block
dbman.c, 231
dbman.h, 253
AK_read_block_for_testing
dbman.c, 232
dbman.h, 254
AK_read_constraint_between
between.c, 561
between.h, 564
AK_read_constraint_not_null
nnull.c, 576
nnull.h, 579
AK_read_constraint_unique
unique.c, 597
unique.h, 600
AK_read_metadata
blobs.c, 262
blobs.h, 268
AK_realloc
mempro.c, 175
mempro.h, 192
AK_recover_archive_log
recovery.c, 487
recovery.h, 492
AK_recover_operation
recovery.c, 488
recovery.h, 492
AK_recovery_insert_row
recovery.c, 488
recovery.h, 493
AK_recovery_test
recovery.c, 489
recovery.h, 493
AK_recovery_tokenize
recovery.c, 489
recovery.h, 493
AK_redo_log, 40
command_recovery, 40
number, 40
AK_redo_log_AK_malloc
memoman.c, 431
memoman.h, 442
AK_redolog_commit
redo_log.c, 496
redo_log.h, 498
AK_ref_item, 41
attributes, 41
attributes_number, 41
constraint, 41
parent, 41
parent_attributes, 42

table, 42
type, 42
AK_REFERENCE
constants.h, 114
AK_reference_check_attribute
reference.c, 582
reference.h, 592
AK_reference_check_entry
reference.c, 582
reference.h, 592
AK_reference_check_if_update_needed
reference.c, 583
reference.h, 593
AK_reference_check_restricion
reference.c, 583
reference.h, 593
AK_reference_test
reference.c, 584
reference.h, 594
AK_reference_update
reference.c, 584
reference.h, 594
AK_refresh_cache
memoman.c, 431
memoman.h, 442
AK_register_observer
Observable, 59
AK_register_system_tables
dbman.c, 232
dbman.h, 254
AK_rel_eq_assoc
rel_eq_assoc.c, 451
rel_eq_assoc.h, 453
AK_rel_eq_assoc_test
rel_eq_assoc.c, 451
rel_eq_assoc.h, 454
AK_rel_eq_can_commute
rel_eq_projection.c, 460
rel_eq_projection.h, 465
AK_rel_eq_collect_cond_attributes
rel_eq_projection.c, 460
rel_eq_projection.h, 466
AK_rel_eq_commute_with_theta_join
rel_eq_comut.c, 455
rel_eq_comut.h, 457
AK_rel_eq_comut
rel_eq_comut.c, 455
rel_eq_comut.h, 458
AK_rel_eq_comut_test
rel_eq_comut.c, 456
rel_eq_comut.h, 458
AK_rel_eq_cond_attributes
rel_eq_selection.c, 471
rel_eq_selection.h, 476
AK_rel_eq_get_atrributes_char
rel_eq_selection.c, 472
rel_eq_selection.h, 477
AK_rel_eq_get_attributes
rel_eq_projection.c, 461
rel_eq_projection.h, 466
AK_rel_eq_is_attr_subset
rel_eq_selection.c, 472
rel_eq_selection.h, 479
AK_rel_eq_is_subset
rel_eq_projection.c, 461
rel_eq_projection.h, 467
AK_rel_eq_projection
rel_eq_projection.c, 462
rel_eq_projection.h, 468
AK_rel_eq_projection_attributes
rel_eq_projection.c, 463
rel_eq_projection.h, 469
AK_rel_eq_projection_test
rel_eq_projection.c, 463
rel_eq_projection.h, 469
AK_rel_eq_remove_duplicates
rel_eq_projection.c, 464
rel_eq_projection.h, 470
AK_rel_eq_selection
rel_eq_selection.c, 473
rel_eq_selection.h, 480
AK_rel_eq_selection_test
rel_eq_selection.c, 473
rel_eq_selection.h, 480
AK_rel_eq_share_attributes
rel_eq_selection.c, 474
rel_eq_selection.h, 480
AK_rel_eq_split_condition
rel_eq_selection.c, 474
rel_eq_selection.h, 481
AK_release_locks
transaction.c, 680
transaction.h, 700
AK_release_oldest_cache_block
memoman.c, 431
memoman.h, 442
AK_remove_all_users_from_group
privileges.c, 628
privileges.h, 639
AK_remove_substring
projection.c, 538
projection.h, 543
AK_remove_transaction_thread
transaction.c, 680
transaction.h, 700
AK_remove_user_from_all_groups
privileges.c, 628
privileges.h, 640
AK_rename
table.c, 379
table.h, 393
tableOld.c, 405
tableOld.h, 419
AK_replace_wild_card
expression_check.c, 515
AK_reset_block

filesort.c, 293
filesort.h, 296
AK_results, 42
date_created, 43
free, 43
header, 43
result_block, 43
result_id, 43
result_size, 43
source_table, 43
AK_Retrieve_L2
auxiliary.h, 100
AK_revoke_all_privileges_group
privileges.c, 629
privileges.h, 640
AK_revoke_all_privileges_user
privileges.c, 629
privileges.h, 641
AK_revoke_privilege_group
privileges.c, 630
privileges.h, 641
AK_revoke_privilege_user
privileges.c, 630
privileges.h, 642
AK_run_custom_action
Observable, 60
AK_search_empty_link
auxiliary.h, 100
AK_search_empty_link_for_hook
transaction.c, 681
transaction.h, 701
AK_search_empty_stack_link
auxiliary.h, 101
AK_search_existing_link_for_hook
transaction.c, 681
transaction.h, 701
AK_search_in_stack
auxiliary.h, 101
AK_search_lock_entry_list_by_key
transaction.c, 682
transaction.h, 702
AK_search_unsorted
aggregation.c, 503
filesearch.c, 287
filesearch.h, 290
AK_search_vertex
auxiliary.h, 101
AK_select
select.c, 646
select.h, 647
AK_select_test
select.c, 646
select.h, 647
AK_selection
reference.h, 594
selection.c, 545
selection.h, 547
AK_selection_op_rename
selection.c, 545
AK_sequence_add
sequence.c, 361
sequence.h, 365
AK_sequence_current_value
sequence.c, 361
sequence.h, 366
AK_sequence_get_id
sequence.c, 362
sequence.h, 366
AK_sequence_modify
sequence.c, 362
sequence.h, 366
AK_sequence_next_value
sequence.c, 363
sequence.h, 367
AK_sequence_remove
sequence.c, 363
sequence.h, 368
AK_sequence_rename
sequence.c, 363
sequence.h, 368
AK_sequence_test
sequence.c, 364
sequence.h, 369
AK_set_check_constraint
check_constraint.c, 568
check_constraint.h, 571
AK_set_constraint_between
between.c, 562
between.h, 565
AK_set_constraint_not_null
nnull.c, 577
nnull.h, 580
AK_set_constraint_unique
unique.c, 598
unique.h, 600
AK_set_notify_info_details
observable.c, 197
TypeObservable, 80
AK_Size_L2
auxiliary.h, 102
AK_sort_segment
filesort.c, 293
filesort.h, 297
AK_split_path_file
blobs.c, 262
blobs.h, 268
AK_stack
auxiliary.h, 86
AK_stackHead
auxiliary.h, 86
AK_strcmp
auxiliary.h, 102
AK_succesor
auxiliary.h, 87
AK_synchronization_info, 44
init, 44

ready, 44
AK_table_empty
table.c, 380
table.h, 394
tableOld.c, 406
tableOld.h, 420
AK_table_exist
table.c, 380
tableOld.c, 406
AK_table_test
table.c, 380
table.h, 394
tableOld.c, 407
tableOld.h, 420
AK_tarjan
auxiliary.h, 103
AK_tarjan_test
auxiliary.h, 103
AK_temp_create_table
table.c, 381
table.h, 394
tableOld.c, 407
tableOld.h, 420
AK_test_command
command.c, 557
command.h, 558
AK_test_get_view_data
view.c, 661
AK_test_Transaction
transaction.c, 682
transaction.h, 702
AK_theta_join
theta_join.c, 549
theta_join.h, 552
AK_thread_Container
transaction.h, 689
AK_thread_elem
transaction.h, 689
AK_thread_safe_block_access_test
dbman.c, 233
dbman.h, 255
AK_TRANSACTION
observable.h, 200
AK_transaction_data
transaction.h, 689
AK_transaction_elem
transaction.h, 689
AK_transaction_elem_P
transaction.h, 689
AK_TRANSACTION_FINISHED
transaction.h, 690
AK_transaction_finished
observable_transaction_struct, 61
transaction.c, 682
transaction.h, 702
AK_transaction_list
transaction.h, 689
AK_transaction_lock_elem
transaction.h, 689
AK_transaction_lock_elem_P
transaction.h, 690
AK_transaction_manager
transaction.c, 682
transaction.h, 702
AK_transaction_register_observer
observable_transaction_struct, 61
transaction.c, 683
transaction.h, 703
AK_transaction_unregister_observer
observable_transaction_struct, 61
transaction.c, 683
transaction.h, 703
AK_TRIGGER
observable.h, 200
AK_trigger_add
trigger.c, 648
trigger.h, 654
AK_trigger_edit
trigger.c, 649
trigger.h, 654
AK_trigger_get_conditions
trigger.c, 650
trigger.h, 655
AK_trigger_get_id
trigger.c, 650
trigger.h, 656
AK_trigger_remove_by_name
trigger.c, 651
trigger.h, 656
AK_trigger_remove_by_obj_id
trigger.c, 651
trigger.h, 657
AK_trigger_rename
trigger.c, 652
trigger.h, 657
AK_trigger_save_conditions
trigger.c, 652
trigger.h, 658
AK_trigger_test
trigger.c, 653
trigger.h, 658
AK_tuple_dict, 45
address, 45
size, 45
type, 45
AK_tuple_to_string
table.c, 381
table.h, 395
tableOld.c, 408
tableOld.h, 421
AK_type_size
auxiliary.h, 104
AK_TypeObservable
observable.c, 195
AK_TypeObserver
observable.c, 195

AK_TypeObserver_Second
observable.c, 195
AK_union
union.c, 554
union.h, 555
AK_unique_test
unique.c, 598
unique.h, 601
AK_unregister_observer
Observable, 60
AK_update
bitmap.c, 306
bitmap.h, 314
AK_update_bucket_in_block
hash.c, 335
hash.h, 342
AK_Update_Existing_Element
fileio.c, 274
reference.h, 595
AK_update_row
fileio.c, 274
fileio.h, 280
reference.h, 596
AK_update_row_from_block
fileio.c, 275
fileio.h, 281
AK_user_add
privileges.c, 631
privileges.h, 643
AK_user_check_pass
privileges.c, 631
privileges.h, 643
AK_user_get_id
privileges.c, 632
privileges.h, 644
AK_user_remove_by_name
privileges.c, 632
AK_user_rename
privileges.c, 633
privileges.h, 644
AK_vertex
auxiliary.h, 87
AK_view_add
view.c, 661
view.h, 665
AK_view_change_query
view.c, 662
view.h, 666
AK_view_remove_by_name
view.c, 662
view.h, 667
AK_view_remove_by_object_id
view.c, 663
AK_view_rename
view.c, 663
view.h, 667
AK_view_test
view.c, 664
view.h, 668
AK_write_block
bitmap.h, 314
dbman.c, 234
dbman.h, 256
AK_write_block_for_testing
dbman.c, 234
dbman.h, 256
AK_write_metadata
blobs.c, 262
blobs.h, 268
AK_write_protect
mempro.c, 176
mempro.h, 192
AK_write_unprotect
mempro.c, 176
mempro.h, 193
alloc_owner
AK_debmod_state, 29
allocationAROUND
dbman.h, 241
allocationLOWER
dbman.h, 241
allocationNOMODE
dbman.h, 241
allocationSEQUENCE
dbman.h, 241
allocationtable
AK_blocktable, 24
allocationUPPER
dbman.h, 241
archive_log.c
AK_archive_log, 483
AK_check_folder_archivelog, 483
AK_get_timestamp, 483
archive_log.h
AK_archive_log, 484
AK_get_timestamp, 485
ARCHIVELOG_PATH
configuration.h, 106
arguments
AK_command_recovery_struct, 26
array
transactionData, 79
ASCIILINESZ
iniparser.c, 144
att_name
AK_agg_value, 19
AK_header, 32
intersect_attr, 53
attName
list_structure_ad, 56
ATTR_DELIMITER
constants.h, 114
ATTR_ESCAPE
constants.h, 115
attribute_name
list_node, 54

attributes
AK_agg_input, 18
AK_ref_item, 41
attributes_number
AK_ref_item, 41
auxi/auxiliary.c, 83
auxi/auxiliary.h, 83
auxi/configuration.h, 105
auxi/constants.h, 108
auxi/debug.c, 130
auxi/debug.h, 131
auxi/dictionary.c, 134
auxi/dictionary.h, 138
auxi/iniparser.c, 142
auxi/iniparser.h, 153
auxi/mempro.c, 162
auxi/mempro.h, 177
auxi/observable.c, 194
auxi/observable.h, 198
auxi/test.c, 201
auxi/test.h, 208
auxiliary.h
AK_add_succesor, 87
AK_add_vertex, 87
AK_chars_num_from_number, 88
AK_convert_type, 88
AK_define_tarjan_graph, 89
AK_Delete_L3, 89
AK_DeleteAll_L3, 90
AK_destroy_critical_section, 90
AK_End_L2, 91
AK_enter_critical_section, 91
AK_First_L2, 91
AK_get_array_perms, 92
AK_GetNth_L2, 93
AK_graph, 86
AK_init_critical_section, 94
AK_Init_L3, 95
AK_InsertAfter_L2, 95
AK_InsertAtBegin_L3, 96
AK_InsertAtEnd_L3, 96
AK_InsertBefore_L2, 97
AK_IsEmpty_L2, 97
AK_leave_critical_section, 98
AK_list, 86
AK_list_elem, 86
AK_Next_L2, 98
AK_pop_from_stack, 98
AK_Previous_L2, 99
AK_push_to_stack, 99
AK_Retrieve_L2, 100
AK_search_empty_link, 100
AK_search_empty_stack_link, 101
AK_search_in_stack, 101
AK_search_vertex, 101
AK_Size_L2, 102
AK_stack, 86
AK_stackHead, 86
AK_strcmp, 102
AK_succesor, 87
AK_tarjan, 103
AK_tarjan_test, 103
AK_type_size, 104
AK_vertex, 87
MAX_LOOP_ITERATIONS, 86
MIN, 104
TBL_BOX_OFFSET, 86
testMode, 104

B
btree.h, 323
between.c
AK_constraint_between_test, 559
AK_delete_constraint_between, 560
AK_find_table_address, 560
AK_print_constraints, 561
AK_read_constraint_between, 561
AK_set_constraint_between, 562
between.h
AK_constraint_between_test, 563
AK_delete_constraint_between, 563
AK_find_table_address, 564
AK_read_constraint_between, 564
AK_set_constraint_between, 565
BITCLEAR
dbman.h, 239
bitmap.c
AK_add_to_bitmap_index, 301
AK_bitmap_test, 302
AK_create_Index, 303
AK_create_Index_Table, 303
AK_delete_bitmap_index, 304
AK_get_Attribute, 305
AK_get_attribute, 304
AK_If_ExistOp, 305
AK_print_Att_Test, 306
AK_print_Header_Test, 306
AK_update, 306
bitmap.h
AK_add_to_bitmap_index, 308
AK_bitmap_test, 309
AK_create_Index, 310
AK_create_Index_Table, 310
AK_create_List_Address_Test, 311
AK_delete_bitmap_index, 311
AK_get_Attribute, 312
AK_get_attribute, 311
AK_If_ExistOp, 312
AK_print_Att_Test, 313
AK_print_Header_Test, 313
AK_update, 314
AK_write_block, 314
BITMASK
dbman.h, 239
BITNSLOTS
dbman.h, 239
BITSET

dbman.h, 239
BITSLOT
dbman.h, 239
bittable
AK_blocktable, 24
BITTEST
dbman.h, 240
BLACK
test.h, 209
blobs.c
AK_check_folder_blobs, 259
AK_clear_all_newline, 259
AK_concat, 259
AK_copy, 259
AK_File_Metadata_malloc, 260
AK_folder_exists, 260
AK_GUID, 260
AK_lo_export, 260
AK_lo_import, 261
AK_lo_test, 261
AK_lo_unlink, 261
AK_mkdir, 262
AK_read_metadata, 262
AK_split_path_file, 262
AK_write_metadata, 262
failed, 263
success, 263
blobs.h
AK_check_folder_blobs, 265
AK_clear_all_newline, 265
AK_concat, 265
AK_copy, 265
AK_File_Metadata, 264
AK_File_Metadata_malloc, 266
AK_folder_exists, 266
AK_GUID, 266
AK_lo_export, 266
AK_lo_import, 267
AK_lo_test, 267
AK_lo_unlink, 267
AK_Metadata, 264
AK_mkdir, 268
AK_read_metadata, 268
AK_split_path_file, 268
AK_write_metadata, 268
block
AK_mem_block, 34
BLOCK_CLEAN
constants.h, 115
BLOCK_DIRTY
constants.h, 115
block_lock
AK_block_activity, 23
BLOCK_TYPE_CHAINED
constants.h, 115
BLOCK_TYPE_FREE
constants.h, 115
BLOCK_TYPE_NORMAL
constants.h, 115
blocktable, 46
BLUE
test.h, 209
BOLDBLACK
test.h, 209
BOLDBLUE
test.h, 210
BOLDCYAN
test.h, 210
BOLDGREEN
test.h, 210
BOLDMAGENTA
test.h, 210
BOLDRED
test.h, 210
BOLDWHITE
test.h, 210
BOLDYELLOW
test.h, 210
btree.c
AK_btree_create, 316
AK_btree_delete, 316
AK_btree_insert, 317
AK_btree_search_delete, 317
AK_btree_test, 318
btree_delete, 318
findCorrectNumber, 318
findPointers, 319
findValues, 319
makevalues, 320
searchValue, 321
setNodePointers, 321
btree.h
AK_btree_create, 324
AK_btree_delete, 324
AK_btree_insert, 324
AK_btree_search_delete, 325
AK_btree_test, 325
B, 323
btree_delete, 326
findCorrectNumber, 326
findPointers, 326
findValues, 327
LEAF, 323
makevalues, 328
NODE, 323
ORDER, 323
searchValue, 328
setNodePointers, 329
btree_delete
btree.c, 318
btree.h, 326
btree_node, 46
pointers, 46
values, 47
bucket_elem, 47
add, 47

value, 47
bucket_level
hash_bucket, 51

cache
AK_db_cache, 28
cFiles
comments, 14
chained_with
AK_block, 21
CHAR_IN_LINE
dbman.h, 240
check_constraint.c
AK_check_constraint, 567
AK_check_constraint_test, 567
AK_delete_check_constraint, 567
AK_set_check_constraint, 568
condition_passed, 569
check_constraint.h
AK_check_constraint_test, 570
AK_delete_check_constraint, 570
AK_set_check_constraint, 571
condition_passed, 571
checksum
_file_metadata, 16
command
command.h, 558
command.c
AK_command, 556
AK_test_command, 557
command.h
AK_command, 558
AK_test_command, 558
command, 558
command_recovery
AK_redo_log, 40
comments, 13
cFiles, 14
commentsFile, 14
detectLanguage, 13
getcommentsFiles, 13
makeCommentsFile, 14
pyFiles, 14
commentsFile
comments, 14
COMMIT
constants.h, 116
cond_lock
transaction.c, 685
condition
AK_command_recovery_struct, 26
condition_passed
check_constraint.c, 569
check_constraint.h, 571
configuration.h
AK_BLOBS_PATH, 106
ARCHIVELOG_PATH, 106
DB_FILE, 106
DB_FILE_BLOCKS_NUM, 106
DB_FILE_SIZE, 106
EXTENT_GROWTH_INDEX, 106
EXTENT_GROWTH_TABLE, 106
EXTENT_GROWTH_TEMP, 107
EXTENT_GROWTH_TRANSACTION, 107
INITIAL_EXTENT_SIZE, 107
MAX_EXTENTS_IN_SEGMENT, 107
MAX_FREE_SPACE_SIZE, 107
MAX_LAST_TUPLE_DICT_SIZE_TO_USE, 107
MAX_NUM_OF_BLOCKS, 108
MAX_REDO_LOG_ENTRIES, 108
MAX_REDO_LOG_MEMORY, 108
NUMBER_OF_THREADS, 108
constants.h
ABORT, 113
AK_CONSTRAINTS_BEWTEEN, 113
AK_CONSTRAINTS_CHECK_CONSTRAINT, 113
AK_CONSTRAINTS_DEFAULT, 113
AK_CONSTRAINTS_FOREIGN_KEY, 113
AK_CONSTRAINTS_INDEX, 113
AK_CONSTRAINTS_NOT_NULL, 114
AK_CONSTRAINTS_PRIMARY_KEY, 114
AK_CONSTRAINTS_UNIQUE, 114
AK_REFERENCE, 114
ATTR_DELIMITER, 114
ATTR_ESCAPE, 115
BLOCK_CLEAN, 115
BLOCK_DIRTY, 115
BLOCK_TYPE_CHAINED, 115
BLOCK_TYPE_FREE, 115
BLOCK_TYPE_NORMAL, 115
COMMIT, 116
DATA_BLOCK_SIZE, 116
DATA_ENTRY_SIZE, 116
DELETE, 116
DROP_CONSTRAINT, 116
DROP_FUNCTION, 116
DROP_GROUP, 117
DROP_INDEX, 117
DROP_SEQUENCE, 117
DROP_TABLE, 117
DROP_TRIGGER, 117
DROP_USER, 117
DROP_VIEW, 118
EXCLUSIVE_LOCK, 118
EXIT_ERROR, 118
EXIT_SUCCESS, 118
EXIT_WARNING, 118
FIND, 118
FREE_CHAR, 119
FREE_INT, 119
HASH_BUCKET, 119
HASH_BUCKET_SIZE, 119
INFO_BUCKET, 119
INSERT, 119
MAIN_BUCKET, 120
MAIN_BUCKET_SIZE, 120
MAX_ACTIVE_TRANSACTIONS_COUNT, 120

MAX_ATT_NAME, 120
MAX_ATTRIBUTES, 120
MAX_BLOCKS_CURRENTLY_ACCESSED, 120
MAX_CACHE_MEMORY, 121
MAX_CONSTR_CODE, 121
MAX_CONSTR_NAME, 121
MAX_CONSTRAINTS, 121
MAX_MAIN_BUCKETS, 121
MAX_OBSERVABLE_OBSERVERS, 121
MAX_QUERY_DICT_MEMORY, 122
MAX_QUERY_LIB_MEMORY, 122
MAX_QUERY_RESULT_MEMORY, 122
MAX_TOKENS, 122
MAX_VARCHAR_LENGTH, 122
NEW_ID, 122
NEW_VALUE, 123
NOT_CHAINED, 123
NOT_OK, 123
NULLL, 123
NUM_SYS_TABLES, 123
NUMBER_OF_KEYS, 123
OK, 124
PASS_LOCK_QUEUE, 124
RO_EXCEPT, 124
RO_INTERSECT, 124
RO_NAT_JOIN, 124
RO_PROJECTION, 124
RO_RENAME, 124
RO_SELECTION, 125
RO_THETA_JOIN, 125
RO_UNION, 125
SEARCH_CONSTRAINT, 125
SEGMENT_TYPE_INDEX, 125
SEGMENT_TYPE_SYSTEM_TABLE, 125
SEGMENT_TYPE_TABLE, 125
SEGMENT_TYPE_TEMP, 126
SEGMENT_TYPE_TRANSACTION, 126
SELECT, 126
SEPARATOR, 126
SHARED_LOCK, 126
TEST_MODE_OFF, 126
TEST_MODE_ON, 127
TYPE_ATTRIBS, 127
TYPE_BLOB, 127
TYPE_BOOL, 127
TYPE_CONDITION, 127
TYPE_DATE, 127
TYPE_DATETIME, 128
TYPE_FLOAT, 128
TYPE_INT, 128
TYPE_INTERNAL, 128
TYPE_NUMBER, 128
TYPE_OPERAND, 128
TYPE_OPERATOR, 129
TYPE_TIME, 129
TYPE_VARCHAR, 129
UPDATE, 129
WAIT_FOR_UNLOCK, 129
constr_code
    AK_header, 32
constr_name
    AK_header, 33
constraint
    AK_ref_item, 41
    list_node, 54
constraint_names.c
    AK_check_constraint_name, 572
    AK_constraint_names_test, 573
constraint_names.h
    AK_check_constraint_name, 573
    AK_constraint_names_test, 574
CONSTRAINTS
    debug.h, 133
cost_eval
    rel_eq_assoc.h, 452
cost_eval_t, 48
    data, 48
    value, 48
counter
    AK_agg_input, 18
create_header_test
    test.c, 206
    test.h, 214
custom_observer_event_handler
    observable.c, 197
CYAN
    test.h, 210

data
    AK_agg_value, 20
    AK_block, 21
    cost_eval_t, 48
    list_node, 54
DATA_BLOCK_SIZE
    constants.h, 116
DATA_ENTRY_SIZE
    constants.h, 116
DATA_ROW_SIZE
    filesort.h, 295
DATA_TUPLE_SIZE
    filesort.h, 295
date_created
    AK_results, 43
db
    dbman.h, 257
db_cache
    memoman.h, 443
DB_FILE
    configuration.h, 106
DB_FILE_BLOCKS_NUM
    configuration.h, 106
DB_FILE_BLOCKS_NUM_EX
    dbman.h, 240
DB_FILE_SIZE
    configuration.h, 106
db_file_size
    dbman.h, 257

DB_FILE_SIZE_EX
dbman.h, 240
DB_MAN
debug.h, 133
dbman.c
AK_allocate_block_activity_modes, 219
AK_allocate_blocks, 219
AK_allocationbit_test, 220
AK_allocationtable_dump, 220
AK_allocationtable_test, 220
AK_blocktable_dump, 220
AK_blocktable_flush, 221
AK_blocktable_get, 221
AK_copy_header, 221
AK_create_header, 222
AK_delete_block, 223
AK_delete_extent, 223
AK_delete_segment, 224
AK_get_allocation_set, 224
AK_get_extent, 225
AK_increase_extent, 225
AK_init_allocation_table, 226
AK_init_block, 226
AK_init_db_file, 226
AK_init_disk_manager, 227
AK_init_system_catalog, 227
AK_init_system_tables_catalog, 227
AK_insert_entry, 229
AK_memset_int, 229
AK_new_extent, 230
AK_new_segment, 231
AK_print_block, 231
AK_read_block, 231
AK_read_block_for_testing, 232
AK_register_system_tables, 232
AK_thread_safe_block_access_test, 233
AK_write_block, 234
AK_write_block_for_testing, 234
fileLockMutex, 235
fsize, 234
test_lastCharacterWritten, 235
test_threadSafeBlockAccessSucceeded, 235
dbman.h
AK_allocate_blocks, 241
AK_allocation_set_mode, 241
AK_ALLOCATION_TABLE_SIZE, 239
AK_allocationbit, 257
AK_allocationbit_test, 241
AK_allocationtable_dump, 242
AK_allocationtable_test, 242
AK_block_activity_info, 257
AK_blocktable_dump, 242
AK_blocktable_flush, 242
AK_blocktable_get, 243
AK_copy_header, 243
AK_create_header, 244
AK_delete_block, 244
AK_delete_extent, 245
AK_delete_segment, 245
AK_get_allocation_set, 246
AK_get_extent, 246
AK_increase_extent, 247
AK_init_allocation_table, 248
AK_init_block, 248
AK_init_db_file, 248
AK_init_disk_manager, 249
AK_init_system_catalog, 249
AK_init_system_tables_catalog, 249
AK_insert_entry, 251
AK_memset_int, 251
AK_new_extent, 252
AK_new_segment, 253
AK_print_block, 253
AK_read_block, 253
AK_read_block_for_testing, 254
AK_register_system_tables, 254
AK_thread_safe_block_access_test, 255
AK_write_block, 256
AK_write_block_for_testing, 256
allocationAROUND, 241
allocationLOWER, 241
allocationNOMODE, 241
allocationSEQUENCE, 241
allocationUPPER, 241
BITCLEAR, 239
BITMASK, 239
BITNSLOTS, 239
BITSET, 239
BITSLOT, 239
BITTEST, 240
CHAR_IN_LINE, 240
db, 257
DB_FILE_BLOCKS_NUM_EX, 240
db_file_size, 257
DB_FILE_SIZE_EX, 240
dbmanFileLock, 257
fsize, 256
MAX_BLOCK_INIT_NUM, 240
SEGMENTLENGTH, 240
dbmanFileLock
dbman.h, 257
debug.c
AK_dbg_messg, 130
debug.h
AK_dbg_messg, 133
CONSTRAINTS, 133
DB_MAN, 133
DEBUG_ALL, 131
DEBUG_LEVEL, 132
debug_level, 132
DEBUG_TYPE, 132
debug_type, 132
FILE_MAN, 133
FUNCTIONS, 133
GLOBAL, 133
HIGH, 132

INDICES, 133
LOW, 132
MEMO_MAN, 133
MIDDLE, 132
REDO, 133
REL_EQ, 133
REL_OP, 133
SEQUENCES, 133
TABLES, 133
TRIGGERS, 133
DEBUG_ALL
debug.h, 131
DEBUG_LEVEL, 49
debug.h, 132
debug_level
debug.h, 132
DEBUG_TYPE, 49
debug.h, 132
debug_type
debug.h, 132
DELETE
constants.h, 116
detectLanguage
comments, 13
DICT_INVALID_KEY
dictionary.c, 135
dictionary
AK_query_mem, 36
AK_query_mem_dict, 37
dictionary.h, 139
dictionary.c
DICT_INVALID_KEY, 135
dictionary_del, 135
dictionary_dump, 135
dictionary_get, 136
dictionary_hash, 136
dictionary_new, 137
dictionary_set, 137
dictionary_unset, 138
DICTMINSZ, 135
MAXVALSZ, 135
dictionary.h
dictionary, 139
dictionary_del, 139
dictionary_dump, 140
dictionary_get, 140
dictionary_hash, 141
dictionary_new, 141
dictionary_set, 141
dictionary_unset, 142
dictionary_del
dictionary.c, 135
dictionary.h, 139
dictionary_dump
dictionary.c, 135
dictionary.h, 140
dictionary_get
dictionary.c, 136
dictionary.h, 140
dictionary_hash
dictionary.c, 136
dictionary.h, 141
dictionary_new
dictionary.c, 137
dictionary.h, 141
dictionary_set
dictionary.c, 137
dictionary.h, 141
dictionary_unset
dictionary.c, 138
dictionary.h, 142
DICTMINSZ
dictionary.c, 135
difference.c
AK_difference, 510
AK_op_difference_test, 511
difference.h
AK_difference, 512
AK_op_difference_test, 512
dirty
AK_debmod_state, 30
AK_mem_block, 34
DLLHead
transaction_list_head, 77
DLLLocksHead
transaction_list_elem, 75
dm/dbman.c, 217
dm/dbman.h, 235
drop.c
AK_drop, 602
AK_drop_help_function, 603
AK_drop_test, 603
AK_if_exist, 603
system_catalog, 604
drop.h
AK_drop, 605
AK_drop_arguments, 605
AK_drop_test, 606
AK_if_exist, 606
drop_arguments, 50
next, 50
value, 50
DROP_CONSTRAINT
constants.h, 116
DROP_FUNCTION
constants.h, 116
DROP_GROUP
constants.h, 117
DROP_INDEX
constants.h, 117
DROP_SEQUENCE
constants.h, 117
DROP_TABLE
constants.h, 117
DROP_TRIGGER
constants.h, 117

DROP_USER
constants.h, 117
DROP_VIEW
constants.h, 118

element
hash_bucket, 51
main_bucket, 57
element_ad
index.h, 353
endTransationTestLockMutex
transaction.c, 685
ERROR
observable.c, 195
error_message
query_optimization.c, 446
EXCLUSIVE_LOCK
constants.h, 118
EXIT_ERROR
constants.h, 118
EXIT_SUCCESS
constants.h, 118
EXIT_WARNING
constants.h, 118
expression_check.c
AK_check_arithmetic_statement, 513
AK_check_if_row_satisfies_expression, 514
AK_check_regex_expression, 514
AK_check_regex_operator_expression, 515
AK_expression_check_test, 515
AK_replace_wild_card, 515
expression_check.h
AK_check_arithmetic_statement, 516
AK_check_if_row_satisfies_expression, 517
AK_check_regex_expression, 518
AK_check_regex_operator_expression, 518
AK_expression_check_test, 519
EXTENT_GROWTH_INDEX
configuration.h, 106
EXTENT_GROWTH_TABLE
configuration.h, 106
EXTENT_GROWTH_TEMP
configuration.h, 107
EXTENT_GROWTH_TRANSACTION
configuration.h, 107

failed
blobs.c, 263
file/blobs.c, 258
file/blobs.h, 263
file/fileio.c, 269
file/fileio.h, 275
file/files.c, 281
file/files.h, 283
file/filesearch.c, 285
file/filesearch.h, 288
file/filesort.c, 291
file/filesort.h, 294
file/id.c, 298
file/id.h, 299
file/idx/bitmap.c, 301
file/idx/bitmap.h, 307
file/idx/btree.c, 315
file/idx/btree.h, 322
file/idx/hash.c, 329
file/idx/hash.h, 336
file/idx/index.c, 343
file/idx/index.h, 351
file/sequence.c, 360
file/sequence.h, 364
file/table.c, 369
file/table.h, 382
file/tableOld.c, 396
file/tableOld.h, 409
file/test.c, 202
file/test.h, 213
FILE_MAN
debug.h, 133
fileio.c
AK_delete_row, 270
AK_delete_row_by_id, 270
AK_delete_row_from_block, 270
AK_delete_update_segment, 271
AK_fileio_test, 271
AK_Insert_New_Element, 271
AK_Insert_New_Element_For_Update, 272
AK_insert_row, 273
AK_insert_row_to_block, 273
AK_Update_Existing_Element, 274
AK_update_row, 274
AK_update_row_from_block, 275
fileio.h
AK_delete_row, 276
AK_delete_row_by_id, 277
AK_delete_row_from_block, 277
AK_delete_update_segment, 277
AK_fileio_test, 278
AK_Insert_New_Element, 278
AK_Insert_New_Element_For_Update, 279
AK_insert_row, 279
AK_insert_row_to_block, 280
AK_update_row, 280
AK_update_row_from_block, 281
fileLockMutex
dbman.c, 235
fileMut
files.c, 283
files.c
AK_files_test, 282
AK_initialize_new_index_segment, 282
AK_initialize_new_segment, 283
fileMut, 283
files.h
AK_files_test, 284
AK_initialize_new_index_segment, 284
AK_initialize_new_segment, 285
filesearch.c

AK_deallocate_search_result, 286
AK_filesearch_test, 286
AK_search_unsorted, 287
filesearch.h
AK_deallocate_search_result, 289
AK_filesearch_test, 289
AK_search_unsorted, 290
SEARCH_ALL, 288
SEARCH_NULL, 289
SEARCH_PARTICULAR, 289
SEARCH_RANGE, 289
filesort.c
AK_block_sort, 291
AK_filesort_test, 292
AK_get_header_number, 292
AK_get_num_of_tuples, 292
AK_get_total_headers, 292
AK_reset_block, 293
AK_sort_segment, 293
filesort.h
AK_block_sort, 295
AK_filesort_test, 295
AK_get_header_number, 296
AK_get_num_of_tuples, 296
AK_get_total_headers, 296
AK_reset_block, 296
AK_sort_segment, 297
DATA_ROW_SIZE, 295
DATA_TUPLE_SIZE, 295
FIND
constants.h, 118
findCorrectNumber
btree.c, 318
btree.h, 326
findPointers
btree.c, 319
btree.h, 326
findValues
btree.c, 319
btree.h, 327
finished
AK_command_recovery_struct, 26
free
AK_results, 43
FREE_CHAR
constants.h, 119
FREE_INT
constants.h, 119
free_owner
AK_debmod_state, 30
fsize
dbman.c, 234
dbman.h, 256
fstack_items
AK_debmod_state, 30
fstack_size
AK_debmod_state, 30
func_used_by
AK_debmod_state, 30
function
AK_debmod_state, 30
function.c
AK_check_function_arguments, 607
AK_check_function_arguments_type, 608
AK_function_add, 608
AK_function_arguments_add, 609
AK_function_arguments_remove_by_obj_id, 609
AK_function_change_return_type, 610
AK_function_remove_by_name, 610
AK_function_remove_by_obj_id, 611
AK_function_rename, 611
AK_function_test, 612
AK_get_function_obj_id, 612
function.h
AK_check_function_arguments, 613
AK_check_function_arguments_type, 614
AK_function_add, 614
AK_function_arguments_add, 615
AK_function_arguments_remove_by_obj_id, 615
AK_function_change_return_type, 616
AK_function_remove_by_name, 616
AK_function_remove_by_obj_id, 617
AK_function_rename, 617
AK_function_test, 618
AK_get_function_obj_id, 618
FUNCTIONS
debug.h, 133

get_column_test
test.c, 206
test.h, 215
get_row_attr_data
table.c, 382
table.h, 395
tableOld.c, 408
tableOld.h, 421
get_row_test
test.c, 207
test.h, 215
getcommentsFiles
comments, 13
GLOBAL
debug.h, 133
grandfailure
recovery.c, 490
GREEN
test.h, 211

handle_AK_custom_type
observable.c, 198
handle_transaction_notify
transaction.c, 684
transaction.h, 704
hash
_dictionary_, 15
hash.c
AK_change_hash_info, 330

AK_create_hash_index, 331
AK_delete_hash_index, 331
AK_delete_in_hash_index, 331
AK_elem_hash_value, 332
AK_find_delete_in_hash_index, 332
AK_find_in_hash_index, 333
AK_get_hash_info, 333
AK_get_nth_main_bucket_add, 334
AK_hash_test, 334
AK_insert_bucket_to_block, 334
AK_insert_in_hash_index, 335
AK_update_bucket_in_block, 335
hash.h
AK_change_hash_info, 337
AK_create_hash_index, 338
AK_delete_hash_index, 338
AK_delete_in_hash_index, 338
AK_elem_hash_value, 339
AK_find_delete_in_hash_index, 339
AK_find_in_hash_index, 340
AK_get_hash_info, 340
AK_get_nth_main_bucket_add, 341
AK_hash_test, 341
AK_insert_bucket_to_block, 341
AK_insert_in_hash_index, 342
AK_update_bucket_in_block, 342
HASH_BUCKET
constants.h, 119
hash_bucket, 50
bucket_level, 51
element, 51
hash_bucket_num
hash_info, 52
HASH_BUCKET_SIZE
constants.h, 119
hash_info, 51
hash_bucket_num, 52
main_bucket_num, 52
modulo, 52
header
AK_block, 21
AK_results, 43
HIGH
debug.h, 132

id.c
AK_get_id, 298
AK_get_table_id, 298
AK_id_test, 299
id.h
AK_get_id, 300
AK_id_test, 300
ID_START_VALUE, 300
id_command
AK_command_struct, 27
ID_START_VALUE
id.h, 300
implemented
TestResult, 73
index
Vertex, 82
index.c
AK_Delete_All_elementsAd, 344
AK_Delete_elementAd, 344
AK_Get_First_elementAd, 345
AK_get_index_header, 345
AK_get_index_num_records, 346
AK_get_index_tuple, 346
AK_Get_Last_elementAd, 347
AK_Get_Next_elementAd, 347
AK_Get_Position_Of_elementAd, 348
AK_Get_Previous_elementAd, 348
AK_index_table_exist, 349
AK_index_test, 349
AK_InitializelistAd, 349
AK_Insert_NewelementAd, 350
AK_num_index_attr, 350
AK_print_index_table, 351
index.h
AK_Delete_All_elementsAd, 353
AK_Delete_elementAd, 353
AK_Get_First_elementAd, 354
AK_get_index_num_records, 354
AK_get_index_tuple, 355
AK_Get_Last_elementAd, 355
AK_Get_Next_elementAd, 356
AK_Get_Position_Of_elementAd, 356
AK_Get_Previous_elementAd, 357
AK_index_table_exist, 357
AK_index_test, 358
AK_InitializelistAd, 358
AK_Insert_NewelementAd, 358
AK_num_index_attr, 359
AK_print_index_table, 359
element_ad, 353
list_ad, 353
list_structure_ad, 353
indexTd
struct_add, 71
INDICES
debug.h, 133
INFO
observable.c, 195
INFO_BUCKET
constants.h, 119
INI_INVALID_KEY
iniparser.c, 144
iniparser.c
_line_status_, 144
AK_config, 152
AK_inflate_config, 145
ASCIILINESZ, 144
INI_INVALID_KEY, 144
iniparser_AK_freedict, 145
iniparser_dump, 145
iniparser_dump_ini, 146
iniparser_dumpsection_ini, 146

iniparser_find_entry, 147
iniparser_getboolean, 147
iniparser_getdouble, 148
iniparser_getint, 148
iniparser_getnsec, 149
iniparser_getseckeys, 149
iniparser_getsecname, 150
iniparser_getsecnkeys, 150
iniparser_getstring, 151
iniparser_load, 151
iniparser_set, 151
iniparser_unset, 152
iniParserMutex, 152
LINE_COMMENT, 145
LINE_EMPTY, 145
LINE_ERROR, 145
LINE_SECTION, 145
line_status, 144
LINE_UNPROCESSED, 145
LINE_VALUE, 145
iniparser.h
AK_config, 162
AK_inflate_config, 154
iniparser_AK_freedict, 154
iniparser_dump, 154
iniparser_dump_ini, 155
iniparser_dumpsection_ini, 155
iniparser_find_entry, 156
iniparser_getboolean, 156
iniparser_getdouble, 157
iniparser_getint, 157
iniparser_getnsec, 158
iniparser_getseckeys, 159
iniparser_getsecname, 159
iniparser_getsecnkeys, 160
iniparser_getstring, 160
iniparser_load, 160
iniparser_set, 161
iniparser_unset, 161
iniparser_AK_freedict
iniparser.c, 145
iniparser.h, 154
iniparser_dump
iniparser.c, 145
iniparser.h, 154
iniparser_dump_ini
iniparser.c, 146
iniparser.h, 155
iniparser_dumpsection_ini
iniparser.c, 146
iniparser.h, 155
iniparser_find_entry
iniparser.c, 147
iniparser.h, 156
iniparser_getboolean
iniparser.c, 147
iniparser.h, 156
iniparser_getdouble
iniparser.c, 148
iniparser.h, 157
iniparser_getint
iniparser.c, 148
iniparser.h, 157
iniparser_getnsec
iniparser.c, 149
iniparser.h, 158
iniparser_getseckeys
iniparser.c, 149
iniparser.h, 159
iniparser_getsecname
iniparser.c, 150
iniparser.h, 159
iniparser_getsecnkeys
iniparser.c, 150
iniparser.h, 160
iniparser_getstring
iniparser.c, 151
iniparser.h, 160
iniparser_load
iniparser.c, 151
iniparser.h, 160
iniparser_set
iniparser.c, 151
iniparser.h, 161
iniparser_unset
iniparser.c, 152
iniparser.h, 161
iniParserMutex
iniparser.c, 152
init
AK_debmod_state, 30
AK_synchronization_info, 44
init_observable_type
observable.c, 198
init_observer_type
observable.c, 198
init_observer_type_second
observable.c, 198
INITIAL_EXTENT_SIZE
configuration.h, 107
INSERT
constants.h, 119
insert.c
AK_get_insert_header, 619
AK_insert, 619
AK_insert_test, 620
insert.h
AK_get_insert_header, 620
AK_insert, 621
AK_insert_test, 621
insert_data_test
test.c, 207
test.h, 216
integrity
AK_header, 33
intersect.c

AK_intersect, 519
AK_op_intersect_test, 520
intersect.h
AK_intersect, 521
AK_op_intersect_test, 521
intersect_attr, 52
att_name, 53
type, 53
iNum_search_attributes
search_result, 68
iNum_tuple_addresses
search_result, 69
iNum_tuple_attributes
search_result, 69
iSearchType
search_params, 66
isWaiting
transaction_list_elem, 76
transaction_locks_list_elem, 78

key
_dictionary_, 15

last_allocated
AK_blocktable, 24
last_function_id
AK_debmod_state, 30
last_initialized
AK_blocktable, 25
last_tuple_dict_id
AK_block, 21
LEAF
btree.h, 323
lengthOfArray
transactionData, 79
level
root_info, 65
LINE_COMMENT
iniparser.c, 145
LINE_EMPTY
iniparser.c, 145
LINE_ERROR
iniparser.c, 145
LINE_SECTION
iniparser.c, 145
line_status
iniparser.c, 144
LINE_UNPROCESSED
iniparser.c, 145
LINE_VALUE
iniparser.c, 145
link
Stack, 70
Succesor, 71
list_ad
index.h, 353
list_node, 53
attribute_name, 54
constraint, 54
data, 54
next, 54
size, 55
table, 55
type, 55
list_structure_ad, 55
add, 56
attName, 56
index.h, 353
next, 56
list_structure_add, 56
lock_type
transaction_list_elem, 76
transaction_locks_list_elem, 78
locked_for_reading
AK_block_activity, 23
locked_for_writing
AK_block_activity, 23
LockTable
transaction.c, 685
LOW
debug.h, 132
lowLink
Vertex, 82
ltime
AK_blocktable, 25

MAGENTA
test.h, 211
MAIN_BUCKET
constants.h, 120
main_bucket, 57
element, 57
main_bucket_num
hash_info, 52
MAIN_BUCKET_SIZE
constants.h, 120
makeCommentsFile
comments, 14
makevalues
btree.c, 320
btree.h, 328
MAX_ACTIVE_TRANSACTIONS_COUNT
constants.h, 120
MAX_ATT_NAME
constants.h, 120
MAX_ATTRIBUTES
constants.h, 120
MAX_BLOCK_INIT_NUM
dbman.h, 240
MAX_BLOCKS_CURRENTLY_ACCESSED
constants.h, 120
MAX_CACHE_MEMORY
constants.h, 121
MAX_CHILD_CONSTRAINTS
reference.h, 587
MAX_CONSTR_CODE
constants.h, 121
MAX_CONSTR_NAME

constants.h, 121
MAX_CONSTRAINTS
constants.h, 121
MAX_EXTENTS_IN_SEGMENT
configuration.h, 107
MAX_FREE_SPACE_SIZE
configuration.h, 107
MAX_LAST_TUPLE_DICT_SIZE_TO_USE
configuration.h, 107
MAX_LOOP_ITERATIONS
auxiliary.h, 86
MAX_MAIN_BUCKETS
constants.h, 121
MAX_NUM_OF_BLOCKS
configuration.h, 108
MAX_OBSERVABLE_OBSERVERS
constants.h, 121
MAX_PERMUTATION
query_optimization.h, 447
MAX_QUERY_DICT_MEMORY
constants.h, 122
MAX_QUERY_LIB_MEMORY
constants.h, 122
MAX_QUERY_RESULT_MEMORY
constants.h, 122
MAX_REDO_LOG_ENTRIES
configuration.h, 108
MAX_REDO_LOG_MEMORY
configuration.h, 108
MAX_REFERENCE_ATTRIBUTES
reference.h, 587
MAX_TOKENS
constants.h, 122
MAX_VARCHAR_LENGTH
constants.h, 122
MAXVALSZ
dictionary.c, 135
MEMO_MAN
debug.h, 133
memoman.c
AK_cache_AK_malloc, 423
AK_cache_block, 423
AK_cache_result, 424
AK_find_AK_free_space, 424
AK_find_available_result_block, 425
AK_flush_cache, 425
AK_generate_result_id, 425
AK_get_block, 425
AK_get_index_addresses, 426
AK_get_index_segment_addresses, 427
AK_get_segment_addresses, 427
AK_get_segment_addresses_internal, 427
AK_get_system_table_address, 428
AK_get_table_addresses, 428
AK_init_new_extent, 429
AK_mem_block_modify, 429
AK_memoman_init, 429
AK_memoman_test, 430
AK_memoman_test2, 430
AK_query_mem_AK_free, 430
AK_query_mem_AK_malloc, 430
AK_redo_log_AK_malloc, 431
AK_refresh_cache, 431
AK_release_oldest_cache_block, 431
memoman.h
AK_cache_AK_malloc, 434
AK_cache_block, 434
AK_cache_result, 435
AK_find_AK_free_space, 435
AK_find_available_result_block, 436
AK_flush_cache, 436
AK_generate_result_id, 436
AK_get_block, 436
AK_get_index_addresses, 437
AK_get_index_segment_addresses, 438
AK_get_segment_addresses, 438
AK_get_segment_addresses_internal, 439
AK_get_table_addresses, 439
AK_init_new_extent, 440
AK_mem_block_modify, 440
AK_memoman_init, 440
AK_memoman_test, 441
AK_memoman_test2, 441
AK_query_mem_AK_free, 441
AK_query_mem_AK_malloc, 441
AK_redo_log_AK_malloc, 442
AK_refresh_cache, 442
AK_release_oldest_cache_block, 442
db_cache, 443
query_mem, 443
redo_log, 443
memoryAddresses, 57
adresa, 58
nextElement, 58
mempro.c
AK_calloc, 164
AK_check_for_writes, 164
AK_debmod_calloc, 164
AK_debmod_d, 165
AK_debmod_die, 165
AK_debmod_dv, 166
AK_debmod_enter_critical_sec, 166
AK_debmod_free, 166
AK_debmod_fstack_pop, 167
AK_debmod_fstack_push, 167
AK_debmod_func_add, 168
AK_debmod_func_get_name, 168
AK_debmod_func_id, 169
AK_debmod_function_current, 169
AK_debmod_function_epilogue, 170
AK_debmod_function_prologue, 170
AK_debmod_init, 171
AK_debmod_leave_critical_sec, 171
AK_debmod_log_memory_alloc, 171
AK_debmod_print_function_use, 172
AK_fread, 172

AK_free, 173
AK_fwrite, 173
AK_malloc, 174
AK_mempro_test, 174
AK_print_active_functions, 174
AK_print_function_use, 174
AK_print_function_uses, 175
AK_realloc, 175
AK_write_protect, 176
AK_write_unprotect, 176
mempro.h
AK_calloc, 181
AK_check_for_writes, 181
AK_debmod_calloc, 182
AK_debmod_d, 182
AK_debmod_die, 183
AK_debmod_dv, 183
AK_debmod_enter_critical_sec, 183
AK_debmod_free, 184
AK_debmod_fstack_pop, 184
AK_debmod_fstack_push, 185
AK_debmod_func_add, 185
AK_debmod_func_get_name, 186
AK_debmod_func_id, 186
AK_debmod_function_current, 187
AK_debmod_function_epilogue, 187
AK_debmod_function_prologue, 188
AK_debmod_init, 188
AK_debmod_leave_critical_sec, 188
AK_debmod_log_memory_alloc, 189
AK_DEBMOD_MAX_FUNC_NAME, 179
AK_DEBMOD_MAX_FUNCTIONS, 179
AK_DEBMOD_MAX_WRITE_DETECTIONS, 179
AK_DEBMOD_ON, 179
AK_DEBMOD_PAGES_NUM, 180
AK_DEBMOD_PRINT, 180
AK_debmod_print_function_use, 189
AK_DEBMOD_STACKSIZE, 180
AK_DEBMOD_STATE, 193
AK_EPI, 180
AK_free, 190
AK_INLINE, 180
AK_malloc, 190
AK_mempro_test, 191
AK_print_active_functions, 191
AK_print_function_use, 191
AK_print_function_uses, 192
AK_PRO, 180
AK_realloc, 192
AK_write_protect, 192
AK_write_unprotect, 193
NEW, 181
message
_notifyDetails, 17
MIDDLE
debug.h, 132
MIN
auxiliary.h, 104
mm/memoman.c, 422
mm/memoman.h, 432
modulo
hash_info, 52

n
_dictionary_, 16
name
AK_create_table_struct, 27
nat_join.c
AK_copy_blocks_join, 522
AK_create_join_block_header, 523
AK_join, 523
AK_merge_block_join, 524
AK_op_join_test, 524
nat_join.h
AK_copy_blocks_join, 525
AK_create_join_block_header, 526
AK_join, 527
AK_merge_block_join, 527
AK_op_join_test, 528
NEW
mempro.h, 181
NEW_ID
constants.h, 122
new_name
_file_metadata, 17
new_path
_file_metadata, 17
NEW_VALUE
constants.h, 123
newTransactionLockMutex
transaction.c, 685
next
drop_arguments, 50
list_node, 54
list_structure_ad, 56
next_replace
AK_db_cache, 28
AK_query_mem_dict, 37
AK_query_mem_lib, 38
AK_query_mem_result, 39
nextBucket
transaction_list_elem, 76
nextElement
memoryAddresses, 58
Stack, 70
nextLock
transaction_locks_list_elem, 78
nextSuccesor
Succesor, 72
Vertex, 82
nextThread
threadContainer, 74
nextVertex
Vertex, 82
nnull.c
AK_check_constraint_not_null, 575
AK_delete_constraint_not_null, 576

AK_nnull_constraint_test, 576
AK_read_constraint_not_null, 576
AK_set_constraint_not_null, 577
nnull.h
AK_check_constraint_not_null, 578
AK_delete_constraint_not_null, 578
AK_nnull_constraint_test, 579
AK_read_constraint_not_null, 579
AK_set_constraint_not_null, 580
NODE
btree.h, 323
nomi
AK_debmod_state, 31
NOT_CHAINED
constants.h, 123
NOT_OK
constants.h, 123
NoticeType
transaction.h, 690
NotifyDetails
observable.c, 195
notifyDetails
TypeObservable, 80
NotifyType
observable.c, 195
NULLL
constants.h, 123
NUM_SYS_TABLES
constants.h, 123
number
AK_redo_log, 40
NUMBER_OF_KEYS
constants.h, 123
NUMBER_OF_THREADS
configuration.h, 108

Observable, 58
AK_destroy_observable, 59
AK_get_observer_by_id, 59
AK_notify_observer, 59
AK_notify_observers, 59
AK_observable_type, 59
AK_ObservableType_Def, 59
AK_register_observer, 59
AK_run_custom_action, 60
AK_unregister_observer, 60
observer_id_counter, 60
observers, 60
observable
observable_transaction_struct, 62
TypeObservable, 80
TypeObserver, 81
observable.c
AK_custom_action, 195
AK_custom_register_observer, 196
AK_custom_unregister_observer, 196
AK_get_message, 196
AK_init_observable, 196
AK_init_observer, 196
AK_observable_pattern, 197
AK_observable_test, 197
AK_set_notify_info_details, 197
AK_TypeObservable, 195
AK_TypeObserver, 195
AK_TypeObserver_Second, 195
custom_observer_event_handler, 197
ERROR, 195
handle_AK_custom_type, 198
INFO, 195
init_observable_type, 198
init_observer_type, 198
init_observer_type_second, 198
NotifyDetails, 195
NotifyType, 195
WARMING, 195
observable.h
AK_CUSTOM_FIRST, 200
AK_CUSTOM_SECOND, 200
AK_init_observable, 200
AK_init_observer, 200
AK_observable, 199
AK_observable_pattern, 200
AK_observable_test, 201
AK_ObservableType_Enum, 199
AK_observer, 199
AK_TRANSACTION, 200
AK_TRIGGER, 200
observable_transaction, 60
transaction.c, 685
observable_transaction_struct, 61
AK_all_transactions_finished, 61
AK_lock_released, 61
AK_transaction_finished, 61
AK_transaction_register_observer, 61
AK_transaction_unregister_observer, 61
observable, 62
Observer, 62
AK_destroy_observer, 62
AK_notify, 62
AK_observer_type, 63
AK_observer_type_event_handler, 63
observer_id, 63
observer
observer_lock, 64
TypeObserver, 81
observer_id
Observer, 63
observer_id_counter
Observable, 60
observer_lock, 63
observer, 64
transaction_list_elem, 76
observers
Observable, 60
OK
constants.h, 124
old_name

_file_metadata, 17
old_path
_file_metadata, 17
operation
AK_command_recovery_struct, 26
opti/query_optimization.c, 443
opti/query_optimization.h, 446
opti/rel_eq_assoc.c, 449
opti/rel_eq_assoc.h, 451
opti/rel_eq_comut.c, 454
opti/rel_eq_comut.h, 456
opti/rel_eq_projection.c, 458
opti/rel_eq_projection.h, 464
opti/rel_eq_selection.c, 470
opti/rel_eq_selection.h, 475
ORDER
btree.h, 323

page
AK_debmod_state, 31
page_size
AK_debmod_state, 31
parameters
AK_command_struct, 27
parent
AK_ref_item, 41
parent_attributes
AK_ref_item, 42
parsed
AK_query_mem, 36
AK_query_mem_lib, 38
PASS_LOCK_QUEUE
constants.h, 124
pData_lower
search_params, 67
pData_upper
search_params, 67
pointers
btree_node, 46
prepared
AK_blocktable, 25
prevBucket
transaction_list_elem, 76
prevLock
transaction_locks_list_elem, 78
print
AK_debmod_state, 31
privileges.c
AK_add_user_to_group, 623
AK_check_group_privilege, 623
AK_check_privilege, 624
AK_check_user_privilege, 624
AK_grant_privilege_group, 625
AK_grant_privilege_user, 625
AK_group_add, 626
AK_group_get_id, 626
AK_group_remove_by_name, 627
AK_group_rename, 627
AK_privileges_test, 628
AK_remove_all_users_from_group, 628
AK_remove_user_from_all_groups, 628
AK_revoke_all_privileges_group, 629
AK_revoke_all_privileges_user, 629
AK_revoke_privilege_group, 630
AK_revoke_privilege_user, 630
AK_user_add, 631
AK_user_check_pass, 631
AK_user_get_id, 632
AK_user_remove_by_name, 632
AK_user_rename, 633
privileges.h
AK_add_user_to_group, 634
AK_check_group_privilege, 635
AK_check_privilege, 635
AK_check_user_privilege, 636
AK_grant_privilege_group, 636
AK_grant_privilege_user, 637
AK_group_add, 637
AK_group_get_id, 638
AK_group_remove_by_name, 638
AK_group_rename, 639
AK_privileges_test, 639
AK_remove_all_users_from_group, 639
AK_remove_user_from_all_groups, 640
AK_revoke_all_privileges_group, 640
AK_revoke_all_privileges_user, 641
AK_revoke_privilege_group, 641
AK_revoke_privilege_user, 642
AK_user_add, 643
AK_user_check_pass, 643
AK_user_get_id, 644
AK_user_rename, 644
product.c
AK_op_product_test, 529
AK_product, 529
AK_product_procedure, 529
product.h
AK_op_product_test, 531
AK_product, 531
AK_product_procedure, 532
projection.c
AK_copy_block_projection, 533
AK_create_block_header, 534
AK_create_header_name, 535
AK_determine_header_type, 535
AK_get_operator, 536
AK_op_projection_test, 536
AK_perform_operation, 536
AK_projection, 537
AK_remove_substring, 538
projection.h
AK_copy_block_projection, 539
AK_create_block_header, 540
AK_create_header_name, 540
AK_determine_header_type, 541
AK_get_operator, 541
AK_op_projection_test, 542

AK_perform_operation, 542
AK_projection, 543
AK_remove_substring, 543
projection_att
projection_att_struct, 64
projection_att_struct, 64
projection_att, 64
pyFiles
comments, 14

query_mem
memoman.h, 443
query_optimization.c
AK_execute_rel_eq, 444
AK_print_optimized_query, 445
AK_query_optimization, 445
AK_query_optimization_test, 446
error_message, 446
query_optimization.h
AK_execute_rel_eq, 447
AK_print_optimized_query, 448
AK_query_optimization, 448
AK_query_optimization_test, 449
MAX_PERMUTATION, 447

reading_done
AK_block_activity, 23
ready
AK_debmod_state, 31
AK_synchronization_info, 44
real
AK_debmod_state, 31
rec/archive_log.c, 482
rec/archive_log.h, 484
rec/recovery.c, 486
rec/recovery.h, 490
rec/redo_log.c, 494
rec/redo_log.h, 496
recovery.c
AK_load_chosen_log, 486
AK_load_latest_log, 487
AK_recover_archive_log, 487
AK_recover_operation, 488
AK_recovery_insert_row, 488
AK_recovery_test, 489
AK_recovery_tokenize, 489
grandfailure, 490
recovery_insert_row, 489
recovery.h
AK_load_chosen_log, 491
AK_load_latest_log, 491
AK_recover_archive_log, 492
AK_recover_operation, 492
AK_recovery_insert_row, 493
AK_recovery_test, 493
AK_recovery_tokenize, 493
recovery_insert_row
recovery.c, 489
RED
test.h, 211
REDO
debug.h, 133
redo_log
memoman.h, 443
redo_log.c
AK_add_to_redolog, 495
AK_add_to_redolog_select, 495
AK_check_attributes, 495
AK_check_redo_log_select, 495
AK_printout_redolog, 496
AK_redolog_commit, 496
redo_log.h
AK_add_to_redolog, 497
AK_add_to_redolog_select, 497
AK_check_attributes, 497
AK_check_redo_log_select, 498
AK_printout_redolog, 498
AK_redolog_commit, 498
REF_TYPE_CASCADE
reference.h, 587
REF_TYPE_NO_ACTION
reference.h, 587
REF_TYPE_NONE
reference.h, 587
REF_TYPE_RESTRICT
reference.h, 587
REF_TYPE_SET_DEFAULT
reference.h, 587
REF_TYPE_SET_NULL
reference.h, 588
reference.c
AK_add_reference, 581
AK_get_reference, 581
AK_reference_check_attribute, 582
AK_reference_check_entry, 582
AK_reference_check_if_update_needed, 583
AK_reference_check_restricion, 583
AK_reference_test, 584
AK_reference_update, 584
reference.h
AK_add_reference, 588
AK_delete_row, 589
AK_get_reference, 589
AK_initialize_new_segment, 589
AK_Insert_New_Element, 590
AK_Insert_New_Element_For_Update, 591
AK_insert_row, 591
AK_reference_check_attribute, 592
AK_reference_check_entry, 592
AK_reference_check_if_update_needed, 593
AK_reference_check_restricion, 593
AK_reference_test, 594
AK_reference_update, 594
AK_selection, 594
AK_Update_Existing_Element, 595
AK_update_row, 596
MAX_CHILD_CONSTRAINTS, 587

MAX_REFERENCE_ATTRIBUTES, 587
REF_TYPE_CASCADE, 587
REF_TYPE_NO_ACTION, 587
REF_TYPE_NONE, 587
REF_TYPE_RESTRICT, 587
REF_TYPE_SET_DEFAULT, 587
REF_TYPE_SET_NULL, 588
rel/aggregation.c, 499
rel/aggregation.h, 504
rel/difference.c, 510
rel/difference.h, 511
rel/expression_check.c, 512
rel/expression_check.h, 516
rel/intersect.c, 519
rel/intersect.h, 520
rel/nat_join.c, 522
rel/nat_join.h, 525
rel/product.c, 528
rel/product.h, 530
rel/projection.c, 533
rel/projection.h, 538
rel/selection.c, 544
rel/selection.h, 546
rel/theta_join.c, 547
rel/theta_join.h, 550
rel/union.c, 553
rel/union.h, 554
REL_EQ
debug.h, 133
rel_eq_assoc.c
AK_compare, 450
AK_print_rel_eq_assoc, 450
AK_rel_eq_assoc, 451
AK_rel_eq_assoc_test, 451
rel_eq_assoc.h
AK_compare, 452
AK_print_rel_eq_assoc, 453
AK_rel_eq_assoc, 453
AK_rel_eq_assoc_test, 454
cost_eval, 452
rel_eq_comut.c
AK_print_rel_eq_comut, 454
AK_rel_eq_commute_with_theta_join, 455
AK_rel_eq_comut, 455
AK_rel_eq_comut_test, 456
rel_eq_comut.h
AK_print_rel_eq_comut, 457
AK_rel_eq_commute_with_theta_join, 457
AK_rel_eq_comut, 458
AK_rel_eq_comut_test, 458
rel_eq_projection.c
AK_print_rel_eq_projection, 459
AK_rel_eq_can_commute, 460
AK_rel_eq_collect_cond_attributes, 460
AK_rel_eq_get_attributes, 461
AK_rel_eq_is_subset, 461
AK_rel_eq_projection, 462
AK_rel_eq_projection_attributes, 463
AK_rel_eq_projection_test, 463
AK_rel_eq_remove_duplicates, 464
rel_eq_projection.h
AK_print_rel_eq_projection, 465
AK_rel_eq_can_commute, 465
AK_rel_eq_collect_cond_attributes, 466
AK_rel_eq_get_attributes, 466
AK_rel_eq_is_subset, 467
AK_rel_eq_projection, 468
AK_rel_eq_projection_attributes, 469
AK_rel_eq_projection_test, 469
AK_rel_eq_remove_duplicates, 470
rel_eq_selection.c
AK_print_rel_eq_selection, 471
AK_rel_eq_cond_attributes, 471
AK_rel_eq_get_atrributes_char, 472
AK_rel_eq_is_attr_subset, 472
AK_rel_eq_selection, 473
AK_rel_eq_selection_test, 473
AK_rel_eq_share_attributes, 474
AK_rel_eq_split_condition, 474
rel_eq_selection.h
AK_print_rel_eq_selection, 476
AK_rel_eq_cond_attributes, 476
AK_rel_eq_get_atrributes_char, 477
AK_rel_eq_is_attr_subset, 479
AK_rel_eq_selection, 480
AK_rel_eq_selection_test, 480
AK_rel_eq_share_attributes, 480
AK_rel_eq_split_condition, 481
REL_OP
debug.h, 133
RESET
test.h, 211
result
AK_query_mem, 36
result_block
AK_results, 43
result_id
AK_results, 43
result_size
AK_results, 43
results
AK_query_mem_result, 39
RO_EXCEPT
constants.h, 124
RO_INTERSECT
constants.h, 124
RO_NAT_JOIN
constants.h, 124
RO_PROJECTION
constants.h, 124
RO_RENAME
constants.h, 124
RO_SELECTION
constants.h, 125
RO_THETA_JOIN
constants.h, 125

RO_UNION
constants.h, 125
root
root_info, 65
root_info, 65
level, 65
root, 65
row_root
rowroot_struct, 66
rowroot_struct, 65
row_root, 66

SEARCH_ALL
filesearch.h, 288
SEARCH_CONSTRAINT
constants.h, 125
SEARCH_NULL
filesearch.h, 289
search_params, 66
iSearchType, 66
pData_lower, 67
pData_upper, 67
szAttribute, 67
SEARCH_PARTICULAR
filesearch.h, 289
SEARCH_RANGE
filesearch.h, 289
search_result, 67
aiBlocks, 68
aiSearch_attributes, 68
aiTuple_addresses, 68
iNum_search_attributes, 68
iNum_tuple_addresses, 69
iNum_tuple_attributes, 69
searchValue
btree.c, 321
btree.h, 328
SEGMENT_TYPE_INDEX
constants.h, 125
SEGMENT_TYPE_SYSTEM_TABLE
constants.h, 125
SEGMENT_TYPE_TABLE
constants.h, 125
SEGMENT_TYPE_TEMP
constants.h, 126
SEGMENT_TYPE_TRANSACTION
constants.h, 126
SEGMENTLENGTH
dbman.h, 240
SELECT
constants.h, 126
select.c
AK_select, 646
AK_select_test, 646
select.h
AK_select, 647
AK_select_test, 647
selection.c
AK_op_selection_test, 544
AK_op_selection_test_pattern, 544
AK_selection, 545
AK_selection_op_rename, 545
selection.h
AK_op_selection_test, 546
AK_op_selection_test_pattern, 546
AK_selection, 547
selection_test
test.c, 208
test.h, 216
SEPARATOR
constants.h, 126
sequence.c
AK_sequence_add, 361
AK_sequence_current_value, 361
AK_sequence_get_id, 362
AK_sequence_modify, 362
AK_sequence_next_value, 363
AK_sequence_remove, 363
AK_sequence_rename, 363
AK_sequence_test, 364
sequence.h
AK_sequence_add, 365
AK_sequence_current_value, 366
AK_sequence_get_id, 366
AK_sequence_modify, 366
AK_sequence_next_value, 367
AK_sequence_remove, 368
AK_sequence_rename, 368
AK_sequence_test, 369
SEQUENCES
debug.h, 133
setNodePointers
btree.c, 321
btree.h, 329
SHARED_LOCK
constants.h, 126
size
_dictionary_, 16
AK_tuple_dict, 45
list_node, 55
source_table
AK_results, 43
sql/command.c, 556
sql/command.h, 557
sql/cs/between.c, 559
sql/cs/between.h, 562
sql/cs/check_constraint.c, 566
sql/cs/check_constraint.h, 569
sql/cs/constraint_names.c, 572
sql/cs/constraint_names.h, 573
sql/cs/nnull.c, 575
sql/cs/nnull.h, 577
sql/cs/reference.c, 580
sql/cs/reference.h, 585
sql/cs/unique.c, 596
sql/cs/unique.h, 599
sql/drop.c, 601

sql/drop.h, 604
sql/function.c, 606
sql/function.h, 612
sql/insert.c, 618
sql/insert.h, 620
sql/privileges.c, 622
sql/privileges.h, 633
sql/select.c, 645
sql/select.h, 646
sql/trigger.c, 648
sql/trigger.h, 653
sql/view.c, 658
sql/view.h, 664
Stack, 69
    link, 70
    nextElement, 70
struct_add, 70
    addBlock, 70
    indexTd, 71
Succesor, 71
    link, 71
    nextSuccesor, 72
success
    blobs.c, 263
system_catalog
    drop.c, 604
szAttribute
    search_params, 67

TABLE
    table.h, 384
    tableOld.h, 410
table
    AK_ref_item, 42
    list_node, 55
table.c
    AK_check_tables_scheme, 370
    AK_create_create_table_parameter, 371
    AK_create_table, 371
    AK_get_attr_index, 372
    AK_get_attr_name, 372
    AK_get_column, 373
    AK_get_header, 373
    AK_get_num_records, 374
    AK_get_row, 374
    AK_get_table_obj_id, 375
    AK_get_tuple, 375
    AK_num_attr, 376
    AK_op_rename_test, 376
    AK_print_row, 376
    AK_print_row_spacer, 377
    AK_print_row_spacer_to_file, 377
    AK_print_row_to_file, 378
    AK_print_table, 378
    AK_print_table_to_file, 379
    AK_rename, 379
    AK_table_empty, 380
    AK_table_exist, 380
    AK_table_test, 380
    AK_temp_create_table, 381
    AK_tuple_to_string, 381
    get_row_attr_data, 382
table.h
    AK_check_tables_scheme, 384
    AK_create_create_table_parameter, 385
    AK_create_table, 385
    AK_create_table_parameter, 384
    AK_get_attr_index, 386
    AK_get_attr_name, 386
    AK_get_column, 387
    AK_get_header, 387
    AK_get_num_records, 388
    AK_get_row, 388
    AK_get_table_obj_id, 389
    AK_get_tuple, 389
    AK_num_attr, 390
    AK_op_rename_test, 390
    AK_print_row, 390
    AK_print_row_spacer, 391
    AK_print_row_spacer_to_file, 391
    AK_print_row_to_file, 392
    AK_print_table, 392
    AK_print_table_to_file, 393
    AK_rename, 393
    AK_table_empty, 394
    AK_table_test, 394
    AK_temp_create_table, 394
    AK_tuple_to_string, 395
    get_row_attr_data, 395
    TABLE, 384
table_addresses, 72
    address_from, 72
    address_to, 72
table_name
    AK_command_recovery_struct, 26
tableOld.c
    AK_check_tables_scheme, 397
    AK_create_create_table_parameter, 397
    AK_create_table, 397
    AK_get_attr_index, 398
    AK_get_attr_name, 398
    AK_get_column, 399
    AK_get_header, 399
    AK_get_num_records, 400
    AK_get_row, 400
    AK_get_table_obj_id, 401
    AK_get_tuple, 401
    AK_num_attr, 402
    AK_op_rename_test, 402
    AK_print_row, 403
    AK_print_row_spacer, 403
    AK_print_row_spacer_to_file, 404
    AK_print_row_to_file, 404
    AK_print_table, 405
    AK_print_table_to_file, 405
    AK_rename, 405
    AK_table_empty, 406

AK_table_exist, 406
AK_table_test, 407
AK_temp_create_table, 407
AK_tuple_to_string, 408
get_row_attr_data, 408
tableOld.h
AK_check_tables_scheme, 410
AK_create_create_table_parameter, 411
AK_create_table, 411
AK_create_table_parameter, 410
AK_get_attr_index, 412
AK_get_attr_name, 412
AK_get_column, 413
AK_get_header, 413
AK_get_num_records, 414
AK_get_row, 414
AK_get_table_obj_id, 415
AK_get_tuple, 415
AK_num_attr, 416
AK_op_rename_test, 416
AK_print_row, 416
AK_print_row_spacer, 417
AK_print_row_spacer_to_file, 417
AK_print_row_to_file, 418
AK_print_table, 418
AK_print_table_to_file, 419
AK_rename, 419
AK_table_empty, 420
AK_table_test, 420
AK_temp_create_table, 420
AK_tuple_to_string, 421
get_row_attr_data, 421
TABLE, 410
TABLES
debug.h, 133
tasks
AK_agg_input, 19
TBL_BOX_OFFSET
auxiliary.h, 86
tblName
AK_command_struct, 27
test.c
AK_create_test_table_assistant, 203
AK_create_test_table_course, 203
AK_create_test_table_department, 203
AK_create_test_table_employee, 204
AK_create_test_table_professor, 204
AK_create_test_table_professor2, 204
AK_create_test_table_student, 205
AK_create_test_tables, 205
AK_get_table_atribute_types, 205
create_header_test, 206
get_column_test, 206
get_row_test, 207
insert_data_test, 207
selection_test, 208
TEST_output_results, 201
TEST_result, 201
test.h
AK_create_test_tables, 213
AK_get_table_atribute_types, 214
BLACK, 209
BLUE, 209
BOLDBLACK, 209
BOLDBLUE, 210
BOLDCYAN, 210
BOLDGREEN, 210
BOLDMAGENTA, 210
BOLDRED, 210
BOLDWHITE, 210
BOLDYELLOW, 210
create_header_test, 214
CYAN, 210
get_column_test, 215
get_row_test, 215
GREEN, 211
insert_data_test, 216
MAGENTA, 211
RED, 211
RESET, 211
selection_test, 216
TEST_output_results, 212
TEST_result, 212
TestResult, 211
WHITE, 211
YELLOW, 211
test_lastCharacterWritten
dbman.c, 235
TEST_MODE_OFF
constants.h, 126
TEST_MODE_ON
constants.h, 127
TEST_output_results
test.c, 201
test.h, 212
TEST_result
test.c, 201
test.h, 212
test_threadSafeBlockAccessSucceeded
dbman.c, 235
testFailed
TestResult, 73
testMode
auxiliary.h, 104
TestResult, 73
implemented, 73
test.h, 211
testFailed, 73
testSucceeded, 73
testSucceded
TestResult, 73
theta_join.c
AK_check_constraints, 548
AK_create_theta_join_header, 548
AK_op_theta_join_test, 549
AK_theta_join, 549

theta_join.h
AK_check_constraints, 551
AK_create_theta_join_header, 551
AK_op_theta_join_test, 552
AK_theta_join, 552
thread
threadContainer, 74
thread_holding_lock
AK_block_activity, 23
threadContainer, 74
nextThread, 74
thread, 74
timestamp_last_change
AK_mem_block, 34
timestamp_read
AK_mem_block, 34
tools/comments.py, 668
tools/getFiles.sh, 669
tools/parseC.sh, 669
tools/parsePy.sh, 669
tools/updateVersion.sh, 669
trans/transaction.c, 669
trans/transaction.h, 686
transaction.c
accessLockMutex, 684
acquireLockMutex, 684
activeThreads, 684
activeTransactionsCount, 685
AK_acquire_lock, 671
AK_add_hash_entry_list, 672
AK_add_lock, 673
AK_all_transactions_finished, 673
AK_create_lock, 673
AK_create_new_transaction_thread, 674
AK_delete_hash_entry_list, 674
AK_delete_lock_entry_list, 675
AK_execute_commands, 675
AK_execute_transaction, 676
AK_get_memory_blocks, 676
AK_handle_observable_transaction_action, 676
AK_init_observable_transaction, 677
AK_init_observer_lock, 677
AK_isLock_waiting, 677
AK_lock_released, 678
AK_memory_block_hash, 678
AK_on_all_transactions_end, 679
AK_on_lock_release, 679
AK_on_observable_notify, 679
AK_on_transaction_end, 680
AK_release_locks, 680
AK_remove_transaction_thread, 680
AK_search_empty_link_for_hook, 681
AK_search_existing_link_for_hook, 681
AK_search_lock_entry_list_by_key, 682
AK_test_Transaction, 682
AK_transaction_finished, 682
AK_transaction_manager, 682
AK_transaction_register_observer, 683
AK_transaction_unregister_observer, 683
cond_lock, 685
endTransationTestLockMutex, 685
handle_transaction_notify, 684
LockTable, 685
newTransactionLockMutex, 685
observable_transaction, 685
transactionsCount, 685
transaction.h
AK_acquire_lock, 690
AK_add_hash_entry_list, 691
AK_add_lock, 692
AK_ALL_TRANSACTION_FINISHED, 690
AK_all_transactions_finished, 692
AK_create_lock, 692
AK_create_new_transaction_thread, 693
AK_delete_hash_entry_list, 693
AK_delete_lock_entry_list, 694
AK_execute_commands, 694
AK_execute_transaction, 695
AK_get_memory_blocks, 696
AK_handle_observable_transaction_action, 696
AK_init_observable_transaction, 696
AK_init_observer_lock, 697
AK_isLock_waiting, 697
AK_LOCK_RELEASED, 690
AK_lock_released, 698
AK_memory_block_hash, 698
AK_memoryAddresses, 688
AK_memoryAddresses_link, 688
AK_observable_transaction, 688
AK_observer_lock, 689
AK_on_all_transactions_end, 699
AK_on_lock_release, 699
AK_on_observable_notify, 699
AK_on_transaction_end, 700
AK_release_locks, 700
AK_remove_transaction_thread, 700
AK_search_empty_link_for_hook, 701
AK_search_existing_link_for_hook, 701
AK_search_lock_entry_list_by_key, 702
AK_test_Transaction, 702
AK_thread_Container, 689
AK_thread_elem, 689
AK_transaction_data, 689
AK_transaction_elem, 689
AK_transaction_elem_P, 689
AK_TRANSACTION_FINISHED, 690
AK_transaction_finished, 702
AK_transaction_list, 689
AK_transaction_lock_elem, 689
AK_transaction_lock_elem_P, 690
AK_transaction_manager, 702
AK_transaction_register_observer, 703
AK_transaction_unregister_observer, 703
handle_transaction_notify, 704
NoticeType, 690
transaction_list_elem, 75

address, 75
DLLLocksHead, 75
isWaiting, 76
lock_type, 76
nextBucket, 76
observer_lock, 76
prevBucket, 76
transaction_list_head, 76
DLLHead, 77
transaction_locks_list_elem, 77
isWaiting, 78
lock_type, 78
nextLock, 78
prevLock, 78
TransactionId, 78
transactionData, 78
array, 79
lengthOfArray, 79
TransactionId
transaction_locks_list_elem, 78
transactionsCount
transaction.c, 685
trigger.c
AK_trigger_add, 648
AK_trigger_edit, 649
AK_trigger_get_conditions, 650
AK_trigger_get_id, 650
AK_trigger_remove_by_name, 651
AK_trigger_remove_by_obj_id, 651
AK_trigger_rename, 652
AK_trigger_save_conditions, 652
AK_trigger_test, 653
trigger.h
AK_trigger_add, 654
AK_trigger_edit, 654
AK_trigger_get_conditions, 655
AK_trigger_get_id, 656
AK_trigger_remove_by_name, 656
AK_trigger_remove_by_obj_id, 657
AK_trigger_rename, 657
AK_trigger_save_conditions, 658
AK_trigger_test, 658
TRIGGERS
debug.h, 133
tuple_dict
AK_block, 21
type
_notifyDetails, 18
AK_block, 22
AK_create_table_struct, 27
AK_header, 33
AK_operand, 35
AK_ref_item, 42
AK_tuple_dict, 45
intersect_attr, 53
list_node, 55
TYPE_ATTRIBS
constants.h, 127
TYPE_BLOB
constants.h, 127
TYPE_BOOL
constants.h, 127
TYPE_CONDITION
constants.h, 127
TYPE_DATE
constants.h, 127
TYPE_DATETIME
constants.h, 128
TYPE_FLOAT
constants.h, 128
TYPE_INT
constants.h, 128
TYPE_INTERNAL
constants.h, 128
TYPE_NUMBER
constants.h, 128
TYPE_OPERAND
constants.h, 128
TYPE_OPERATOR
constants.h, 129
TYPE_TIME
constants.h, 129
TYPE_VARCHAR
constants.h, 129
TypeObservable, 79
AK_custom_register_observer, 80
AK_custom_unregister_observer, 80
AK_get_message, 80
AK_set_notify_info_details, 80
notifyDetails, 80
observable, 80
TypeObserver, 81
observable, 81
observer, 81

union.c
AK_op_union_test, 554
AK_union, 554
union.h
AK_op_union_test, 555
AK_union, 555
unique.c
AK_delete_constraint_unique, 597
AK_read_constraint_unique, 597
AK_set_constraint_unique, 598
AK_unique_test, 598
unique.h
AK_delete_constraint_unique, 599
AK_read_constraint_unique, 600
AK_set_constraint_unique, 600
AK_unique_test, 601
UPDATE
constants.h, 129
used
AK_debmod_state, 31

val

_dictionary_, 16
value
AK_operand, 35
bucket_elem, 47
cost_eval_t, 48
drop_arguments, 50
values
btree_node, 47
Vertex, 81
index, 82
lowLink, 82
nextSuccesor, 82
nextVertex, 82
vertexId, 82
vertexId
Vertex, 82
view.c
AK_check_view_name, 659
AK_get_relation_expression, 660
AK_get_view_object_id, 660
AK_get_view_query, 660
AK_test_get_view_data, 661
AK_view_add, 661
AK_view_change_query, 662
AK_view_remove_by_name, 662
AK_view_remove_by_object_id, 663
AK_view_rename, 663
AK_view_test, 664
view.h
AK_check_view_name, 665
AK_get_view_query, 665
AK_view_add, 665
AK_view_change_query, 666
AK_view_remove_by_name, 667
AK_view_rename, 667
AK_view_test, 668

WAIT_FOR_UNLOCK
constants.h, 129
WARMING
observable.c, 195
WHITE
test.h, 211
writing_done
AK_block_activity, 24

YELLOW
test.h, 211