

Kalashnikov DB

0.9.3

Generated by Doxygen 1.8.17



<b>1 Todo List</b>	<b>1</b>
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List . . . . .	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List . . . . .	5
<b>4 File Index</b>	<b>9</b>
4.1 File List . . . . .	9
<b>5 Namespace Documentation</b>	<b>13</b>
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
<b>6 Class Documentation</b>	<b>15</b>
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 nextSucesor . . . . .	82
6.63.2.4 nextVertex . . . . .	82
6.63.2.5 vertexId . . . . .	82
<b>7 File Documentation</b>	<b>83</b>
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_sucesor . . . . .	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 ASCIIINESZ . . . . .	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_getseckey() . . . . .	149
7.9.5.12 iniparser_getsecname() . . . . .	150
7.9.5.13 iniparser_getsecnkey() . . . . .	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_getseckey() . . . . .	159
7.10.2.12 iniparser_getsecname() . . . . .	159
7.10.2.13 iniparser_getsecnkey() . . . . .	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_attribute_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_attribute_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_attributes_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_attributes_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/nnnull.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/nnnull.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

<b>Index</b>	<b>705</b>
--------------	------------



# 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:

<a href="#">comments</a> . . . . .	13
------------------------------------	----



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">_dictionary_</a>	
Dictionary object	15
<a href="#">_file_metadata</a>	16
<a href="#">_notifyDetails</a>	17
<a href="#">AK_agg_input</a>	
Structure that contains attributes from table header, tasks for this table and counter value	18
<a href="#">AK_agg_value</a>	
Structure that contains attribute name, date and aggregation task associated	19
<a href="#">AK_block</a>	
Structure that defines a block of data inside a DB file. It contains address, type, chained_with, AK_free space, last_tuple_dict_id, header and tuple_dict and data	20
<a href="#">AK_block_activity</a>	
Structure which holds information about each block, whether it is locked for reading or writing. It is important to note such information, to enable quick and thread-safe reading from or writing to disk. Structure contains of: locked_for_reading - thread which locks particular block for reading will set this value locked_for_writing - thread which locks particular block for writing will set this value block_lock - each reading and writing operation will be done atomically and un-interruptable, using this mutex block lock reading_done - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block writing_done - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it thread_holding_lock - the only thread which can unlock locked "block_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it	22
<a href="#">AK_blocktable</a>	24
<a href="#">AK_command_recovery_struct</a>	
Recovery structure used to recover commands from binary file	25
<a href="#">AK_command_struct</a>	26
<a href="#">AK_create_table_struct</a>	27
<a href="#">AK_db_cache</a>	
Structure that defines global cache memory	28
<a href="#">AK_debmod_state</a>	
Global structure that holds all relevant information for the debug mode and related functionality	29
<a href="#">AK_header</a>	
Structure that represents header structure of blocks (describes an attribute inside an object). It contains type, attribute name, integrity, constraint name and constraint code	32

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

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



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

<a href="#">auxi/auxiliary.c</a>	83
<a href="#">auxi/auxiliary.h</a>	83
<a href="#">auxi/configuration.h</a>	105
<a href="#">auxi/constants.h</a>	108
<a href="#">auxi/debug.c</a>	130
<a href="#">auxi/debug.h</a>	131
<a href="#">auxi/dictionary.c</a>	
Implements a dictionary for string variables	134
<a href="#">auxi/dictionary.h</a>	
Implements a dictionary for string variables	138
<a href="#">auxi/iniparser.c</a>	
Parser for ini files	142
<a href="#">auxi/iniparser.h</a>	
Parser for ini files	153
<a href="#">auxi/mempro.c</a>	162
<a href="#">auxi/mempro.h</a>	177
<a href="#">auxi/observable.c</a>	194
<a href="#">auxi/observable.h</a>	198
<a href="#">auxi/test.c</a>	201
<a href="#">auxi/test.h</a>	208
<a href="#">dm/dbman.c</a>	217
<a href="#">dm/dbman.h</a>	235
<a href="#">file/blobs.c</a>	258
<a href="#">file/blobs.h</a>	263
<a href="#">file/fileio.c</a>	269
<a href="#">file/fileio.h</a>	275
<a href="#">file/files.c</a>	281
<a href="#">file/files.h</a>	283
<a href="#">file/filesearch.c</a>	285
<a href="#">file/filesearch.h</a>	288
<a href="#">file/filesort.c</a>	291
<a href="#">file/filesort.h</a>	294
<a href="#">file/id.c</a>	298
<a href="#">file/id.h</a>	299
<a href="#">file/sequence.c</a>	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/nnnull.c	575
sql/cs/nnnull.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 attribute name, date and aggregation task associated.

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

### Public Attributes

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

### 6.5.1 Detailed Description

Structure that contains attribute 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 uninterruptable, using this mutex block lock reading\_done - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block writing\_done - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it thread\_holding\_lock - the only thread which can unlock locked "block\_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it.

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

### Public Attributes

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

### 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 uninterruptable, using this mutex block\_lock\_reading\_done - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block writing\_done - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it thread\_holding\_lock - the only thread which can unlock locked "block\_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it.

#### Author

Domagoj Šitum

### 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 constraints*
- char [constr\\_name](#) [MAX\_CONSTRAINTS][MAX\_CONSTR\_NAME]  
*extra integrity constraint names*
- char [constr\\_code](#) [MAX\_CONSTRAINTS][MAX\_CONSTR\_CODE]  
*extra integrity constraint codes*

### 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 constraint 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 constraints

#### 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

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

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

### Public Attributes

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

### 6.29.1 Detailed Description

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

Author

Dino Laktašić

### 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 additional added to the enum. Each debug type can be easily 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 additional added to the enum. Each debug type can be easily excluded from output by setting corresponding enum element to 0.

#### Author

Dino Laktašić

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

- [auxi/debug.h](#)

## 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 [testSucceeded](#)
- 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 testSucceeded

```
int TestResult::testSucceeded
```

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 [Successor](#) \* [nextSuccessor](#)
- 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 nextSucesor

```
struct Sucesor* Vertex::nextSucesor
```

#### 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 [Successor](#)  
*Structure defines a [Successor](#) element. Every [Successor](#) has its [Vertex](#) pointer and pointer to next [Successor](#) in the linked list.*
- struct [Stack](#)  
*Structure defines a [Stack](#) element. Every [Stack](#) has its [Vertex](#) pointer and pointer to next [Stack](#) in the linked list.*
- struct [AK\\_synchronization\\_info](#)  
*Structure for managing the synchronization between multiple threads accessing the same resources (essentially a mutex).*

## Macros

- `#define MAX_LOOP_ITERATIONS 1000`
- `#define TBL_BOX_OFFSET 1`

## Typedefs

- `typedef struct list_node AK_list`
- `typedef struct list_node * AK_list_elem`
- `typedef struct Vertex AK_graph`
- `typedef struct 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 succesorld, 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)
- Tarjan algorithm that looks for a strongly connected component inside all subgraphs; using DFS.*

  - 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

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

#### Returns

pointer to the newly created edge

### 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**

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

**Returns**

pointer to the newly created node

**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**

<i>number</i>	number to evaluate
<i>int</i>	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**

<i>*arg_type</i>	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**

<i>*arg_type</i>	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**

<i>graph</i>	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**

<i>current</i>	current element of the list
<i>L</i>	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**

<i>L</i>	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**

<i>info</i>	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**

<i>L</i>	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**

<i>info</i>	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

<i>L</i>	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

<i>arr</i>	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

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

**Returns**

returns the posititon number of some elelemnt

**Author**

Dino Laktašić.

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

**Parameters**

<i>arr</i>	array of chars to 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**

<i>pos</i>	position of element in a row
<i>row</i>	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**

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

**Returns**

data type of the current list element

**Author**

Matija Šestak.

Function that fetches the data size of the element

**Parameters**

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

**Returns**

data size of the current list element

**Author**

Ljiljana Pintarić

Function that fetches the nth element in a row

**Parameters**

<i>pos</i>	position of element in a row
<i>row</i>	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**

<i>L</i>	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**

<i>data</i>	new data
<i>current</i>	current element of the list
<i>L</i>	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**

<i>data</i>	new data
<i>L</i>	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**

<i>data</i>	new data
<i>L</i>	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

<i>data</i>	new data
<i>current</i>	current element of the list
<i>L</i>	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

<i>L</i>	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**

<i>info</i>	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**

<i>current</i>	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

<i>current</i>	current element of the list
<i>L</i>	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**

<i>id</i>	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**

<i>current</i>	current element of the list
<i>L</i>	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**

<i>graphRoot</i>	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**

<i>stackRoot</i>	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**

<i>id</i>	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

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

##### Returns

found graph nod or null

#### 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

<i>L</i>	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**

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

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

**Author**

Frane Jakelić, updated by Blaž Rajič

**Parameters**

<i>id</i>	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**

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

**Returns**

size of provided data type in bytes if 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 of 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 of 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 length 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 successfully.*

  - #define `NOT_OK` 0
- Constant declaring that the method isn't completed successfully.*

  - #define `COMMIT` 1
- Constant declaring that the transaction is completed successfully.*

  - #define `ABORT` 0
- Constant declaring if the transaction is being aborted.*

  - #define `NEW_ID` 0
- Constant declaring if new obj\_id should be created.*

  - #define `MAX_BLOCKS_CURRENTLY_ACCESSED` 32
- Indicates 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 successfully.

#### 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 length 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 the number 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 successfully.

#### 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 successfully.

#### 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 debugging

### 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

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



### Returns

if debug message is printed return 1, else return 0

## 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 debugging

#### 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

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

### Returns

if debug message is printed return 1, else return 0

## 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

<i>d</i>	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

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

#### Returns

void

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

### 7.7.3.3 dictionary\_get()

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

Get a value from a dictionary.

#### Parameters

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

#### Returns

1 pointer to internally allocated character string.

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

### 7.7.3.4 dictionary\_hash()

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

Compute the hash key for a string.

#### Parameters

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

#### Returns

1 unsigned int on at least 32 bits.

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

#### 7.7.3.5 dictionary\_new()

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

Create a new dictionary object.

#### Parameters

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

#### Returns

1 newly allocated dictionary objet.

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

#### 7.7.3.6 dictionary\_set()

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

Set a value in a dictionary.

#### Parameters

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

#### Returns

int 0 if Ok, anything else otherwise

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

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

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

This function returns non-zero in case of failure.

### 7.7.3.7 dictionary\_unset()

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

Delete a key in a dictionary.

#### Parameters

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

#### Returns

void

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

## 7.8 aux/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**

<i>d</i>	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**

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

**Returns**

void

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

**7.8.3.3 dictionary\_get()**

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

Get a value from a dictionary.

**Parameters**

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

**Returns**

1 pointer to internally allocated character string.

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

#### 7.8.3.4 dictionary\_hash()

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

Compute the hash key for a string.

##### Parameters

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

##### Returns

1 unsigned int on at least 32 bits.

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

#### 7.8.3.5 dictionary\_new()

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

Create a new dictionary object.

##### Parameters

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

##### Returns

1 newly allocated dictionary objet.

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

#### 7.8.3.6 dictionary\_set()

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

Set a value in a dictionary.

**Parameters**

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

**Returns**

int 0 if Ok, anything else otherwise

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

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

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

This function returns non-zero in case of failure.

**7.8.3.7 dictionary\_unset()**

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

Delete a key in a dictionary.

**Parameters**

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

**Returns**

void

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

**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 [ASCII\\_LINESZ](#) (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

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

## Returns

void

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

### 7.9.5.3 iniparser\_dump()

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

Dump a dictionary to an opened file pointer.

**Parameters**

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

**Returns**

void

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

**7.9.5.4 iniparser\_dump\_ini()**

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

Save a dictionary to a loadable ini file.

**Parameters**

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

**Returns**

void

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

**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**

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



**Returns**

void

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

**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**

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

**Returns**

integer 1 if entry exists, 0 otherwise

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

**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**

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

**Returns**

integer

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

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

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

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

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

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

#### 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

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

##### Returns

double

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

#### 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**

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

**Returns**

integer

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

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

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

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

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

**7.9.5.10 iniparser\_getnsec()**

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

Get number of sections in a dictionary.

**Parameters**

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

**Returns**

int Number of sections found in dictionary

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

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

This function returns -1 in case of error.

**7.9.5.11 iniparser\_getseckey()**

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

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

**Parameters**

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

**Returns**

pointer to statically allocated character strings

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

This function returns NULL in case of error.

**7.9.5.12 iniparser\_getsecname()**

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

Get name for section n in a dictionary.

**Parameters**

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

**Returns**

Pointer to char string

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

This function returns NULL in case of error.

**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**

<i>d</i>	Dictionary to examine
<i>s</i>	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**

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

**Returns**

pointer to statically allocated character string

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

**7.9.5.15 iniparser\_load()**

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

Parse an ini file and return an allocated dictionary object.

**Parameters**

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

**Returns**

Pointer to newly allocated dictionary

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

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

#### 7.9.5.16 iniparser\_set()

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

Set an entry in a dictionary.

##### Parameters

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

##### Returns

int 0 if Ok, -1 otherwise.

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

#### 7.9.5.17 iniparser\_unset()

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

Delete an entry in a dictionary.

##### Parameters

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

##### Returns

void

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

### 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

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

#### Returns

void

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

#### 7.10.2.3 `iniparser_dump()`

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

Dump a dictionary to an opened file pointer.



**Parameters**

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

**Returns**

void

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

**7.10.2.4 iniparser\_dump\_ini()**

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

Save a dictionary to a loadable ini file.

**Parameters**

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

**Returns**

void

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

**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**

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

**Returns**

void

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

**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**

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

**Returns**

integer 1 if entry exists, 0 otherwise

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

**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**

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

**Returns**

integer

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

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

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

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

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

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

#### 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

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

##### Returns

double

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

#### 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**

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

**Returns**

integer

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

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

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

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

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

**Parameters**

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

**Returns**

integer

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

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

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

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

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

**7.10.2.10 iniparser\_getnsec()**

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

Get number of sections in a dictionary.

### Parameters

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

### Returns

int Number of sections found in dictionary

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

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

This function returns -1 in case of error.

#### 7.10.2.11 iniparser\_getseckey()

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

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

### Parameters

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

### Returns

pointer to statically allocated character strings

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

This function returns NULL in case of error.

#### 7.10.2.12 iniparser\_getsecname()

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

Get name for section n in a dictionary.

### Parameters

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

**Returns**

Pointer to char string

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

This function returns `NULL` in case of error.

**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**

<i>d</i>	Dictionary to examine
<i>s</i>	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**

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

**Returns**

pointer to statically allocated character string

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

### 7.10.2.15 iniparser\_load()

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

Parse an ini file and return an allocated dictionary object.

#### Parameters

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

#### Returns

Pointer to newly allocated dictionary

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

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

### 7.10.2.16 iniparser\_set()

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

Set an entry in a dictionary.

#### Parameters

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

#### Returns

int 0 if Ok, -1 otherwise.

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

### 7.10.2.17 iniparser\_unset()

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

Delete an entry in a dictionary.

## Parameters

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

## Returns

void

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

### 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\_malloc()

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

Allocates memory (see calloc) [public function].

#### Author

Marin Rukavina, Mislav Bozicevic

#### Parameters

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

#### Returns

allocated memory or NULL

### 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\_malloc()

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

Allocates memory [private function].

#### Author

Marin Rukavina, Mislav Bozicevic

**Parameters**

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

**Returns**

pointer to allocated memory or NULL

**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**

<i>ds</i>	debug mode state
<i>message</i>	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**

<i>ds</i>	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**

<i>ds</i>	debug mode state
<i>format</i>	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**

<i>ds</i>	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

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

#### 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

<i>ds</i>	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**

<i>ds</i>	debug mode state
<i>func</i> ↔ <i>_id</i>	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**

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

**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

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

## 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

<i>ds</i>	debug mode state
<i>func_name</i>	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**

<i>ds</i>	debug mode state
<i>new_function</i> $\leftrightarrow$ <i>_id</i>	

**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**

<i>func_name</i>	function name as in source
<i>source_file</i>	file name where function is defined
<i>source_line</i>	line from which this function is called

**Returns**

void

**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

<i>func_name</i>	function name as in source
<i>source_file</i>	file name where function is defined
<i>source_line</i>	line from which this function is called

## Returns

void

**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

<i>ds</i>	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

<i>func_id</i>	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

<i>func_name</i>	function name
<i>in_recur</i>	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

<i>ptr</i>	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**

<i>size</i>	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

<i>func_name</i>	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**

<i>ptr</i>	old memory
<i>size</i>	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**

<i>memory</i>	
---------------	--

**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**

<i>memory</i>	
---------------	--

## 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

<i>num</i>	number of elements
<i>size</i>	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**

<i>ds</i>	debug mode state
<i>size</i>	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**

<i>ds</i>	debug mode state
<i>message</i>	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**

<i>ds</i>	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**

<i>ds</i>	debug mode state
<i>format</i>	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

<i>ds</i>	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

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

#### 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**

<i>ds</i>	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**

<i>ds</i>	debug mode state
<i>func</i> <sub>↔</sub> <i>_id</i>	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**

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

**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**

<i>ds</i>	debug mode state
<i>function↔ _id</i>	

**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

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

<i>ds</i>	debug mode state
<i>new_function_id</i>	

## 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**

<i>func_name</i>	function name as in source
<i>source_file</i>	file name where function is defined
<i>source_line</i>	line from which this function is called

**Returns**

void

**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**

<i>func_name</i>	function name as in source
<i>source_file</i>	file name where function is defined
<i>source_line</i>	line from which this function is called

**Returns**

void

**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

<i>ds</i>	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

<i>func</i> <sub>↔</sub> <i>_id</i>	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**

<i>func_name</i>	function name
<i>in_recur</i>	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**

<i>ptr</i>	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**

<i>size</i>	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**

<i>func_name</i>	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**

<i>ptr</i>	old memory
<i>size</i>	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

<i>memory</i>	
---------------	--

#### 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

<i>memory</i>	
---------------	--

#### 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_ObservableTypeEnum) 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_ObservableTypeEnum 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\\_observable\\_type\\_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

<i>successfulAmount</i>	amount of successful tests
<i>failedAmount</i>	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\\_attribute\\_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\_attribute\_types()

```
char* AK_get_table_attribute_types (
    char * tblName )
```

returns a string containing attribute types for the supplied table name, separated by ATTR\_DELIMITER

##### Author

Goran Štrok

**Parameters**

<i>tblName</i>	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**

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

**Returns**

1 if ok, 0 otherwise

**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

<i>num</i>	- 0 based index of column
<i>tbl</i>	- 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

<i>num</i>	- 0 based index of row
<i>tbl</i>	- 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

<i>tbl_name</i>	- name of the table for which the header will be created
<i>attr_name</i>	- array of attribute names
<i>attr_value</i>	- values of attributes to be inserted
<i>_num</i>	- number of attributes
<i>_type</i>	- array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.)

**Returns**

EXIT\_SUCCESS if ok, EXIT\_ERROR otherwise

**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**

<i>src_table</i>	- name of the source table •
<i>dest_table</i>	- table in which selection will be stored
<i>sel_query</i>	- array of operators, operands and attributes (postfix query)
<i>_num</i>	- number of attributes
<i>_type</i>	- array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.)

**Returns**

EXIT\_SUCCESS if ok, EXIT\_ERROR otherwise

**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

<i>successfulAmount</i>	amount of successful tests
<i>failedAmount</i>	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\\_attribute\\_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\_attribute\_types()**

```
char* AK_get_table_attribute_types (
    char * tblName )
```

returns a string containing attribute types for the supplied table name, seperated by ATTR\_DELIMITER

**Author**

Goran Štrok

**Parameters**

<i>tblName</i>	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

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

## Returns

1 if ok, 0 otherwise

**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

<i>num</i>	- 0 based index of column
<i>tbl</i>	- 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**

<i>num</i>	- 0 based index of row
<i>tbl</i>	- 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**

<i>tbl_name</i>	- name of the table for which the header will be created
<i>attr_name</i>	- array of attribute names
<i>attr_value</i>	- values of attributes to be inserted
<i>_num</i>	- number of attributes
<i>_type</i>	- array of attribute types (eg. TYPE_INT, TYPE_VARCHAR, etc.)

**Returns**

EXIT\_SUCCESS if ok, EXIT\_ERROR otherwise

**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

<code>src_table</code>	- name of the source table •
<code>dest_table</code>	- table in which selection will be stored
<code>sel_query</code>	- array of operators, operands and attributes (postfix query)
<code>_num</code>	- number of attributes
<code>_type</code>	- 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\_EXTENT\_SIZE blocks can be allocated. If extent is successfully allocated, number of allocated extents is incremented and function goes to next block after allocated extent. Otherwise, function moves to INITIAL\_EXTENT\_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK\_free.*
  - [AK\\_header](#) \* [AK\\_create\\_header](#) (char \*attribute\_name, int type, int integrity, char \*constr\_name, char \*contr\_code)  
*Function that creates header and initialize integrity, constraint name and constraint code with parameter values of function.*
  - void [AK\\_insert\\_entry](#) ([AK\\_block](#) \*block\_address, int type, void \*entry\_data, int i)  
*Function 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 contained in this variables. If they don't match, then the error occurred! It is assumed that the same block is being written to and read from (just like [AK\\_thread\\_safe\\_block\\_access\\_test](#) function works!)*
- int [test\\_threadSafeBlockAccessSucceeded](#) = 1  
*Used in combination with [test\\_lastCharacterWritten](#). Will give the answer to question: "Has [AK\\_thread\\_safe\\_block\\_access\\_test](#) succeeded?" 0 means NO, 1 means YES.*

## 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

<i>verbosity</i>	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

<i>verbosity</i>	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

<i>header</i>	Pointer to header which will be copied into each block in blockSet
<i>blockSet</i>	Pointer to array of block addresses into which to copy header
<i>blockSetSize</i>	Number of blocks in blockSet

#### Returns

number of performed header copy

### 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 initialize integrity, constraint name and constraint code with parameter values of function.

#### Author

Matija Novak

#### Parameters

<i>name</i>	name of the attribute
<i>type</i>	type of the attribute
<i>integrity</i>	standard integrity constraint
<i>constr_name</i>	extra integrity constraint name
<i>contr_code</i>	extra integrity constraint code

**Returns**[AK\\_header](#)**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**

<i>address</i>	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**

<i>begin</i>	address of extent's first block
<i>end</i>	address of extent's last block

**Returns**

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

### 7.19.2.13 AK\_delete\_segment()

```
int AK_delete_segment (
    char * name,
    int type )
```

#### Author

Mislav Ėakariæ, fixed by Josip Susnjara

#### Parameters

<i>name</i>	name of the segment
<i>type</i>	type of the segment

#### Returns

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

### 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

<i>allocationSet</i>	Pointer to array which will be filled and represent the allocation set
<i>fromWhere</i>	Has meaning only if mode is SEQUENCE. It describes from which address searching starts.
<i>gaplength</i>	Tells how many used blocks can be tolerated in allocation set
<i>numRequestedBlocks</i>	Tells how many AK_free blocks have been requested
<i>mode</i>	Defines how to obtain set of indexes to AK_free addresses
<i>target</i>	Has meaning just if mode is AROUND: set will be as close as possible to the requested target address from both sides

**Returns**

the first element of the allocation set

**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**

<i>start_address</i>	address (block number) to start searching for sufficient space
<i>desired_size</i>	number of desired blocks
<i>AK_allocation_set_mode</i>	a way of trying to find AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND
<i>border</i>	number of allocated blocks gap
<i>target</i>	block address around which other blocks have to be searched
<i>header</i>	pointer to header that should be written to the new extent (all blocks)
<i>int</i>	gl gap size

**Returns**

pointer to set of allocated block addresses

vars for loop [for]

if some blocks are not successfully allocated, which means that the extend allocation has FAILED

**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 new blocks for increasing extent size.

**Author**

dv

**Parameters**

<i>start_address</i>	first address of extent that is subject of increasing
<i>add_size</i>	number how many new blocks is to be added to existing extent
<i>AK_allocation_set_mode</i>	a way of trying to find AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND
<i>border</i>	number of allocated blocks gap
<i>target</i>	block address around which other blocks have to be searched
<i>header</i>	pointer to header that should be written to the new extent (all blocks)
<i>int</i>	gl gap size

**Returns**

pointer to set of allocated block addresses

**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

<i>relation</i>	address of system table of relation in db_file
<i>attribute</i>	address of system table of attribute in db_file
<i>index</i>	address of system table of index in db_file
<i>view</i>	address of system table of view in db_file
<i>sequence</i>	address of system table of sequence in db_file
<i>function</i>	address of system table of function in db_file
<i>function_arguments</i>	address of system table of function_arguments in db_file
<i>trigger</i>	address of system table of trigger in db_file
<i>trigger_conditions</i>	address of system table of trigger_conditions in db_file
<i>db</i>	address of system table of db in db_file
<i>db_obj</i>	address of system table of db_obj in db_file
<i>user</i>	address of system table of user in db_file
<i>group</i>	address of system table of group in db_file
<i>user_group</i>	address of system table of users associated with groups in db_file
<i>user_right</i>	address of system table of user right in db_file
<i>group_right</i>	address of system table of group right in db_file
<i>constraint</i>	address of system table of constraint in db_file
<i>constraintNull</i>	address of system table of constraintNull in db_file
<i>constraintCheck</i>	system table address for check constraint
<i>reference</i>	address of system table of reference in db_file

**Returns**

EXIT\_SUCCESS if initialization was succesful if not returns EXIT\_ERROR

first header attribute of catalog\_block

second attribute of catalog\_block

initialize other elements of block (adress, type, chained\_with, AK\_free\_space)

using as an address for the first AK\_free space in block->data

merge catalog\_heder with heders created before

**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**

<i>block_adress</i>	adress of a block in which we want insert data
<i>type</i>	type of entry_data
<i>entry_data</i>	(char) data which is inserted, can be int but must first be converted to char
<i>i</i>	(int) adress in tuple_dict array (example block_address->tuple_dict[i])

**Returns**

No return value because it gets the address of an block like a function parameter and works directly with the original block

copy data into bloc->data on start position bloc->AK\_free\_space

address of entry data in block->data

calculate next AK\_free space for the next entry data

sizeof(entry\_data)+1);/(sizeof(int)); no need for "+strlen(entry\_data)" while "+1" is like "new line"

type of entry data

size of entry data

copy tuple\_dict to block->tuple\_dict[i] must use & becouse tuple\_dict[i] is value and catalog\_tuple\_dict adress

### 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

<i>block</i>	pointer to the block of memory to fill
<i>value</i>	int value to be set
<i>num</i>	number of ints in the block of memory to be set

#### Returns

No return value

### 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 Laktašić (fixed header BUG), refurbished by dv

#### Parameters

<i>start_address</i>	address (block number) to start searching for sufficient space
<i>old_size</i>	size of previous extent in same segment (in blocks)
<i>extent_type</i>	type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP)
<i>header</i>	pointer to header that should be written to the new extent (all blocks)

**Returns**

address (block number) of new extent if successful, EXIT\_ERROR otherwise

**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**

<i>name</i>	(character pointer) name of segment
<i>type</i>	segment type (possible values: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP)
<i>header</i>	(header pointer) pointer to header that should be written to the new extent (all blocks)

**Returns**

EXIT\_SUCCESS for success or EXIT\_ERROR if some error occurs

start address for segment because we can not allocate segment in block 0

**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

<i>address</i>	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

<i>relation</i>	relation in database
<i>attribute</i>	attribute in database
<i>index</i>	index in database
<i>view</i>	view in database
<i>sequence</i>	sequence in database
<i>function</i>	function in database
<i>function_arguments</i>	functional_arguments in database
<i>trigger</i>	trigger in database
<i>trigger_conditions</i>	trigger conditions in database
<i>db</i>	database
<i>db_obj</i>	database object
<i>user</i>	user in database
<i>group</i>	group in database
<i>user_group</i>	user associated with group in database
<i>user_right</i>	user right in database
<i>group_right</i>	group right in database
<i>constraint</i>	constraint in database
<i>constraintNull</i>	Null constraint in database
<i>constraintCheck</i>	Check constraint in database
<i>reference</i>	reference database

#### Returns

EXIT\_SUCCESS

#### 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**

<i>block</i>	pointer 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 contained in this variables. If they don't match, then the error occurred! It is assumed that the same block is being written to and read from (just like AK\_thread\_safe\_block\_access\_test function works!)

#### 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 succeeded?" 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 uninterruptable, using this mutex block lock reading\_done - represents signal, which sends thread that just finished reading block. This signal will indicate that writing thread can start writing to block writing\_done - represents signal, which sends thread that just finished writing to block. This signal will indicate that other threads can start reading from this block or even writing to it thread\_holding\_lock - the only thread which can unlock locked "block\_lock" is the one that locked it. This variable makes sure that ONLY the thread, which actually holds the lock, releases it.

## Macros

- #define [BITMASK\(b\)](#) (1 << ((b) % CHAR\_BIT))
- #define [BITSLOT\(b\)](#) ((int)((b) / CHAR\_BIT))
- #define [BITSET\(a, b\)](#) ((a)[[BITSLOT\(b\)](#)] |= [BITMASK\(b\)](#))
- #define [BITCLEAR\(a, b\)](#) ((a)[[BITSLOT\(b\)](#)] &= ~[BITMASK\(b\)](#))
- #define [BITTEST\(a, b\)](#) ((a)[[BITSLOT\(b\)](#)] & [BITMASK\(b\)](#))
- #define [BITNSLOTS\(nb\)](#) ((int)(nb + CHAR\_BIT - 1) / CHAR\_BIT)
- #define [SEGMENTLENGTH\(\)](#) ([BITNSLOTS\(DB\\_FILE\\_BLOCKS\\_NUM\)](#) + 2\*sizeof(int))
- #define [DB\\_FILE\\_SIZE\\_EX](#) 200
- #define [DB\\_FILE\\_BLOCKS\\_NUM\\_EX](#) (int)(1024 \* 1024 \* [DB\\_FILE\\_SIZE\\_EX](#) / sizeof([AK\\_block](#)))
- #define [AK\\_ALLOCATION\\_TABLE\\_SIZE](#) sizeof([AK\\_blocktable](#))  
Holds size of allocation table.
- #define [CHAR\\_IN\\_LINE](#) 80  
How many characters could line contain.
- #define [MAX\\_BLOCK\\_INIT\\_NUM](#) [MAX\\_CACHE\\_MEMORY](#)  
How many blocks would be initially allocated.

## Enumerations

- enum [AK\\_allocation\\_set\\_mode](#) {  
    [allocationSEQUENCE](#) = 10001, [allocationUPPER](#), [allocationLOWER](#), [allocationAROUND](#),  
    [allocationNOMODE](#) }  
Different modes to obtain allocation indexes: SEQUENCE - first found set of sequence indexes UPPER - set tries to place itself to upper part of allocation table LOWER - set tries to place itself to lower part of allocation table AROUND - set tries to place itself around targeted index.

## Functions

- `int AK_print_block (AK_block *block, int num, char *gg, FILE *fpp)`  
*Function that dumps 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\_EXTENT\_SIZE blocks can be allocated. If extent is successfully allocated, number of allocated extents is incremented and function goes to next block after allocated extent. Otherwise, function moves to INITIAL\_EXTENT\_SIZE blocks. In that way function gets either first block of new extent or some block in that extent which will not be AK\_free.

- [AK\\_header](#) \* [AK\\_create\\_header](#) (char \*name, int type, int integrity, char \*constr\_name, char \*constr\_code)

Function that creates header and initialize integrity, constraint name and constraint code with parameter values of function.

- void [AK\\_insert\\_entry](#) ([AK\\_block](#) \*block\_address, int type, void \*entry\_data, int i)

Function 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 of allocation table LOWER - set tries to place itself to lower part of allocation table AROUND - set tries to place itself around targeted index.

#### Author

dv

#### 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

<i>verbosity</i>	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

<i>verbosity</i>	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

<i>header</i>	Pointer to header which will be copied into each block in blockSet
<i>blockSet</i>	Pointer to array of block addresses into which to copy header
<i>blockSetSize</i>	Number of blocks in blockSet

**Returns**

number of performed header copy

**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 initialize integrity, constraint name and constraint code with parameter values of function.

**Author**

Matija Novak

**Parameters**

<i>name</i>	name of the attribute
<i>type</i>	type of the attribute
<i>integrity</i>	standard integrity constraint
<i>constr_name</i>	extra integrity constraint name
<i>contr_code</i>	extra integrity constraint code

**Returns**

[AK\\_header](#)

**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

<i>address</i>	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

<i>begin</i>	address of extent's first block
<i>end</i>	address of extent's last block

## Returns

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

**7.20.4.12 AK\_delete\_segment()**

```
int AK_delete_segment (
    char * name,
    int type )
```

## Author

Mislav Ćakarjæ, fixed by Josip Susnjara

## Parameters

<i>name</i>	name of the segment
<i>type</i>	type of the segment

**Returns**

EXIT\_SUCCESS if extent has been successfully deleted, EXIT\_ERROR otherwise

**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**

<i>allocationSet</i>	Pointer to array which will be filled and represent the allocation set
<i>fromWhere</i>	Has meaning only if mode is SEQUENCE. It describes from which address searching starts.
<i>gaplength</i>	Tells how many used blocks can be tolerated in allocation set
<i>numRequestedBlocks</i>	Tells how many AK_free blocks have been requested
<i>mode</i>	Defines how to obtain set of indexes to AK_free addresses
<i>target</i>	Has meaning just if mode is AROUND: set will be as close as possible to the requested target address from both sides

**Returns**

the first element of the allocation set

**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

<i>start_address</i>	address (block number) to start searching for sufficient space
<i>desired_size</i>	number of desired blocks
<i>AK_allocation_set_mode</i>	a way of trying to find AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND
<i>border</i>	number of allocated blocks gap
<i>target</i>	block address around which other blocks have to be searched
<i>header</i>	pointer to header that should be written to the new extent (all blocks)
<i>int</i>	gl gap size

## Returns

pointer to set of allocated block addresses

vars for loop [for]

if some blocks are not successfully allocated, which means that the extend allocation has FAILED

## 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

<i>start_address</i>	first address of extent that is subject of increasing
<i>add_size</i>	number how many new blocks is to be added to existing extent
<i>AK_allocation_set_mode</i>	a way of trying to find AK_free space. Can be one of: allocationSEQUENCE, allocationUPPER, allocationLOWER, allocationAROUND
<i>border</i>	number of allocated blocks gap
<i>target</i>	block address around which other blocks have to be searched
<i>header</i>	pointer to header that should be written to the new extent (all blocks)
<i>int</i>	gl gap size

**Returns**

pointer to set of allocated block addresses

**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**

<i>size</i>	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

<i>relation</i>	address of system table of relation in db_file
<i>attribute</i>	address of system table of attribute in db_file
<i>index</i>	address of system table of index in db_file
<i>view</i>	address of system table of view in db_file
<i>sequence</i>	address of system table of sequence in db_file
<i>function</i>	address of system table of function in db_file
<i>function_arguments</i>	address of system table of function_arguments in db_file
<i>trigger</i>	address of system table of trigger in db_file
<i>trigger_conditions</i>	address of system table of trigger_conditions in db_file
<i>db</i>	address of system table of db in db_file
<i>db_obj</i>	address of system table of db_obj in db_file
<i>user</i>	address of system table of user in db_file
<i>group</i>	address of system table of group in db_file
<i>user_group</i>	address of system table of users associated with groups in db_file
<i>user_right</i>	address of system table of user right in db_file
<i>group_right</i>	address of system table of group right in db_file
<i>constraint</i>	address of system table of constraint in db_file
<i>constraintNull</i>	address of system table of constraintNull in db_file
<i>constraintCheck</i>	system table address for check constraint
<i>reference</i>	address of system table of reference in db_file



**Returns**

EXIT\_SUCCESS if initialization was succesful if not returns EXIT\_ERROR

first header attribute of catalog\_block

second attribute of catalog\_block

initialize other elements of block (adress, type, chained\_with, AK\_free\_space)

using as an address for the first AK\_free space in block->data

merge catalog\_heder with heders created before

**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**

<i>block_adress</i>	adress of a block in which we want insert data
<i>type</i>	type of entry_data
<i>entry_data</i>	(char) data which is inserted, can be int but must first be converted to char
<i>i</i>	(int) adress in tuple_dict array (example block_address->tuple_dict[i])

**Returns**

No return value because it gets the address of an block like a function parameter and works directly with the original block

copy data into bloc->data on start position bloc->AK\_free\_space

address of entry data in block->data

calculate next AK\_free space for the next entry data

sizeof(entry\_data)+1);/(sizeof(int)); no need for "+strlen(entry\_data)" while "+1" is like "new line"

type of entry data

size of entry data

copy tuple\_dict to block->tuple\_dict[i] must use & becouse tuple\_dict[i] is value and catalog\_tuple\_dict adress

#### 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

<i>block</i>	pointer to the block of memory to fill
<i>value</i>	int value to be set
<i>num</i>	number of ints in the block of memory to be set

##### Returns

No return value

#### 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 Laktašić (fixed header BUG), refurbished by dv

##### Parameters

<i>start_address</i>	address (block number) to start searching for sufficient space
<i>old_size</i>	size of previous extent in same segment (in blocks)
<i>extent_type</i>	type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP)
<i>header</i>	pointer to header that should be written to the new extent (all blocks)

**Returns**

address (block number) of new extent if successful, EXIT\_ERROR otherwise

**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**

<i>name</i>	(character pointer) name of segment
<i>type</i>	segment type (possible values: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP)
<i>header</i>	(header pointer) pointer to header that should be written to the new extent (all blocks)

**Returns**

EXIT\_SUCCESS for success or EXIT\_ERROR if some error occurs

start address for segment because we can not allocate segment in block 0

**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

<i>address</i>	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

<i>relation</i>	relation in database
<i>attribute</i>	attribute in database
<i>index</i>	index in database
<i>view</i>	view in database
<i>sequence</i>	sequence in database
<i>function</i>	function in database
<i>function_arguments</i>	functional_arguments in database
<i>trigger</i>	trigger in database
<i>trigger_conditions</i>	trigger conditions in database
<i>db</i>	database
<i>db_obj</i>	database object
<i>user</i>	user in database
<i>group</i>	group in database
<i>user_group</i>	user associated with group in database
<i>user_right</i>	user right in database
<i>group_right</i>	group right in database
<i>constraint</i>	constraint in database
<i>constraintNull</i>	Null constraint in database
<i>constraintCheck</i>	Check constraint in database
<i>reference</i>	reference database

#### Returns

EXIT\_SUCCESS

#### 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**

<i>block</i>	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 globally 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 globally 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 elements 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 whether block contains 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 violate reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_DIRTY.*
- 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 are 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

<i>row_root</i>	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

<i>id</i>	id of row
<i>tableName</i>	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**

<i>temp_block</i>	block to work with
<i>row_list</i>	list of elements which contain data for delete or update

**Returns**

No return value

**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**

<i>row_root</i>	elements of one row
<i>del</i>	- 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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	is NEW_VALUE

#### Returns

No return value

### 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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for

## Returns

No return value

## 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 violate reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_DIRTY.

## Author

Matija Novak, updated by Matija Šestak (function now uses caching), updated by Dejan Frankovic (added reference check), updated by Dino Laktašić (removed variable AK\_free, variable table initialized using memset)

## Parameters

<i>row_root</i>	list of elements which contain data of one row
-----------------	--

## Returns

EXIT\_SUCCESS if success else EXIT\_ERROR

## 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 whether block contains attributes from the list. Then data, type, size and last\_tuple\_id are put in temp\_block.

## Author

Matija Novak, updated by Dino Laktašić

**Parameters**

<i>row_root</i>	list of elements to insert
<i>temp_block</i>	block in which we insert data

**Returns**

EXIT 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**

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	is NEW_VALUE

**Returns**

No return value

**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**

<i>row_root</i>	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**

<i>temp_block</i>	block to work with
<i>row_list</i>	list of elements which contain data for delete or update

**Returns**

No return value

**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 elements 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 whether block contains 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 violate reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_DIRTY.*
- 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 are 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

<i>row_root</i>	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

<i>id</i>	id of row
<i>tableName</i>	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

<i>temp_block</i>	block to work with
<i>row_list</i>	list of elements which contain data for delete or update

## Returns

No return value

#### 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

<i>row_root</i>	elements of one row
<i>del</i>	- 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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	is NEW_VALUE

**Returns**

No return value

**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 elements are set according to function arguments. Pointers are changed so that before element points to new element.

**Author**

Matija Novak

**Parameters**

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for

**Returns**

No return value

**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 violate reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_↵DIRTY.

**Author**

Matija Novak, updated by Matija Šestak (function now uses caching), updated by Dejan Frankovic (added reference check), updated by Dino Laktašić (removed variable AK\_free, variable table initialized using memset)

**Parameters**

<i>row_root</i>	list of elements which contain data of one row
-----------------	--

**Returns**

EXIT\_SUCCESS if success else EXIT\_ERROR

**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**

<i>row_root</i>	list of elements to insert
<i>temp_block</i>	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**

<i>row_root</i>	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**

<i>temp_block</i>	block to work with
<i>row_list</i>	list of elements which contain data for delete or update

**Returns**

No return value

**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

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	pointer to header that should be written to the new extent (all blocks)



**Returns**

start address of new segment

**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**

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	pointer to header that should be written to the new extent (all blocks)

**Returns**

start address of new segment

**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

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	pointer to header that should be written to the new extent (all blocks)

## Returns

start address of new segment

### 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

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	pointer to header that should be written to the new extent (all blocks)

## Returns

start address of new segment

## 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

<i>srResult</i>	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**

<i>szRelation</i>	relation name
<i>aspParams</i>	array of search parameters
<i>iNum_search_params</i>	number of search parameters

**Returns**

[search\\_result](#) structure defined in [filesearch.h](#). Use [AK\\_deallocate\\_search\\_result](#) to deallocate.

iterate through all the blocks

count number of attributes in segment/relation

determine index of attributes on which search will be performed

if any of the provided attributes are not found in the relation, return empty result

in every tuple, for all required attributes, compare attribute value with searched-for value and store matched tuple addresses

## 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

<i>srResult</i>	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

<i>szRelation</i>	relation name
<i>aspParams</i>	array of search parameters
<i>iNum_search_params</i>	number of search parameters

#### Returns

[search\\_result](#) structure defined in [filesearch.h](#). Use [AK\\_deallocate\\_search\\_result](#) to deallocate.

iterate through all the blocks

count number of attributes in segment/relation

determine index of attributes on which search will be performed

if any of the provided attributes are not found in the relation, return empty result

in every tuple, for all required attributes, compare attribute value with searched-for value and store matched tuple addresses



## 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

<i>iBlock</i>	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

<i>block</i>	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
```

Constant 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

<i>iBlock</i>	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

<i>block</i>	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 return 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 return 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 num↔Attributes, [AK\\_header](#) \*headerIndex)  
*Function that loads index table with the value of particular attribute.*
- [list\\_ad](#) \* [AK\\_get\\_attribute](#) (char \*indexName, char \*attribute)  
*Function that gets addresses of the particular attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. That data is put in a list called add\_root.*
- void [AK\\_print\\_Att\\_Test](#) ([list\\_ad](#) \*list)  
*Function that prints the list of addresses.*
- [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 already exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.*
- 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

<i>block</i>	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

<i>tableName</i>	name of table
<i>attributeName</i>	name of attribute
<i>newAttributeValue</i>	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 particular attribute.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

<i>tblName</i>	source table
<i>tblNameIndex</i>	new name of index table
<i>attributeName</i>	attribute on which we make index
<i>positionTbl</i>	position of attribute in header of table
<i>numAtributes</i>	number of attributes in table
<i>headerIndex</i>	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 headerAtributes. According to those elements new indexes are created.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

<i>tblName</i>	name of table
<i>attributes</i>	list of attributes on which we will create indexes

**Returns**

No return value

**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**

<i>Bitmap</i>	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 addresses of the particular attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. That data is put in a list called `add_root`.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

<i>indexName</i>	name of index
<i>attribute</i>	name of attribute

**Returns**

list of adresses

**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**

<i>tableName</i>	name of table
<i>attributeValue</i>	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**

<i>L</i>	list of elements
<i>ele</i>	operator to be found in list

**Returns**

1 if operator ele is found in list, otherwise 0

**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**

<i>list</i>	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**

<i>tblName</i>	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 already exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.

#### Author

Saša Vukšić

#### Parameters

<i>addBlock</i>	adress of block
<i>addTD</i>	adress of tuple dict
<i>tableName</i>	name of table
<i>attributeName</i>	name of attribute
<i>attributeValue</i>	value of attribute
<i>newAttributeValue</i>	new value of updated attribute

#### Returns

No return value

## 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 numAttributes, [AK\\_header](#) \*headerIndex)  
*Function that loads index table with the value of particular attribute.*
- [list\\_ad](#) \* [AK\\_get\\_attribute](#) (char \*indexName, char \*attribute)  
*Function that gets addresses of the particular attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. 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 addresses.*
- [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 already exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.*
- int [AK\\_write\\_block](#) ([AK\\_block](#) \*block)  
*Function 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**

<i>tableName</i>	name of table
<i>attributeName</i>	name of attribute
<i>newAttributeValue</i>	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**

<i>block</i>	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**

<i>tableName</i>	name of table
<i>attributeName</i>	name of attribute
<i>newAttributeValue</i>	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 particular attribute.

**Author**

Saša Vukšić, Lovro Predovan

**Parameters**

<i>tblName</i>	source table
<i>tblNameIndex</i>	new name of index table
<i>attributeName</i>	attribute on which we make index
<i>positionTbl</i>	position of attribute in header of table
<i>numAtributes</i>	number of attributes in table
<i>headerIndex</i>	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 headerAtributes. According to those elements new indexes are created.

**Author**

Saša Vukšić, Lovro Predovan

## Parameters

<i>tblName</i>	name of table
<i>attributes</i>	list of attributes on which we will create indexes

## Returns

No return value

**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

<i>Bitmap</i>	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 addresses of the particular attribute from bitmap index. It fetches addresses of indexes and header of index table. Using while loop it goes through index and gets necessary data. That data is put in a list called `add_root`.

## Author

Saša Vukšić, Lovro Predovan

**Parameters**

<i>indexName</i>	name of index
<i>attribute</i>	name of attribute

**Returns**

list of adresses

**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**

<i>tableName</i>	name of table
<i>attributeValue</i>	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

<i>L</i>	list of elements
<i>ele</i>	operator to be found in list

## Returns

1 if operator *ele* is found in list, otherwise 0

**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

<i>list</i>	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

<i>tblName</i>	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 already exist. If there is no value in bitmap index or bitmap index on this value, warning is showed to the user. Otherwise, bitmap index is updated with new attribute value.

**Author**

Saša Vukšić

**Parameters**

<i>addBlock</i>	adress of block
<i>addTD</i>	adress of tuple dict
<i>tableName</i>	name of table
<i>attributeName</i>	name of attribute
<i>attributeValue</i>	value of attribute
<i>newAttributeValue</i>	new value of updated attribute

**Returns**

No return value

**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

<i>block</i>	block to write on
--------------	-------------------

## Returns

EXIT\_SUCCESS 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

<i>block</i>	pointer 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

Andelko Spevec

#### Parameters

<i>tblName</i>	- name of the table on which we are creating index
<i>attributes</i>	- attribute on which we are creating index
<i>indexName</i>	- name of the index

#### 7.35.2.2 AK\_btree\_delete()

```
int AK_btree_delete (
    char * indexName )
```

Function that deletes index.

**Author**

unknown

**Parameters**

<i>indexName</i>	- 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**

<i>indexName</i>	- name of the index
<i>insertValue</i>	- value for insert
<i>insertTd</i>	- index table destination
<i>insertBlock</i>	- block address
<i>inputBlock</i>	- block containing btree
<i>EXIT_SUCCESS</i>	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**

Andelko Spevec

**Parameters**

<i>indexName</i>	- name of the index
<i>searchValue</i>	- value that we are searching in the index
<i>endRange</i>	- if 0 search is for 0 value, else searching in range
<i>toDo</i>	- if 0 we just search else we delete the element if we find it

**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**

Andelko Spevec

**Parameters**

<i>temp</i>	- node for deletion
<i>block</i>	- block that contains binary tree
<i>idNext</i>	- 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

<i>temp_node_one</i>	- node that has it's values set
<i>temp</i>	- node with data about existing node
<i>id</i>	- value to which a pointer of a node is to be set
<i>nodeInserted</i>	- determines if a node has value
<i>nodeIncrease</i>	- shows node that is currently watched
<i>number</i>	- determines the way node values are checked
<i>pointerIndex</i>	- 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

<i>temp_node_one</i>	- node that has it's values set
<i>block</i>	- block containing btree
<i>helpAddress</i>	- address of current node
<i>helpType</i>	- type of current node
<i>value_help</i>	- 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

<i>temp_help</i>	- node that has it's values set
<i>insertValue</i>	- value for insert
<i>insertTd</i>	- index table destination
<i>insertBlock</i>	- block address
<i>i</i>	- determines 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**

<i>inserted</i>	- determines if a value in a tree smaller then the value for insert has been found
<i>insertValue</i>	- value for insert
<i>temp</i>	- node with data about existing node
<i>temp_help</i>	- node that has it's values set
<i>insertTd</i>	- index table destination
<i>insertBlock</i>	- block address
<i>increase</i>	- determines the index of element of a node
<i>number</i>	- 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**

<i>temp</i>	- node with data about existing node
<i>temp_help</i>	- node that has it's values set
<i>pointerIndex</i>	- indicates what pointer is used
<i>secondValue</i>	- value of a node
<i>firstPointer</i>	- value to which a pointer of a node is to be set
<i>secondPointer</i>	- 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 structures, 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

Andelko Spevec

#### Parameters

<i>tblName</i>	- name of the table on which we are creating index
<i>attributes</i>	- attribute on which we are creating index
<i>indexName</i>	- name of the index

### 7.36.3.2 AK\_btree\_delete()

```
int AK_btree_delete (
    char * indexName )
```

Function that deletes index.

#### Author

unknown

#### Parameters

<i>indexName</i>	- 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

<i>indexName</i>	- name of the index
<i>insertValue</i>	- value for insert
<i>insertTd</i>	- index table destination
<i>insertBlock</i>	- block address
<i>inputBlock</i>	- block containing btree
<i>EXIT_SUCCESS</i>	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

Andelko Spevec

#### Parameters

<i>indexName</i>	- name of the index
<i>searchValue</i>	- value that we are searching in the index
<i>endRange</i>	- if 0 search is for 0 value, else searching in range
<i>todo</i>	- if 0 we just search else we delete the element if we find it

### 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

Andelko Spevec

#### Parameters

<i>temp</i>	- node for deletion
<i>block</i>	- block that contains binary tree
<i>idNext</i>	- 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

<i>temp_node_one</i>	- node that has it's values set
<i>temp</i>	- node with data about existing node
<i>id</i>	- value to which a pointer of a node is to be set
<i>nodeInserted</i>	- determines if a node has value
<i>nodeIncrease</i>	- shows node that is currently watched
<i>number</i>	- determines the way node values are checked
<i>pointerIndex</i>	- 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

<i>temp_node_one</i>	- node that has it's values set
<i>block</i>	- block containing btree
<i>helpAddress</i>	- address of current node
<i>helpType</i>	- type of current node
<i>value_help</i>	- 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**

<i>temp_help</i>	- node that has it's values set
<i>insertValue</i>	- value for insert
<i>insertTd</i>	- index table destination
<i>insertBlock</i>	- block address
<i>i</i>	- determines 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

<i>inserted</i>	- determines if a value in a tree smaller then the value for insert has been found
<i>insertValue</i>	- value for insert
<i>temp</i>	- node with data about existing node
<i>temp_help</i>	- node that has it's values set
<i>insertTd</i>	- index table destination
<i>insertBlock</i>	- block address
<i>increase</i>	- determines the index of element of a node
<i>number</i>	- gives information about the number of elements in a leaf

## Returns

node that has it's values set

## 7.36.3.12 setNodePointers()

```
btreenode* setNodePointers (
    btreenode * temp,
    btreenode * temp_help,
    int pointerIndex,
    int secondValue,
    int firstPointer,
    int secondPointer )
```

Function that sets values for node.

## Author

unknown

## Parameters

<i>temp</i>	- node with data about existing node
<i>temp_help</i>	- node that has it's values set
<i>pointerIndex</i>	- indicates what pointer is used
<i>secondValue</i>	- value of a node
<i>firstPointer</i>	- value to which a pointer of a node is to be set
<i>secondPointer</i>	- 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

<i>indexName</i>	name of index
<i>modulo</i>	value for modulo hash function
<i>main_bucket_num</i>	number of main buckets
<i>hash_bucket_num</i>	number of hash buckets

## Returns

No return value

**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

<i>tblName</i>	name of table for which the index is being created
<i>indexName</i>	name of index
<i>attributes</i>	list of attributes over which the index is being created

## Returns

success or error

**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

<i>indexName</i>	name of index
<i>values</i>	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

<i>elem</i>	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

<i>indexName</i>	name of index
<i>values</i>	list of values (one row) to search in hash index
<i>delete</i>	if delete is 0 then record is only read otherwise it's deleted from hash index

## Returns

address structure with data where the record is in table

## 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

<i>indexName</i>	name of index
<i>values</i>	list of values (one row) to search in hash index

## Returns

address structure with data where the record is in table

## 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**

<i>indexName</i>	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**

<i>indexName</i>	name of index
<i>n</i>	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

<i>indexName</i>	name of index
<i>data</i>	content of bucket stored in char array
<i>type</i>	type of bucket (MAIN_BUCKET or HASH_BUCKET)

#### Returns

address structure with data where the bucket is stored

### 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

<i>indexName</i>	name of index
<i>hashValue</i>	hash value of record that is being inserted
<i>add</i>	address structure with data where the hash bucket is stored

#### Returns

No return value

### 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

<i>add</i>	address of where the bucket is stored
<i>data</i>	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

<i>indexName</i>	name of index
<i>modulo</i>	value for modulo hash function
<i>main_bucket_num</i>	number of main buckets
<i>hash_bucket_num</i>	number of hash buckets

## Returns

No return value

**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

<i>tblName</i>	name of table for which the index is being created
<i>indexName</i>	name of index
<i>attributes</i>	list of attributes over which the index is being created

## Returns

success or error

**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

<i>indexName</i>	name of index
<i>values</i>	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

<i>elem</i>	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**

<i>indexName</i>	name of index
<i>values</i>	list of values (one row) to search in hash index
<i>delete</i>	if delete is 0 then record is only read otherwise it's deleted from hash index

**Returns**

address structure with data where the record is in table

**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**

<i>indexName</i>	name of index
<i>values</i>	list of values (one row) to search in hash index

**Returns**

address structure with data where the record is in table

**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

<i>indexName</i>	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

<i>indexName</i>	name of index
<i>n</i>	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

<i>indexName</i>	name of index
<i>data</i>	content of bucket stored in char array
<i>type</i>	type of bucket (MAIN_BUCKET or HASH_BUCKET)

#### Returns

address structure with data where the bucket is stored

### 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

<i>indexName</i>	name of index
<i>hashValue</i>	hash value of record that is being inserted
<i>add</i>	address structure with data where the hash bucket is stored

#### Returns

No return value

**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**

<i>add</i>	address of where the bucket is stored
<i>data</i>	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

<a href="#">L</a>	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

<i>Deletedelement_op</i>	- address of node to delete
<i>list_ad</i>	*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

<i>list_ad</i>	*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**

<i>*tblName</i>	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**

<i>*tableName</i>	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

<i>row</i>	zero-based row index
<i>column</i>	zero-based column index
<i>*tblName</i>	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

<i>list_ad</i>	*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**

<i>Currentelement_op</i>	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**

<i>Searchedelement_op</i>	address of current note
<i>*L</i>	linked list head

**Returns**

Integer value of current node's order in the list

**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**

<i>Currentelement_op</i>	Address of current node
<i>L</i>	previous element

**Returns**

Address of previous node or 0 if the current node is the head or the list is empty

**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**

<i>tblName</i>	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

<i>list_ad</i>	*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

<i>addBlock</i>	address block
<i>indexTd</i>	index table destination
<i>*attname</i>	attribute name
<i>elementBefore</i>	address of the node after which the new node will be inserted

#### Returns

No return value

### 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

<i>index</i>	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

<i>*tblName</i>	table name
-----------------	------------

#### Returns

No return value

## 7.40 file/idx/index.h File Reference

```
#include "../aux/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

<i>L</i>	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

<i>Deletedelement_op</i>	- address of node to delete
<i>list_ad</i>	*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

<i>list_ad</i>	*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**

<i>*tableName</i>	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**

<i>row</i>	zero-based row index
<i>column</i>	zero-based column index
<i>*tblName</i>	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

<i>list_ad</i>	*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

<i>Currentelement_op</i>	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**

<i>Searchedelement_op</i>	address of current note
<i>*L</i>	linked list head

**Returns**

Integer value of current node's order in the list

**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**

<i>Currentelement_op</i>	Address of current node
<i>L</i>	previous element

**Returns**

Address of previous node or 0 if the current node is the head or the list is empty

**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**

<i>tblName</i>	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**

<i>list_ad</i>	*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

<i>addBlock</i>	address block
<i>indexTd</i>	index table destination
<i>*attname</i>	attribute name
<i>elementBefore</i>	address of the node after which the new node will be inserted

#### Returns

No return value

### 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

<i>index</i>	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

<code>*tblName</code>	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

<i>name</i>	name of the sequence
<i>start_value</i>	start value of the sequence
<i>increment</i>	increment of the sequence
<i>max_value</i>	maximum value of the sequence
<i>min_value</i>	minimum value of the sequence
<i>cycle</i>	0:non-cyclic sequence, 1:cyclic sequence

#### Returns

sequence\_id or EXIT\_ERROR

### 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

<i>name</i>	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**

<i>name</i>	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**

<i>name</i>	Name of the sequence
<i>start_value</i>	start value of the sequence
<i>increment</i>	increment of the sequence
<i>max_value</i>	maximum value of the sequence
<i>min_value</i>	minimum value of the sequence
<i>cycle</i>	0:non-cyclic sequence, 1:cyclic sequence



**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**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**

<i>name</i>	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**

<i>name</i>	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

<i>old_name</i>	Name of the sequence to be renamed
<i>new_name</i>	New name of the sequence

#### Returns

EXIT\_SUCCESS or EXIT\_ERROR

### 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

<i>name</i>	name of the sequence
<i>start_value</i>	start value of the sequence
<i>increment</i>	increment of the sequence
<i>max_value</i>	maximum value of the sequence
<i>min_value</i>	minimum value of the sequence
<i>cycle</i>	0:non-cyclic sequence, 1:cyclic sequence

**Returns**

sequence\_id or EXIT\_ERROR

**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**

<i>name</i>	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**

<i>name</i>	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

<i>name</i>	Name of the sequence
<i>start_value</i>	start value of the sequence
<i>increment</i>	increment of the sequence
<i>max_value</i>	maximum value of the sequence
<i>min_value</i>	minimum value of the sequence
<i>cycle</i>	0:non-cyclic sequence, 1:cyclic sequence

##### Returns

EXIT\_SUCCESS or EXIT\_ERROR

#### 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

<i>name</i>	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

<i>name</i>	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

<i>old_name</i>	Name of the sequence to be renamed
<i>new_name</i>	New name of the sequence

##### Returns

EXIT\_SUCCESS or EXIT\_ERROR

##### Author

Boris Kišić

## Parameters

<i>old_name</i>	Name of the sequence to be renamed
<i>new_name</i>	New name of the sequence

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

## 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 attribute.*

- 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**

<i>tbl1_temp_block</i>	first cache block of the first table
<i>tbl2_temp_block</i>	first cache block of the second table
<i>operator_name</i>	the name of operator, used for displaying error message

**Returns**

if success returns num of attributes in schema, else returns EXIT\_ERROR

**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**

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

**Returns**

No return value

**7.43.2.4 AK\_get\_attr\_index()**

```
int AK_get_attr_index (
    char * tblName,
    char * attrName )
```

Function that fetches zero-based index for attribute.

**Author**

Matija Šestak.

**Parameters**

<i>*tblName</i>	table name
<i>*attrName</i>	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**

<i>*tblName</i>	table name
<i>index</i>	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

<i>num</i>	zero-based column index
<i>*tblName</i>	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

<i>*tblName</i>	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**

<i>*tableName</i>	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

<i>num</i>	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

* <i>table</i>	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

<i>row</i>	zero-based row index
<i>column</i>	zero-based column index
* <i>tblName</i>	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

<i>col_len[]</i>	array of max lengths for each attribute
<i>*row</i>	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

<i>col_len[]</i>	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len[]</i>	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len[]</i>	array of max lengths for each attribute
<i>*row</i>	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

<i>*tblName</i>	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

<i>*tblName</i>	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

<i>old_table_name</i>	old name of the table
<i>new_table_name</i>	new name of the table
<i>old_attr</i>	name of the attribute to rename
<i>new_attr</i>	new name for the attribute to rename

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**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**

<i>*tblName</i>	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**

<i>tblName</i>	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

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

#### Returns

No return value

### 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**

<i>*tuple</i>	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 Palać

**Parameters**

<i>column</i>	index of column attribute
<i>*row</i>	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 attribute.*
- 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**

<i>tbl1_temp_block</i>	first cache block of the first table
<i>tbl2_temp_block</i>	first cache block of the second table
<i>operator_name</i>	the name of operator, used for displaying error message

**Returns**

if success returns num of attributes in schema, else returns EXIT\_ERROR

**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**

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

**Returns**

No return value

**7.44.4.4 AK\_get\_attr\_index()**

```
int AK_get_attr_index (
    char * tblName,
    char * attrName )
```

Function that fetches zero-based index for attribute.

**Author**

Matija Šestak.

**Parameters**

<i>*tblName</i>	table name
<i>*attrName</i>	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**

<i>*tblName</i>	table name
<i>index</i>	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

<i>num</i>	zero-based column index
<i>*tblName</i>	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

<i>*tblName</i>	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**

<i>*tableName</i>	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

<i>num</i>	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

* <i>table</i>	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

<i>row</i>	zero-based row index
<i>column</i>	zero-based column index
* <i>tblName</i>	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

<i>col_len[]</i>	array of max lengths for each attribute
<i>*row</i>	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

<i>col_len[]</i>	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len[]</i>	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len[]</i>	array of max lengths for each attribute
<i>*row</i>	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

<i>*tblName</i>	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

<i>*tblName</i>	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

<i>old_table_name</i>	old name of the table
<i>new_table_name</i>	new name of the table
<i>old_attr</i>	name of the attribute to rename
<i>new_attr</i>	new name for the attribute to rename

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**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**

<i>*tblName</i>	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

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

## Returns

No return value

**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

<i>*tuple</i>	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

<i>column</i>	index of column attribute
<i>*row</i>	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 attribute.*
- 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

<i>tbl1_temp_block</i>	first cache block of the first table
<i>tbl2_temp_block</i>	first cache block of the second table
<i>operator_name</i>	the name of operator, used for displaying error message

#### Returns

if success returns num of attributes in schema, else returns EXIT\_ERROR

### 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

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

#### Returns

No return value

### 7.45.1.4 AK\_get\_attr\_index()

```
int AK_get_attr_index (
    char * tblName,
    char * attrName )
```

Function that fetches zero-based index for attribute.

#### Author

Matija Šestak.

#### Parameters

<i>*tblName</i>	table name
<i>*attrName</i>	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

<i>*tblName</i>	table name
<i>index</i>	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

<i>num</i>	zero-based column index
<i>*tblName</i>	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**

<i>*tblName</i>	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**

<i>*tableName</i>	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

<i>num</i>	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

* <i>table</i>	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**

<i>row</i>	zero-based row index
<i>column</i>	zero-based column index
<i>*tblName</i>	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**

<i>*</i>	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**

<i>col_len[]</i>	array of max lengths for each attribute
<i>*row</i>	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**

<i>col_len[]</i>	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len[]</i>	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len[]</i>	array of max lengths for each attribute
<i>*row</i>	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

<i>*tblName</i>	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

<i>*tblName</i>	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

<i>old_table_name</i>	old name of the table
<i>new_table_name</i>	new name of the table
<i>old_attr</i>	name of the attribute to rename
<i>new_attr</i>	new name for the attribute to rename

#### Returns

EXIT\_ERROR or EXIT\_SUCCESS

#### 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

<i>*tblName</i>	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**

<i>tblName</i>	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**

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

**Returns**

No return value

**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**

<i>*tuple</i>	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**

<i>column</i>	index of column attribute
<i>*row</i>	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 attribute.*
- 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**

<i>tbl1_temp_block</i>	first cache block of the first table
<i>tbl2_temp_block</i>	first cache block of the second table
<i>operator_name</i>	the name of operator, used for displaying error message

**Returns**

if success returns num of attributes in schema, else returns EXIT\_ERROR

**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**

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

**Returns**

No return value

**7.46.3.4 AK\_get\_attr\_index()**

```
int AK_get_attr_index (
    char * tblName,
    char * attrName )
```

Function that fetches zero-based index for attribute.

**Author**

Matija Šestak.

**Parameters**

<i>*tblName</i>	table name
<i>*attrName</i>	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**

<i>*tblName</i>	table name
<i>index</i>	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

<i>num</i>	zero-based column index
<i>*tblName</i>	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

<i>*tblName</i>	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**

<i>*tableName</i>	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

<i>num</i>	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

<i>col_len[]</i>	array of max lengths for each attribute
<i>*row</i>	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

<i>col_len[]</i>	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len[]</i>	max lengths for each attribute cell
<i>length</i>	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**

<i>col_len[]</i>	array of max lengths for each attribute
<i>*row</i>	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

<i>*tblName</i>	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

<i>*tblName</i>	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

<i>old_table_name</i>	old name of the table
<i>new_table_name</i>	new name of the table
<i>old_attr</i>	name of the attribute to rename
<i>new_attr</i>	new name for the attribute to rename

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**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**

<i>*tblName</i>	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

<i>table</i>	table name
<i>header</i>	<a href="#">AK_header</a> of the new table
<i>type_segment</i>	type of the new segment

## Returns

No return value

**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

<i>*tuple</i>	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

<i>column</i>	index of column attribute
<i>*row</i>	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 between block addresses. It's made for insert\_row()*

- int [AK\\_init\\_new\\_extent](#) (char \*table\_name, int extent\_type)

*Function that extends the segment.*

- int [AK\\_flush\\_cache](#) ()

*Function that flushes memory blocks to disk file.*

- [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**

<i>num</i>	block number (address)
<i>mem_block</i>	address of memmory block

**Returns**

EXIT\_SUCCESS if the block has been successfully read into memory, EXIT\_ERROR otherwise

read the block from the given address

set dirty bit in mem\_block struct

get the timestamp

set timestamp\_read

set timestamp\_last\_change

**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 between block addresses. It's made for insert\_row()

**Author**

Matija Novak, updated by Matija Šestak( function now uses caching)

**Parameters**

<i>address</i>	addresses of extents
----------------	----------------------

**Returns**

address of the block to write in

**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

<i>num</i>	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

<i>index</i>	index name that you search for
--------------	--------------------------------

#### Returns

structure `table_addresses` witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses



### 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

<i>segmentName</i>	table name that you search for
--------------------	--------------------------------

#### Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

### 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

<i>segmentName</i>	table name that you search for
--------------------	--------------------------------

#### Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

### 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**

<i>tableName</i>	table name that you search for
<i>segmentName</i>	segment name

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

**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**

<i>name</i>	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

<i>table</i>	table name that you search for
--------------	--------------------------------

## Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when from and to are 0 you are on the end of addresses

## 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

<i>table_name</i>	name of segment to extent
<i>extent_type</i>	type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP

## Returns

address of new extent, otherwise EXIT\_ERROR

!! to correct header BUG iterate through header from 0 to N-th block while there is

## 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 successfully 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 between block addresses. It's made for [insert\\_row\(\)](#)*
- int [AK\\_init\\_new\\_extent](#) (char \*table\_name, int extent\_type)  
*Function that extends the segment.*
- int [AK\\_flush\\_cache](#) ()  
*Function that flushes memory blocks to disk file.*
- [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

<i>num</i>	block number (address)
<i>mem_block</i>	address of memmory block

## Returns

EXIT\_SUCCESS if the block has been successfully read into memory, EXIT\_ERROR otherwise

read the block from the given address

set dirty bit in mem\_block struct

get the timestamp

set timestamp\_read

set timestamp\_last\_change

### 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 between block addresses. It's made for insert\_row()

## Author

Matija Novak, updated by Matija Šestak( function now uses caching)

## Parameters

<i>address</i>	addresses of extents
----------------	----------------------

**Returns**

address of the block to write in

**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 successfully 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

<i>num</i>	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

<i>index</i>	index name that you search for
--------------	--------------------------------

#### Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when from and to are 0 you are on the end of addresses

#### 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

<i>segmentName</i>	table name that you search for
--------------------	--------------------------------

##### Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

#### 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

<i>segmentName</i>	table name that you search for
--------------------	--------------------------------

##### Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

Function for getting a index segment address.

@Author Antonio Martinović

##### Parameters

<i>segmentName</i>	table name that you search for
--------------------	--------------------------------

### Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

#### 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

<i>tableName</i>	table name that you search for
<i>segmentName</i>	segment name

### Returns

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when form and to are 0 you are on the end of addresses

#### 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

<i>table</i>	table name that you search for
--------------	--------------------------------

**Returns**

structure [table\\_addresses](#) witch contains start and end addresses of table extents, when from and to are 0 you are on the end of addresses

**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**

<i>table_name</i>	name of segment to extent
<i>extent_type</i>	type of extent (can be one of: SEGMENT_TYPE_SYSTEM_TABLE, SEGMENT_TYPE_TABLE, SEGMENT_TYPE_INDEX, SEGMENT_TYPE_TRANSACTION, SEGMENT_TYPE_TEMP

**Returns**

address of new extent, otherwise EXIT\_ERROR

!! to correct header BUG iterate through header from 0 to N-th block while there is

**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 wried to DB file -> EXIT\_ERROR

block is clean after successfully 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

<i>*list_query</i>	RA expresion list where we need to apply relational equivalences rules
<i>rel_eq</i>	rel_eq to execute
<i>*FLAGS</i>	flags for relation equivalences (execute rel_eq for given flags)

**Returns**

returns struct `list_node` (RA expresion list) optimized by given relational equivalence rule

**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**

<i>*list_query</i>	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**

<i>*list_query</i>	RA expresion list where we need to apply relational equivalences rules
<i>*FLAGS</i>	flags for relation equivalences (execute rel_eq for given flags)

**Returns**

returns AK\_list (RA expresion list) optimized by all relational equivalence rules provided by FLAGS (commented code can be edited so AK\_list can return the list of lists (lists of different optimization plans), with permutation switched on (DIFF\_PLANS = 1) time for execution will be significantly increased Current implementation without uncommenting code doesn't produce list of list, it rather apply all permutations on the same list

For futher development consider to implement cost estimation for given plan based on returned heuristically optimized list

**7.49.2.4 AK\_query\_optimization\_test()**

```
TestResult AK_query_optimization_test ( )
```

**Author**

Dino Laktašić

**Parameters**

<i>Function</i>	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**

<i>*list_query</i>	RA expresion list where we need to apply relational equivalences rules
<i>rel_eq</i>	rel_eq to execute
<i>*FLAGS</i>	flags for relation equivalences (execute rel_eq for given flags)

**Returns**

returns struct `list_node` (RA expresion list) optimized by given relational equivalence rule

**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**

<i>*list_query</i>	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

<i>*list_query</i>	RA expresion list where we need to apply relational equivalences rules
<i>*FLAGS</i>	flags for relation equivalences (execute rel_eq for given flags)

## Returns

returns AK\_list (RA expresion list) optimized by all relational equivalence rules provided by FLAGS (commented code can be edited so AK\_list can return the list of lists (lists of different optimization plans), with permutation switched on (DIFF\_PLANS = 1) time for execution will be significantly increased Current implementation without uncommenting code doesn't produce list of list, it rather apply all permutations on the same list

For futher development consider to implement cost estimation for given plan based on returned heuristicly optimized list

## 7.50.3.4 AK\_query\_optimization\_test()

```
TestResult AK_query_optimization_test ( )
```

## Author

Dino Laktašić

## Parameters

<i>Function</i>	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

<i>*a</i>	first value
<i>*b</i>	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

<i>*list_rel_eq</i>	RA expresion as the struct <a href="#">list_node</a>
---------------------	--



**Returns**

optimised RA expression 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 expression according to associativity equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

<i>*list_rel_eq</i>	RA expression as the struct <a href="#">list_node</a>
---------------------	---

**Returns**

optimised RA expression as the struct [list\\_node](#)

**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](#)

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

## Typedefs

- typedef struct [cost\\_eval\\_t](#) [cost\\_eval](#)

## Functions

- int [AK\\_compare](#) (const void \*a, const void \*b)  
*Function for Struct cost\_eval comparison.*
- struct [list\\_node](#) \* [AK\\_rel\\_eq\\_assoc](#) (struct [list\\_node](#) \*list\_rel\_eq)  
*Main function for 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**

<i>*a</i>	first value
<i>*b</i>	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**

<i>*list_rel_eq</i>	RA expresion as the struct <a href="#">list_node</a>
---------------------	--

**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**

<i>*list_rel_eq</i>	RA expresion as the struct <a href="#">list_node</a>
---------------------	--

**Returns**

optimised RA expression 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

<i>*list_rel_eq</i>	RA expresion as the struct <a href="#">list_node</a>
---------------------	--

### 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

<i>*cond</i>	condition array that contains condition data
<i>*tblName</i>	name of the table

## Returns

pointer to char array that contains new condition for a given table

### 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**

<code>*list_rel_eq</code>	RA expression as the struct <a href="#">list_node</a>
---------------------------	---

**Returns**

optimised RA expression 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

<i>*list_rel_eq</i>	RA expresion as the struct <a href="#">list_node</a>
---------------------	--

### 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

<i>*cond</i>	condition array that contains condition data
<i>*tblName</i>	name of the table

**Returns**

pointer to char array that contains new condition for a given table

**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**

<i>*list_rel_eq</i>	RA expresion as the struct <a href="#">list_node</a>
---------------------	--

**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

<i>*list_rel_eq</i>	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**

<i>list_elem_attribs</i>	list element containing projection data
<i>list_elem_conds</i>	list element containing selection condition data

**Returns**

EXIT\_SUCCESS if selection uses only attributes retained by projection, else returns EXIT\_FAILURE

**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**

<i>list_elem</i>	list element that contains selection or theta_join condition data
------------------	---

**Returns**

only attributes from selection or theta\_join condition as the AK\_list

**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**

<i>*tblName</i>	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  $L1 \cup L2$   $p[L1 \cup L2](R1 \bowtie R2) = (p[L1](R1)) \bowtie (p[L2](R2))$  Rule 3b. Let  $L1 \cup L2$  be attributes from  $R1$  and  $R2$ , respectively. Let  $L3$  be attributes from  $R1$ , but are not in  $L1 \cup L2$  and let  $L4$  be attributes from  $R2$ , but are not in  $L1 \cup L2$ .  $p[L1 \cup L2](R1 \bowtie R2) = p[L1 \cup L2]((p[L1 \cup L3](R1)) \bowtie (p[L2 \cup L4](R2)))$  Rule 4. distribution according to union  $p[L](R1 \cup R2) = (p[L](R1)) \cup (p[L](R2))$

**Author**

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is reached
8. on loop exit return EXIT\_SUCCESS

**Parameters**

<i>list_elem_set</i>	first list element containing projection attributes
<i>list_elem_subset</i>	second list element containing projection attributes

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**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

<i>*list_rel_eq</i>	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

<i>*attribs</i>	projection attributes delimited by ";" (ATTR_DELIMITER)
<i>*tblName</i>	name of the table

#### Returns

filtered list of projection attributes as the AK\_list

### 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 * attrs )
```

Function which removes duplicate attributes from attributes expresion.

**Author**

Dino Laktašić.

**Parameters**

<i>*attrs</i>	attributes from which to remove duplicates
---------------	--

**Returns**

pointer to char array without duplicate attributes

**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\_attrs, 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 \*attrs, 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

<i>*list_rel_eq</i>	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**

<i>list_elem_attrbs</i>	list element containing projection data
<i>list_elem_conds</i>	list element containing selection condition data

**Returns**

EXIT\_SUCCESS if selection uses only attributes retained by projection, else returns EXIT\_FAILURE

**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**

<i>list_elem</i>	list element that contains selection or theta_join condition data
------------------	---

**Returns**

only attributes from selection or theta\_join condition as the AK\_list

**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**

<i>*tblName</i>	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 reached
8. on loop exit return EXIT\_SUCCESS

**Parameters**

<i>list_elem_set</i>	first list element containing projection attributes
<i>list_elem_subset</i>	second list element containing projection attributes

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**Author**

Dino Laktašić. =====> Optimization plan using Relational Algebra Equivalences <=====  
Equivalence rule that apply on every equivalent expresion generated by Query optimizer

Rules to implement Rule 1. projection comutes with selection that only uses attributes retained by the projection  $p[L](s[L1](R)) = s[L1](p[L](R))$  Rule 2. only the last in a sequence of projection operations is needed, the others can be omitted.  $p[L1] = p[L1](R)$  Rule 3a. distribution according to theta join, only if join includes attributes from  $L1 \cup L2$   $p[L1 \cup L2](R1 \bowtie R2) = (p[L1](R1)) \bowtie (p[L2](R2))$  Rule 3b. Let  $L1 \cup L2$  be attributes from  $R1$  and  $R2$ , respectively. Let  $L3$  be attributes from  $R1$ , but are not in  $L1 \cup L2$  and let  $L4$  be attributes from  $R2$ , but are not in  $L1 \cup L2$ .  $p[L1 \cup L2](R1 \bowtie R2) = p[L1 \cup L2]((p[L1 \cup L3](R1)) \bowtie (p[L2 \cup L4](R2)))$  Rule 4. distribution according to union  $p[L](R1 \cup R2) = (p[L](R1)) \cup (p[L](R2))$

**Author**

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is ritched
8. on loop exit return EXIT\_SUCCESS

**Parameters**

<i>list_elem_set</i>	first list element containing projection attributes
<i>list_elem_subset</i>	second list element containing projection attributes

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**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

<i>*list_rel_eq</i>	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

<i>*attribs</i>	projection attributes delimited by ";" (ATTR_DELIMITER)
<i>*tblName</i>	name of the table

### Returns

filtered list of projection attributes as the AK\_list

#### 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 * attrs )
```

Function which removes duplicate attributes from attributes expresion.

**Author**

Dino Laktašić.

**Parameters**

<i>*attrs</i>	attributes from which to remove duplicates
---------------	--

**Returns**

pointer to char array without duplicate attributes

**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\\_attributes\\_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

* <a href="#">list_rel_eq</a>	RA expresion as the struct <a href="#">list_node</a>
-------------------------------	--

##### 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**

<i>*cond</i>	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\_attributes\_char()**

```
char* AK_rel_eq_get_attributes_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**

<i>*tblName</i>	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 reached
8. on loop exit return EXIT\_SUCCESS

**Parameters**

<i>*set</i>	set array
<i>*subset</i>	subset array

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**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 expression according to selection equivalence rules.

**Author**

Dino Laktašić.

**Parameters**

<i>*list_rel_eq</i>	RA expression as the struct <code>list_node</code>
---------------------	--

**Returns**

optimised RA expression 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**

<i>*set</i>	first set of attributes delimited by ";" (ATTR_DELIMITER)
<i>*subset</i>	second set of attributes delimited by ";" (ATTR_DELIMITER)

**Returns**

EXIT\_SUCCESS if set and subset share at least one attribute, else returns EXIT\_FAILURE

**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 discharge collected conds
6. else append token to collected data
7. When exits from loop if id greater then 0, append the last collected data to result
8. return pointer to char array that contains new condition for a given table

**Parameters**

<i>*cond</i>	condition array that contains condition data
<i>*tblName</i>	name of the table

**Returns**

pointer to char array that contains new condition for a given table

**Author**

Dino Laktašić.

Break conjunctive conditions to individual conditions (currently not used - commented in main AK\_rel\_eq\_selection function), it can be usefull in some optimization cases

1. For each delimited item (' AND ') insert item to the struct [list\\_node](#)
2. Remove unused pointers and return the conditions list

**Parameters**

<i>*cond</i>	condition expression
--------------	----------------------

**Returns**

conditions list

## 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\\_attributes\\_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

<i>*<a href="#">list_rel_eq</a></i>	RA expresion as the struct <a href="#">list_node</a>
-------------------------------------	--

#### 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

<i>*cond</i>	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\_attributes\_char()

```
char* AK_rel_eq_get_attributes_char (
    char * tblName )
```

Function that fetches attributes for a given table and store them to the char array.

#### Author

Dino Laktašić.

#### Parameters

<i>*tblName</i>	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**

<i>*tblName</i>	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**

<i>*set</i>	set array
<i>*subset</i>	subset array

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**Author**

Dino Laktašić.

1. Tokenize set and subset of projection attributes and store each of them to it's own array
2. Check if the size of subset array is larger than the size of set array
3. if the subset array is larger return 0
4. else sort both arrays ascending
5. Compare the subset and set items at the same positions, starting from 0
6. if there is an item in the subset array that doesn't match attribute at the same position in the set array return 0
7. else continue comparing until final item in the subset array is reached
8. on loop exit return EXIT\_SUCCESS

**Parameters**

<i>*set</i>	set array
<i>*subset</i>	subset array

**Returns**

EXIT\_SUCCESS if some set of attributes is subset of larger set, else returns EXIT\_FAILURE

**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**

<i>*list_rel_eq</i>	RA expresion as the struct <a href="#">list_node</a>
---------------------	--

**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**

<i>*set</i>	first set of attributes delimited by ";" (ATTR_DELIMITER)
<i>*subset</i>	second set of attributes delimited by ";" (ATTR_DELIMITER)

**Returns**

EXIT\_SUCCESS if set and subset share at least one attribute, else returns EXIT\_FAILURE

**Author**

Dino Laktašić.

1. If is empty set or subset returns EXIT\_FAILURE
2. For each attribute in one set check if there is same attribute in the second set
3. If there is the same attribute return EXIT\_SUCCESS
4. else remove unused pointers and return EXIT\_FAILURE

**Parameters**

<i>*set</i>	first set of attributes delimited by ";" (ATTR_DELIMITER)
<i>*subset</i>	second set of attributes delimited by ";" (ATTR_DELIMITER)

**Returns**

EXIT\_SUCCESS if set and subset share at least one attribute, else returns EXIT\_FAILURE

**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**

<i>*cond</i>	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 discharge collected conds
6. else append token to collected data
7. When exits from loop if id greater then 0, append the last collected data to result
8. return pointer to char array that contains new condition for a given table

#### Parameters

<i>*cond</i>	condition array that contains condition data
<i>*tblName</i>	name of the table

#### Returns

pointer to char array that contains new condition for a given table

#### Author

Dino Laktašić.

Break conjunctive conditions to individual conditions (currently not used - commented in main AK\_rel\_eq\_selection function), it can be usefull in some optimization cases

1. For each delimited item (' AND ') insert item to the struct [list\\_node](#)
2. Remove unused pointers and return the conditions list

#### Parameters

<i>*cond</i>	condition expression
--------------	----------------------

#### Returns

conditions list

## 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\\_archive\\_log](#) ()

### 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 return value

#### 7.59.1.2 [AK\\_check\\_folder\\_archive\\_log\(\)](#)

```
int AK_check_folder_archive_log ( )
```

### 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 "../aux/mempro.h"
```

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

### Functions

- void [AK\\_archive\\_log](#) (int sig)  
*Function for making archive log.*
- 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 return 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 return 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 necessary actions.

#### Author

Matija Večenaj

**Parameters**

<i>none</i>	
-------------	--

**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 necessary recovery operations.

**Author**

Matija Večenaj

**Parameters**

<i>none</i>	
-------------	--

**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**

<i>fileName</i>	- 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**

<i>sig</i>	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**

<i>table</i>	- table name to insert to
<i>commandNumber</i>	- 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**

<i>input</i>	- input to tokenize
<i>delimiter</i>	- delimiter
<i>valuesOrNot</i>	- 1 if the input are values, 0 otherwise

**Returns**

new double pointer structure with tokens

### 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 necessary actions.

#### Author

Matija Večenaj

#### Parameters

<i>none</i>	
-------------	--

#### 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 necessary actions.

#### Author

Matija Večenaj

#### Parameters

<i>none</i>	
-------------	--

#### 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 necessary recovery operations.

**Author**

Matija Večenaj

**Parameters**

<i>none</i>	
-------------	--

**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**

<i>fileName</i>	- 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

<i>sig</i>	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

<i>table</i>	- table name to insert to
<i>commandNumber</i>	- 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

<i>input</i>	- input to tokenize
<i>delimiter</i>	- delimiter
<i>valuesOrNot</i>	- 1 if the input are values, 0 otherwise

#### Returns

new double pointer structure with tokens

## 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 assigns the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed\_values array and results are put in new table.*
- [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

<i>header</i>	a header that is being aggregated
<i>agg_task</i>	the task which is to be done on the header
<i>input</i>	the input object

#### Returns

On success, returns EXIT\_SUCCESS, otherwise EXIT\_FAILURE

### 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

<i>header</i>	a header that is being aggregated
<i>agg_task</i>	the task which is to be done on the header
<i>input</i>	the input object

#### Returns

On success, returns EXIT\_SUCCESS, otherwise EXIT\_FAILURE

### 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\_TASK\_AVG\_COUNT is put on the beginning of input object. After that, AGG\_TASK\_AVG\_SUM is put on the beginning of input object.

#### Author

Dejan Frankovic

#### Parameters

<i>input</i>	the input object
--------------	------------------

#### Returns

No return value

### 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

<i>input</i>	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 assigns the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed\_values array and results are put in new table.

#### Author

Dejan Frankovic

#### Parameters

<i>input</i>	input object with list of attributes by which we aggregate and types of aggregations
<i>source_table</i>	- table name for the source table
<i>agg_table</i>	table name for aggregated table

#### Returns

EXIT\_SUCCESS if continues successfully, 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

<i>header</i>	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\_TIMESTAMP. Do not provide the wrong data types in the array of search parameters. There is no way to test for that and it could cause a memory access violation.

#### Author

Miroslav Policki

#### Parameters

<i>szRelation</i>	relation name
<i>aspParams</i>	array of search parameters
<i>iNum_search_params</i>	number of search parameters

#### Returns

[search\\_result](#) structure defined in [filesearch.h](#). Use [AK\\_deallocate\\_search\\_result](#) to deallocate.

iterate through all the blocks

count number of attributes in segment/relation

determine index of attributes on which search will be performed

if any of the provided attributes are not found in the relation, return empty result

in every tuple, for all required attributes, compare attribute value with searched-for value and store matched tuple addresses

## 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 attribute name, date and aggregation task associated.*
- struct [AK\\_agg\\_input](#)  
*Structure that contains attributes from table header, tasks for this table and counter value.*
- 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 assigns the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed\_values array and results are put in new table.*
- [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

<i>header</i>	a header that is being aggregated
<i>agg_task</i>	the task which is to be done on the header
<i>input</i>	the input object

## Returns

On success, returns EXIT\_SUCCESS, otherwise EXIT\_FAILURE

**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

<i>header</i>	a header that is being aggregated
<i>agg_task</i>	the task which is to be done on the header
<i>input</i>	the input object

## Returns

On success, returns EXIT\_SUCCESS, otherwise EXIT\_FAILURE

**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\_TASK\_AVG\_COUNT is put on the beginning of input object. After that, AGG\_TASK\_AVG\_SUM is put on the beginning of input object.

## Author

Dejan Frankovic

**Parameters**

<i>input</i>	the input object
--------------	------------------

**Returns**

No return value

**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**

<i>input</i>	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 assigns the type of aggregation operation according to aggregation operation. New table has to be created. For loop goes through given table. GROUP operation is executed separately from other operations. Addresses of records are put in needed\_values array and results are put in new table.

**Author**

Dejan Frankovic

#### Parameters

<i>input</i>	input object with list of attributes by which we aggregate and types of aggregations
<i>source_table</i>	- table name for the source table
<i>agg_table</i>	table name for aggregated table

#### Returns

EXIT\_SUCCESS if continues successfully, 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

<i>header</i>	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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the new table

## Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

## 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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the new table

#### Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

### 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 character wildcard (%\_) ch in string s with repl characters.
- int [AK\\_check\\_regex\\_expression](#) (const char \*value, const char \*expression, int sensitive, int checkWildCard)  
Function that evaluates regex expression on 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

<i>el</i>	list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation
<i>*op</i>	comparison operator
<i>*a</i>	left operand
<i>*b</i>	right operand

**Returns**

0 if arithmetic statement is false, 1 if arithmetic statement is true

**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 character wildcard (%,\_ ) ch in string s with repl characters.

**Author**

Matija Šestak, updated by Dino Laktašić, Nikola Miljancic, abstracted by Tomislav Mikulček

**Parameters**

<i>row_root</i>	beginning of the row that is to be evaluated
<i>*expr</i>	list with the logical expression in postfix notation

**Returns**

0 if row does not satisfy, 1 if row satisfies expression

**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**

<i>value</i>	string value that must match regex expression
<i>expression</i>	POSIX regex expression
<i>checkWildCard</i>	replaces SQL wildcard to corresponding POSIX regex character
<i>sensitive</i>	case insensitive indicator 1-case sensitive,0- case insensitive



**Returns**

0 if regex didnt match or syntax of regex is incorrecct 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**

<i>value</i>	string value that must match regex expression
<i>expression</i>	POSIX regex expression

**Returns**

0 if regex didnt match or syntax of regex is incorrecct 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**

<i>s</i>	input string
<i>ch</i>	charachter to be replaced

**Returns**

new sequence of characters

**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 character wildcard (%,\_ ) ch in string s with repl characters.*
- int [AK\\_check\\_regex\\_expression](#) (const char \*value, const char \*expression, int sensitive, int checkWildCard)  
*Function that evaluates regex expression on 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**

<i>el</i>	list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation
<i>*op</i>	comparison operator
<i>*a</i>	left operand
<i>*b</i>	right operand

**Returns**

0 if arithmetic statement is false, 1 if arithmetic statement is true

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**

<i>el</i>	list element, last element put in list temp which holds elements of row ordered according to expression and results of their evaluation
<i>*op</i>	comparison operator
<i>*a</i>	left operand
<i>*b</i>	right operand

**Returns**

0 if arithmetic statement is false, 1 if arithmetic statement is true

**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 character wildcard (%,\_ ) ch in string s with repl characters.

@Author Leon Palaić

**Parameters**

<i>s</i>	input string
<i>ch</i>	character to be replaced

**Returns**

new sequence of characters

Function that replaces character wildcard (%\_ ) ch in string s with repl characters.

**Author**

Matija Šestak, updated by Dino Laktašić, Nikola Miljancic, abstracted by Tomislav Mikulček

**Parameters**

<i>row_root</i>	beginning of the row that is to be evaluated
<i>*expr</i>	list with the logical expression in postfix notation

**Returns**

0 if row does not satisfy, 1 if row satisfies expression

**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**

<i>value</i>	string value that must match regex expression
<i>expression</i>	POSIX regex expression
<i>checkWildCard</i>	replaces SQL wildcard to corresponding POSIX regex character
<i>sensitive</i>	case insensitive indicator 1-case sensitive,0- case insensitive

**Returns**

0 if regex didnt match or syntax of regex is incorrect 1 if string matches corresponding regex expression

**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 Palać

#### Parameters

<i>value</i>	string value that must match regex expression
<i>expression</i>	POSIX regex expression

#### Returns

0 if regex didnt match or sytnax of regex is incorcct 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**

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the new table

**Returns**

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

**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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the new table

#### Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

### 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 between 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

<i>tbl1_temp_block</i>	block of the first table
<i>tbl2_temp_block</i>	block of the second join table
<i>att</i>	attributes on which we make nat_join
<i>new_table</i>	name of the nat_join table

## Returns

No return value

## 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

<i>table_address1</i>	address of the block of the first table
<i>table_address2</i>	address of the block of the second table
<i>new_table</i>	name of the join table
<i>att_root</i>	tributes on which we make nat_join

## Returns

No return value

## 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 between two tables on some attributes.

**Author**

Matija Novak, updated to work with AK\_list and support cacheing by Dino Laktašić

**Parameters**

<i>srcTable1</i>	name of the first table to join
<i>srcTable2</i>	name of the second table to join
<i>att</i>	attributes on which we make nat_join
<i>dstTable</i>	name of the nat_join table

**Returns**

if success returns EXIT\_SUCCESS

**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**

<i>row_root</i>	- list of values from the first table to be merged with table2
<i>row_root_insert</i>	- list of values from the first table to be inserted into nat_join table
<i>temp_block</i>	- block from the second table to be merged
<i>new_table</i>	- name of the nat_join table

**Returns**

No return value

**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 between 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

<i>tbl1_temp_block</i>	block of the first table
<i>tbl2_temp_block</i>	block of the second join table
<i>att</i>	attributes on which we make nat_join
<i>new_table</i>	name of the nat_join table

#### Returns

No return value

### 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

<i>table_address1</i>	address of the block of the first table
<i>table_address2</i>	address of the block of the second table
<i>new_table</i>	name of the join table
<i>att_root</i>	tributes on which we make nat_join

### Returns

No return value

#### 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 between two tables on some attributes.

### Author

Matija Novak, updated to work with AK\_list and support cacheing by Dino Laktašić

### Parameters

<i>srcTable1</i>	name of the first table to join
<i>srcTable2</i>	name of the second table to join
<i>att</i>	attributes on which we make nat_join
<i>dstTable</i>	name of the nat_join table

### Returns

if success returns EXIT\_SUCCESS

#### 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**

<i>row_root</i>	- list of values from the first table to be marged with table2
<i>row_root_insert</i>	- list of values from the first table to be inserted into nat_join table
<i>temp_block</i>	- block from the second table to be merged
<i>new_table</i>	- name of the nat_join table

**Returns**

No return value

**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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the product table

#### Returns

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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the product table

#### Returns

destination table filled with data sized  $n(\text{rows srcTable1}) * m(\text{rows srcTable2})$

#### Parameters

<i>header</i>	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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the product table

#### Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

**Author**

Dino Laktašić

**Parameters**

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the product table

**Returns**

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**

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the product table
<i>header</i>	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**

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the product table

**Returns**

destination table filled with data sized  $n(\text{rows srcTable1}) * m(\text{rows srcTable2})$

**Parameters**

<i>header</i>	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 arithmetic 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](#) \*secondOperand, int type)  
*Function that performs 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 arithmetic operation or expression.*

### 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

<i>old_block</i>	block from which we copy data
<i>dstTable</i>	name of the new table
<i>att</i>	list of the attributes which should the projection table contain
<i>expr</i>	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

<i>old_block_add</i>	address of the block from which we copy headers we need
<i>dstTable</i>	name of the new table - destination table
<i>att</i>	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**

<i>first</i>	operand name
<i>second</i>	operand name
<i>operator</i>	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**

<i>firstOperand</i>	operand type
<i>secondOperand</i>	operand type

**Returns**

Function returns determined 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 arithmetic operations.

**Author**

Leon Palaić

**Parameters**

<i>exp</i>	input expression string
------------	-------------------------

**Returns**

character - arithmetic 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 arithmetic operation or expression.

**Author**

Dino Laktašić, rewritten and optimized by Irena Ilišević to support ILIKE operator and perform usual projection

**Returns**

Projection tables and number of 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 performs arithmetics operation depended on given operator.

#### Author

Leon Palać

#### Parameters

<i>firstOperand</i>	first operand
<i>secondOperand</i>	second operand
<i>op</i>	aritmetic operator
<i>type</i>	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

<i>srcTable</i>	source table - table on which projection is made
<i>expr</i>	given expression to check while doing projection
<i>att</i>	list of atributes on which we make projection
<i>dstTable</i>	table name for projection table - new table - destination table

**Returns**

Projection table on given attributes

EXIT\_SUCCESS if continues succesfully, 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**

<i>s</i>	input string
<i>substring</i>	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 arithmetic 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 performs 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 arithmetic operation or expression.*

## 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

<i>old_block</i>	block from which we copy data
<i>dstTable</i>	name of the new table
<i>att</i>	list of the attributes which should the projection table contain
<i>expr</i>	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**

<i>old_block_add</i>	address of the block from which we copy headers we need
<i>dstTable</i>	name of the new table - destination table
<i>att</i>	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 Palać

**Parameters**

<i>first</i>	operand name
<i>second</i>	operand name
<i>operator</i>	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

<i>firstOperand</i>	operand type
<i>secondOperand</i>	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 arithmetic operations.

### Author

Leon Palaić

### Parameters

<i>exp</i>	input expression string
------------	-------------------------

**Returns**

character - arithmetic operator  
character

**Author**

Leon Palać

**Parameters**

<i>exp</i>	input expression string
------------	-------------------------

**Returns**

character - arithmetic 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 arithmetic operation or expression.

**Author**

Dino Laktašić, rewritten and optimized by Irena Ilišević to support ILIKE operator and perform usual projection

**Returns**

Projection tables and number of 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 performs arithmetic operation depending on given operator.

**Author**

Leon Palać

## Parameters

<i>firstOperand</i>	first operand
<i>secondOperand</i>	second operand
<i>op</i>	aritmetic operator
<i>type</i>	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

<i>srcTable</i>	source table - table on which projection is made
<i>expr</i>	given expression to check while doing projection
<i>att</i>	list of atributes on which we make projection
<i>dstTable</i>	table name for projection table - new table - destination table

## Returns

Projection table on given attributes  
EXIT\_SUCCESS if continues succesfully, 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**

<i>s</i>	input string
<i>substring</i>	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

<i>*srcTable</i>	source table name
<i>*dstTable</i>	destination table name
<i>*expr</i>	list with posfix notation of the logical expression

#### Returns

EXIT\_SUCCESS

### 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**

<i>*srcTable</i>	source table name
<i>*dstTable</i>	destination table name
<i>*expr</i>	list with postfix notation of the logical expression

**Returns**

EXIT\_SUCCESS

**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

<i>*srcTable</i>	source table name
<i>*dstTable</i>	destination table name
<i>*expr</i>	list with posfix notation of the logical expression

#### Returns

EXIT\_SUCCESS

## 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 between two tables on specified conditions. Names of the attributes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.*

- [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

<i>tbl1_temp_block</i>	block of the first table
<i>tbl2_temp_block</i>	block of the second join table
<i>tbl1_num_att</i>	number of attributes in the first table
<i>tbl2_num_att</i>	number of attributes in the second table
<i>constraints</i>	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
<i>new_table</i>	name of the theta_join table

#### Returns

No return value

### 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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>new_table</i>	name of the destination table

#### Returns

EXIT\_SUCCESS if the header was successfully created and EXIT\_ERROR if the renamed headers are too long

### 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 between two tables on specified conditions. Names of the attributes in the constraints parameter must be prefixed with the table name followed by a dot if and only if they exist in both tables. This is left for the preprocessing. Also, for now the constraints must come from the two source tables and not from a third.

Function that creates a theta join between two tables on specified conditions.

#### Author

Tomislav Mikulček, updated by Nikola Miljancic

#### Parameters

<i>srcTable1</i>	name of the first table to join
<i>srcTable2</i>	name of the second table to join
<i>constraints</i>	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
<i>dstTable</i>	name of the theta join table

#### Returns

if successful returns EXIT\_SUCCESS and EXIT\_ERROR otherwise

## 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 between 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

<i>tbl1_temp_block</i>	block of the first table
<i>tbl2_temp_block</i>	block of the second join table
<i>tbl1_num_att</i>	number of attributes in the first table
<i>tbl2_num_att</i>	number of attributes in the second table
<i>constraints</i>	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
<i>new_table</i>	name of the theta_join table

#### Returns

No return value

### 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**

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>new_table</i>	name of the destination table

**Returns**

EXIT\_SUCCESS if the header was successfully created and EXIT\_ERROR if the renamed headers are too long

**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 between two tables on specified conditions.

**Author**

Tomislav Mikulček, updated by Nikola Miljancic

## Parameters

<i>srcTable1</i>	name of the first table to join
<i>srcTable2</i>	name of the second table to join
<i>constraints</i>	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
<i>dstTable</i>	name of the theta join table

## Returns

if successful returns EXIT\_SUCCESS and EXIT\_ERROR otherwise

Function that creates a theta join between two tables on specified conditions.

## Author

Tomislav Mikulček, updated by Nikola Miljancic

## Parameters

<i>srcTable1</i>	name of the first table to join
<i>srcTable2</i>	name of the second table to join
<i>constraints</i>	list of attributes, (in)equality and logical operators which are the conditions for the join in postfix notation
<i>dstTable</i>	name of the theta join table

## Returns

if successful returns EXIT\_SUCCESS and EXIT\_ERROR otherwise

## 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

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the new table

#### Returns

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

## 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**

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the new table

**Returns**

if success returns EXIT\_SUCCESS, else returns EXIT\_ERROR

Function that makes a union of two tables.

**Author**

Dino Laktašić

**Parameters**

<i>srcTable1</i>	name of the first table
<i>srcTable2</i>	name of the second table
<i>dstTable</i>	name of the 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

<i>commands</i>	Commands array to execute
<i>commandNum</i>	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

<i>commands</i>	Commands array to execute
<i>commandNum</i>	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 written in lowercase. It searches for AK\_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.*
- 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 constraint

### 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

<i>tableName</i>	name of table on which constraint refers
<i>attName</i>	name of attribute on which constraint is declared
<i>constraintName</i>	name of constraint

#### Returns

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

### 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

<i>_systemTableName</i>	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

<i>tableName</i>	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

<i>tableName</i>	table name
<i>newValue</i>	value we want to insert
<i>attNamePar</i>	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 written in lowercase. It searches for AK\_free space. Then it inserts id, name of table, name of constraint, name of attribute, start and end value in temporary block.

Function that sets between constraints on particular attribute, string constraint should be written in lowercase.

**Author**

Saša Vukšić, updated by Mislav Jurinić, updated by Blaž Rajič

**Parameters**

<i>tableName</i>	table name
<i>constraintName</i>	name of constraint
<i>attName</i>	name of attribute
<i>startValue</i>	initial constraint
<i>endValue</i>	final constraint

**Returns**

No return value

**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 written 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 constraint

## 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**

<i>tableName</i>	name of table on which constraint refers
<i>attName</i>	name of attribute on which constraint is declared
<i>constraintName</i>	name of constraint

**Returns**

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

**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**

<i>_systemTableName</i>	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

<i>tableName</i>	table name
<i>newValue</i>	value we want to insert
<i>attNamePar</i>	attribute name

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

## Author

Saša Vukšić, updated by Mislav Jurinić, updated by Blaž Rajič

## Parameters

<i>tableName</i>	table name
<i>newValue</i>	value we want to insert
<i>attNamePar</i>	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 written in lowercase.

## Author

Saša Vukšić, updated by Mislav Jurinić

## Parameters

<i>tableName</i>	table name
<i>constraintName</i>	name of constraint
<i>attName</i>	name of attribute
<i>startValue</i>	initial constraint
<i>endValue</i>	final constraint

**Returns**

No return value

Function that sets between constraints on particular attribute, string constraint should be written in lowercase.

**Author**

Saša Vukšić, updated by Mislav Jurinić, updated by Blaž Rajič

**Parameters**

<i>tableName</i>	table name
<i>constraintName</i>	name of constraint
<i>attName</i>	name of attribute
<i>startValue</i>	initial constraint
<i>endValue</i>	final constraint

**Returns**

No return value

## 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

<i>table</i>	target table name
<i>attribute</i>	target attribute name
<i>value</i>	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

<i>tableName</i>	System table where constraint will be deleted from
<i>constraintName</i>	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

<i>table_name</i>	target table for "check" constraint evaluation
<i>constraint_name</i>	new "check" constraint name that will be visible in the system table
<i>attribute_name</i>	target attribute for "check" constraint evaluation
<i>condition</i>	logical operator ['<', '>', '!=', ...]
<i>type</i>	data type [int, float, varchar, datetime, ...]
<i>value</i>	condition to be set

**Returns**

1 - 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**

<i>condition</i>	logical operator ['<', '>', '!=', ...]
<i>type</i>	data type [int, float, varchar, datetime, ...]
<i>value</i>	condition to be set
<i>row_data</i>	data in table

**Returns**

1 - 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

<i>table</i>	target table name
<i>attribute</i>	target attribute name
<i>value</i>	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**

<i>tableName</i>	System table where constraint will be deleted from
<i>constraintName</i>	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**

<i>table_name</i>	target table for "check" constraint evaluation
<i>constraint_name</i>	new "check" constraint name that will be visible in the system table
<i>attribute_name</i>	target attribute for "check" constraint evaluation
<i>condition</i>	logical operator ['<', '>', '!=', ...]
<i>type</i>	data type [int, float, varchar, datetime, ...]
<i>value</i>	condition to be set

**Returns**

1 - 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

<i>condition</i>	logical operator ['<', '>', '!=', ...]
<i>type</i>	data type [int, float, varchar, datetime, ...]
<i>value</i>	condition to be set
<i>row_data</i>	data in table

#### Returns

1 - 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**

<code>char</code>	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

<i>char</i>	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

<i>char</i>	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

<i>char*</i>	tableName name of table
<i>char*</i>	attName name of attribute
<i>char*</i>	constraintName name of constraint

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**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**

<i>tableName</i>	System table where constraint will be deleted from
<i>constraintName</i>	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

<i>char*</i>	tableName name of table
<i>char*</i>	attName name of attribute
<i>char*</i>	newValue new value

## Returns

EXIT\_ERROR or EXIT\_SUCCESS

**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

<i>char*</i>	tableName name of table
<i>char*</i>	attName name of attribute
<i>char*</i>	constraintName name of constraint

## Returns

EXIT\_ERROR or EXIT\_SUCCESS

**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

<i>char*</i>	tableName name of table
<i>char*</i>	attName name of attribute
<i>char*</i>	constraintName name of constraint

#### Returns

EXIT\_ERROR or EXIT\_SUCCESS

### 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**

<i>tableName</i>	name of table on which constraint refers
<i>attName</i>	name of attribute on which constraint is declared
<i>constraintName</i>	name of constraint

**Returns**

EXIT\_SUCCESS when constraint is deleted, else EXIT\_ERROR

**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**

<i>char*</i>	tableName name of table
<i>char*</i>	attName name of attribute
<i>char*</i>	newValue new value

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**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**

<i>char*</i>	tableName name of table
<i>char*</i>	attName name of attribute
<i>char*</i>	constraintName name of constraint

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**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)  
*Function that quickly checks if there are any referential constraints that should be applied on a given list of changes.*
- int [AK\\_reference\\_check\\_restricion](#) (struct [list\\_node](#) \*lista, int action)  
*Function that checks for a REF\_TYPE\_RESTRICT references applicable to the operation of updating or deleting a row in a table.*

- `int AK_reference_update` (struct `list_node` \*lista, int action)  
*Function 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

<i>name</i>	of the child table
<i>array</i>	of child table attribute names (foreign key attributes)
<i>name</i>	of the parent table
<i>array</i>	of parent table attribute names (primary key attributes)
<i>number</i>	of attributes in foreign key
<i>name</i>	of the constraint
<i>type</i>	of the constraint, constants defined in ' <a href="#">reference.h</a> '

#### Returns

EXIT\_SUCCESS

### 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

<i>name</i>	of the table with reference (with foreign key)
<i>name</i>	of the reference constraint

#### Returns

[AK\\_ref\\_item](#) object with all necessary information about the reference

### 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

<i>child</i>	table name
<i>attribute</i>	name (foreign key attribute)
<i>value</i>	of the attribute we're checking

#### Returns

EXIT\_ERROR if check failed, EXIT\_SUCCESS if referential integrity is ok

#### 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

<i>list</i>	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

<i>list</i>	of elements for update
<i>is</i>	action UPDATE or DELETE ?

##### Returns

EXIT\_SUCCESS if update is needed, EXIT\_ERROR if not

#### 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 applicable to the operation of updating or deleting a row in a table.

**Author**

Dejan Franković

**Parameters**

<i>list</i>	of elements for update
<i>is</i>	action UPDATE or DELETE?

**Returns**

EXIT\_SUCCESS if there is no restriction on this action, EXIT\_ERROR if there is

**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

<i>list</i>	of elements for update
<i>is</i>	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 applicable 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

d 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

<i>name</i>	of the child table
<i>array</i>	of child table attribute names (foreign key attributes)
<i>name</i>	of the parent table
<i>array</i>	of parent table attribute names (primary key attributes)
<i>number</i>	of attributes in foreign key
<i>name</i>	of the constraint
<i>type</i>	of the constraint, constants defined in ' <a href="#">reference.h</a> '

**Returns**

EXIT\_SUCCESS

**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**

<i>row_root</i>	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**

<i>name</i>	of the table with reference (with foreign key)
<i>name</i>	of the reference constraint

**Returns**

[AK\\_ref\\_item](#) object with all 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

<i>name</i>	segment name
<i>type</i>	segment type
<i>header</i>	pointer to header that should be written to the new extent (all blocks)

#### Returns

start address of new segment

### 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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	is NEW_VALUE

#### Returns

No return value

### 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 elements are set according to function arguments. Pointers are changed so that before element points to new element.

#### Author

Matija Novak

#### Parameters

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	NEW_VALUE if data is new value, SEARCH_CONSTRAINT if data is constraint to search for

#### Returns

No return value

### 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 violate reference integrity. Then it is checked in which table should row be inserted. If there is no AK\_free space for new table, new extent is allocated. New block is allocated on given address. Row is inserted in this block and dirty flag is set to BLOCK\_↔DIRTY.

#### Author

Matija Novak, updated by Matija Šestak (function now uses caching), updated by Dejan Frankovic (added reference check), updated by Dino Laktašić (removed variable AK\_free, variable table initialized using memset)

**Parameters**

<i>row_root</i>	list of elements which contain data of one row
-----------------	--

**Returns**

EXIT\_SUCCESS if success else EXIT\_ERROR

**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**

<i>child</i>	table name
<i>attribute</i>	name (foreign key attribute)
<i>value</i>	of the attribute we're checking

**Returns**

EXIT\_ERROR if check failed, EXIT\_SUCCESS if referential integrity is ok

**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**

<i>list</i>	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**

<i>list</i>	of elements for update
<i>is</i>	action UPDATE or DELETE ?

**Returns**

EXIT\_SUCCESS if update is needed, EXIT\_ERROR if not

**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**

<i>list</i>	of elements for update
<i>is</i>	action UPDATE or DELETE?

**Returns**

EXIT\_SUCCESS if there is no restriction on this action, EXIT\_ERROR if there is

**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**

<i>list</i>	of elements for update
<i>is</i>	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

<i>*srcTable</i>	source table name
<i>*dstTable</i>	destination table name
<i>*expr</i>	list with posfix notation of the logical expression

#### Returns

EXIT\_SUCCESS

### 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

<i>newtype</i>	type of the data
<i>data</i>	the data
<i>table</i>	table name
<i>attribute_name</i>	attribute name
<i>element</i>	element after we which insert the new element
<i>constraint</i>	is NEW_VALUE

**Returns**

No return value

**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**

<i>row_root</i>	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

<i>tableName</i>	name of table on which constraint refers
<i>constraintName</i>	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

<i>char*</i>	tableName name of table
<i>char</i>	attName[] name(s) of attribute(s), if you want to check combination of values of more attributes separate names of attributes with constant SEPARATOR (see test)
<i>char</i>	newValue[] new value(s), if you want to check combination of values of more attributes separate their values with constant SEPARATOR (see test), if some value(s) which you want to check isn't stored as char (string) convert it to char (string) using <a href="#">AK_tuple_to_string(struct list_node *tuple)</a> or with <code>sprintf</code> in a similar way it's used in that function (if value isn't part of a *tuple), to concatenate more values in newValue[] use <code>strcat(destination, source)</code> and put constant SEPARATOR between them (see test) if newValue[] should contain NULL sign pass it as " " (space)

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**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**

<i>char*</i>	tableName name of table
<i>char</i>	attName[] name(s) of attribute(s), if you want to set UNIQUE constraint on combination of attributes separate their names with constant SEPARATOR (see test)
<i>char</i>	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

<i>tableName</i>	name of table on which constraint refers
<i>constraintName</i>	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**

<i>char*</i>	tableName name of table
<i>char</i>	attName[] name(s) of attribute(s), if you want to check combination of values of more attributes separate names of attributes with constant SEPARATOR (see test)
<i>char</i>	newValue[] new value(s)

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

**Author**

Domagoj Tuličić, updated by Nenad Makar

**Parameters**

<i>char*</i>	tableName name of table
<i>char</i>	attName[] name(s) of attribute(s), if you want to check combination of values of more attributes separate names of attributes with constant SEPARATOR (see test)
<i>char</i>	newValue[] new value(s), if you want to check combination of values of more attributes separate their values with constant SEPARATOR (see test), if some value(s) which you want to check isn't stored as char (string) convert it to char (string) using <a href="#">AK_tuple_to_string(struct list_node *tuple)</a> or with <code>sprintf</code> in a similar way it's used in that function (if value isn't part of a *tuple), to concatenate more values in newValue[] use <code>strcat(destination, source)</code> and put constant SEPARATOR between them (see test) if newValue[] should contain NULL sign pass it as " " (space)

**Returns**

EXIT\_ERROR or EXIT\_SUCCESS

### 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

<i>char*</i>	tableName name of table
<i>char</i>	attName[] name(s) of attribute(s), if you want to set UNIQUE constraint on combination of attributes separate their names with constant SEPARATOR (see test)
<i>char</i>	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

<i>type</i>	drop type
<a href="#">drop_arguments</a>	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

<i>tblName</i>	name of table or index
<i>sys_table</i>	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**

<i>tblName</i>	name of table, index view, function, trigger, sequence, user, group or constraint
<i>sys_table</i>	name of system catalog table

**Returns**

if element exist in system catalog returns 1, if not returns 0

**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

<i>type</i>	drop type
<i>drop_arguments</i>	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

<i>tblName</i>	name of table, index view, function, trigger, sequence, user, group or constraint
<i>sys_table</i>	name of system catalog table

## Returns

if element exist in system catalog returns 1, if not returns 0

**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**

<i>*function_id</i>	id of the function
<i>*arguments_list</i>	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**

<i>function↔ _id</i>	id of the function
<i>args</i>	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

<i>*name</i>	name of the function
<i>*return_type</i>	data type returned from a function - values from 0 to 13 - defined in <a href="#">constants.h</a>
<i>*arguments_list</i>	list of function arguments

## Returns

function id or EXIT\_ERROR

**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

<i>*function_id</i>	id of the function to which the argument belongs
<i>*arg_number</i>	number of the argument
<i>*arg_type</i>	data type of the argument
<i>*argname</i>	name of the argument

## Returns

function argument id or EXIT\_ERROR

**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**

<i>obj_id</i>	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**

<i>*name</i>	name of the function to be modified
<i>*arguments_list</i>	list of function arguments
<i>*new_return_type</i>	new return type

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**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

<i>*name</i>	name of the function
<i>*arguments_list</i>	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

<i>obj_id</i>	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

<i>*name</i>	name of the function to be modified
<i>*arguments_list</i>	list of arguments to be modified
<i>*new_name</i>	new name of the function

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**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**

<i>*function</i>	name of the function
<i>*arguments_list</i>	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**

<i>*function_id</i>	id of the function
<i>*arguments_list</i>	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**

<i>function↔ _id</i>	id of the function
<i>args</i>	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

<i>*name</i>	name of the function
<i>*return_type</i>	data type returned from a function - values from 0 to 13 - defined in <a href="#">constants.h</a>
<i>*arguments_list</i>	list of function arguments

## Returns

function id or EXIT\_ERROR

**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

<i>*function_id</i>	id of the function to which the argument belongs
<i>*arg_number</i>	number of the argument
<i>*arg_type</i>	data type of the argument
<i>*argname</i>	name of the argument

## Returns

function argument id or EXIT\_ERROR

**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

<i>obj_id</i>	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

<i>*name</i>	name of the function to be modified
<i>*arguments_list</i>	list of function arguments
<i>*new_return_type</i>	new return type

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

## 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

<i>*name</i>	name of the function
<i>*arguments_list</i>	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

<i>obj_id</i>	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

<i>*name</i>	name of the function to be modified
<i>*arguments_list</i>	list of arguments to be modified
<i>*new_name</i>	new name of the function

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**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**

<i>*function</i>	name of the function
<i>*arguments_list</i>	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

<i>size</i>	pointer to integer in which size of header will be saved
<i>tblName</i>	table in which rows will be inserted
<i>columns</i>	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**

<i>tableName</i>	table in which rows will be inserted
<i>columns</i>	list of columns
<i>values</i>	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

<i>size</i>	pointer to integer in which size of header will be saved
<i>tblName</i>	table in which rows will be inserted
<i>columns</i>	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

<i>tableName</i>	table in which rows will be inserted
<i>columns</i>	list of columns
<i>values</i>	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

<i>*user</i>	username of user which will be put in group
<i>*group</i>	name of group in which user will be put

#### Returns

EXIT\_SUCCESS or EXIT\_ERROR if the user is already in the group

### 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**

<i>group</i>	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**

<i>*user</i>	username for which we want check privileges
<i>*table</i>	name of table for which we want to check whether user has right on
<i>*privilege</i>	privilege for which we want to check whether user has right for

**Returns**

EXIT\_SUCCESS if user has right, EXIT\_ERROR if user has no right

**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**

<i>user</i>	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**

<i>*groupname</i>	name of group to which we want to grant privilege
<i>*table</i>	name of table on which privilege will be granted to user
<i>*right</i>	type of privilege which will be granted to user on given table

**Returns**

privilege\_id or EXIT\_ERROR if table or user aren't correct

**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**

<i>*username</i>	username of user to whom we want to grant privilege
<i>*table</i>	name of table on which privilege will be granted to user
<i>*right</i>	type of privilege which will be granted to user on given table

**Returns**

privilege\_id or EXIT\_ERROR if table or user aren't correct

**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**

<i>*name</i>	name of group to be added
<i>set_id</i>	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**

<i>*name</i>	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 Barač

**Parameters**

<i>name</i>	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 Barač, update by Lidija Lastavec

**Parameters**

<i>old_name</i>	Name of the group to be renamed
<i>new_name</i>	New name of the group

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**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**

<i>group</i>	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**

<i>user</i>	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**

<i>groupname</i>	name of group from which we want to revoke all privileges
------------------	---

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**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**

<i>username</i>	name of user from whom we want to revoke all privileges
-----------------	---

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**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**

<i>*grounamep</i>	name of group which user belongs to
<i>*table</i>	name of table on which privilege will be granted to group
<i>*right</i>	type of privilege which will be granted to group on 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**

<i>*username</i>	username of user to whom we want to grant privilege
<i>*table</i>	name of table on which privilege will be revoked from user
<i>*right</i>	type of privilege which will be revoked from user on given table

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**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**

<i>*username</i>	username of user to be added
<i>*password</i>	password of user to be added
<i>set_id</i>	obj_id of the new user

**Returns**

user\_id

**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**

<i>*username</i>	username of user whose password we are checking
<i>*password</i>	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**

<i>*username</i>	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**

<i>name</i>	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**

<i>old_name</i>	Name of the user to be renamed
<i>new_name</i>	New name of the user
<i>password</i>	Password of the user to be renamed (should be provided)

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

## 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

<i>*user</i>	username of user which will be put in group
<i>*group</i>	name of group in which user will be put

#### Returns

EXIT\_SUCCESS or EXIT\_ERROR if the user is already in the group

### 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

<i>group</i>	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

<i>*user</i>	username for which we want check privileges
<i>*table</i>	name of table for which we want to check whether user has right on
<i>*privilege</i>	privilege for which we want to check whether user has right for

#### Returns

EXIT\_SUCCESS if user has right, EXIT\_ERROR if user has no right

### 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

<i>user</i>	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**

<i>*groupname</i>	name of group to which we want to grant privilege
<i>*table</i>	name of table on which privilege will be granted to user
<i>*right</i>	type of privilege which will be granted to user on given table

**Returns**

privilege\_id or EXIT\_ERROR if table or user aren't correct

**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**

<i>*username</i>	username of user to whom we want to grant privilege
<i>*table</i>	name of table on which privilege will be granted to user
<i>*right</i>	type of privilege which will be granted to user on given table

**Returns**

privilege\_id or EXIT\_ERROR if table or user aren't correct

**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 Barač

**Parameters**

<i>*name</i>	name of group to be added
<i>set_id</i>	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**

<i>*name</i>	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 Barač

**Parameters**

<i>name</i>	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 Barač, update by Lidija Lastavec

**Parameters**

<i>old_name</i>	Name of the group to be renamed
<i>new_name</i>	New name of the group

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**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

<i>group</i>	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

<i>user</i>	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**

<i>groupname</i>	name of group from which we want to revoke all privileges
------------------	---

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**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**

<i>username</i>	name of user from whom we want to revoke all privileges
-----------------	---

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**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**

<i>*grounamep</i>	name of group which user belongs to
<i>*table</i>	name of table on which privilege will be granted to group
<i>*right</i>	type of privilege which will be granted to group on 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**

<i>*grounamep</i>	name of group which user belongs to
<i>*table</i>	name of table on which privilege will be granted to group
<i>*right</i>	type of privilege which will be granted to group on 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**

<i>*username</i>	username of user to whom we want to grant privilege
<i>*table</i>	name of table on which privilege will be revoked from user
<i>*right</i>	type of privilege which will be revoked from user on given table



**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

NOTICE: Test 12 isn't currently revoking a privilege since the obj\_id in the AK\_group\_right table is passing the value of 127. Once the issue #87 on GitHub concerning the data type is solved, the test should be working as expected.

**Author**

Kristina Takač, updated by Mario Peroković - added comparing by table id, and use of user\_id in AK\_user\_right

**Parameters**

<i>*username</i>	username of user to whom we want to grant privilege
<i>*table</i>	name of table on which privilege will be revoked from user
<i>*right</i>	type of privilege which will be revoked from user on given table

**Returns**

EXIT\_SUCCESS if privilege is revoked, EXIT\_ERROR if it isn't

**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**

<i>*username</i>	username of user to be added
<i>*password</i>	password of user to be added
<i>set_id</i>	obj_id of the new user

**Returns**

user\_id

**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**

<i>*username</i>	username of user whose password we are checking
<i>*password</i>	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**

<i>*username</i>	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

<i>old_name</i>	Name of the user to be renamed
<i>new_name</i>	New name of the user
<i>password</i>	Password of the user to be renamed (should be provided)

#### Returns

EXIT\_SUCCESS or EXIT\_ERROR

## 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

<i>srcTable</i>	- original table that is used for selection
<i>destTable</i>	- table that contains the result
<i>condition</i>	- condition for selection
<i>attributes</i>	- attributes to be selected
<i>ordering</i>	- attributes 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

<i>srcTable</i>	- original table that is used for selection
<i>destTable</i>	- table that contains the result
<i>condition</i>	- condition for selection
<i>attributes</i>	- attributes to be selected
<i>ordering</i>	- attributes 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

<i>*name</i>	name of the trigger
<i>*event</i>	event that calls the trigger - this should perhaps be an integer with defined constants...
<i>*condition</i>	AK_list list of conditions in postfix
<i>*table</i>	name of the table trigger is hooked on
<i>*function</i>	function that is being called by the trigger

#### Returns

trigger id or EXIT\_ERROR

### 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**

<i>*name</i>	name of the trigger (or NULL if using obj_id)
<i>*event</i>	event of the trigger (or NULL if it isn't changing)
<i>*condition</i>	list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed)
<i>*table</i>	name of the connected table (or NULL id using obj_id)
<i>*function</i>	name of the connected function (or NULL if it isn't changing)

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR @IMPORTANT: \*AK\_dbg\_messg.. need to be fixed then we could un-comment 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**

<i>trigger</i>	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

<i>*name</i>	name of the trigger
<i>*table</i>	name of the table on which the trigger is hooked

## Returns

obj\_id of the trigger or EXIT\_ERROR

**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

<i>*name</i>	name of the trigger
<i>*table</i>	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

<i>obj_id</i>	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

<i>old_name</i>	Name of the trigger to be renamed
<i>new_name</i>	New name of the trigger

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

**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

<i>trigger</i>	obj_id of the trigger in question
<i>*condition</i>	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

<i>*name</i>	name of the trigger
<i>*event</i>	event that calls the trigger - this should perhaps be an integer with defined constants...
<i>*condition</i>	AK_list list of conditions in postfix
<i>*table</i>	name of the table trigger is hooked on
<i>*function</i>	function that is being called by the trigger

#### Returns

trigger id or EXIT\_ERROR

#### 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

<i>*name</i>	name of the trigger (or NULL if using obj_id)
<i>*event</i>	event of the trigger (or NULL if it isn't changing)
<i>*condition</i>	list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed)
<i>*table</i>	name of the connected table (or NULL id using obj_id)
<i>*function</i>	name of the connected function (or NULL if it isn't changing)

## Returns

EXIT\_SUCCESS or EXIT\_ERROR

Function that edits information about the trigger in system table.

## Author

Unknown

## Parameters

<i>*name</i>	name of the trigger (or NULL if using obj_id)
<i>*event</i>	event of the trigger (or NULL if it isn't changing)
<i>*condition</i>	list of conditions for trigger (or NULL if it isn't changing; empty list if all conditions are to be removed)
<i>*table</i>	name of the connected table (or NULL id using obj_id)
<i>*function</i>	name of the connected function (or NULL if it isn't changing)

## Returns

EXIT\_SUCCESS or EXIT\_ERROR @IMPORTANT: \*AK\_dbg\_messg.. need to be fixed then we could un-comment 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

<i>trigger</i>	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**

<i>*name</i>	name of the trigger
<i>*table</i>	name of the table on which the trigger is hooked

**Returns**

obj\_id of the trigger or EXIT\_ERROR

**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**

<i>*name</i>	name of the trigger
<i>*table</i>	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**

<i>obj_id</i>	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**

<i>old_name</i>	Name of the trigger to be renamed
<i>new_name</i>	New name of the trigger

**Returns**

EXIT\_SUCCESS or EXIT\_ERROR

**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**

<i>trigger</i>	obj_id of the trigger in question
<i>*condition</i>	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 Kistic

**Parameters**

<i>name</i>	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**

<i>name</i>	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

<i>name</i>	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

<i>rel_exp</i>	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**

<i>name</i>	name of the view
<i>query</i>	query of the view
<i>rel_exp</i>	relation expression of the view
<i>set_id</i>	id of view

**Returns**

Id of the newly inserted view

**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**

<i>name</i>	of the query
<i>query</i>	new query of the view
<i>rel_exp</i>	relation expression of the view

**Returns**

error or success

**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**

<i>name</i>	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**

<i>obj_id</i>	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

<i>name</i>	name of the view
<i>new_name</i>	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 Kisić

#### Parameters

<i>name</i>	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

<i>name</i>	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

<i>name</i>	name og the view
<i>query</i>	query of the view
<i>rel_exp</i>	relation expression of the view
<i>set_id</i>	id of view

#### Returns

Id of the newly inserted view

### 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

<i>name</i>	of the query
<i>query</i>	new query of the view
<i>rel_exp</i>	relation expression of the view



**Returns**

error or success

**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**

<i>name</i>	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**

<i>name</i>	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**

<i>name</i>	name of the view
<i>new_name</i>	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\_transaction type.*

- void [handle\\_transaction\\_notify](#) ([AK\\_observer\\_lock](#) \*[observer\\_lock](#))  
Function for handling [AK\\_observable\\_transaction](#) notify. Function is associated to some observer instance.
- void [AK\\_on\\_observable\\_notify](#) (void \*[observer](#), void \*[observable](#), [AK\\_ObservableType\\_Enum](#) type)  
Function for handling notify from some observable type.
- void [AK\\_on\\_transaction\\_end](#) (pthread\_t [transaction\\_thread](#))  
Function for handling event when some transaction is finished.
- void [AK\\_on\\_all\\_transactions\\_end](#) ()  
Function for handling event when all transactions are finished.
- void [AK\\_on\\_lock\\_release](#) ()  
Function for handling event when one of lock is released.
- void [AK\\_handle\\_observable\\_transaction\\_action](#) ([NoticeType](#) \*[noticeType](#))  
Function for handling action which is called from [observable\\_transaction](#) type.
- void [AK\\_lock\\_released](#) ()  
Function which is called when 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

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.

#### Returns

OK or NOT\_OK based on the success of the function.

```
#####\n# Lock Granted after wait\n#-----\n# Lock ID:lu TYPE:i \n#-----\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\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

<i>blockAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.

**Returns**

pointer to the newly created doubly linked element.

**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**

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.

**Returns**

pointer to the newly created Locks doubly linked element.

**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**

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.

**Returns**

pointer to the newly created Locks doubly linked element.

**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**

<i>transaction_data</i>	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**

<i>blockAddress</i>	integer representation of memory address.
---------------------	---



**Returns**

integer OK or NOT\_OK based on success of finding the specific element in the list.

**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**

<i>blockAddress</i>	integer representation of memory address.
<i>id</i>	integer representation of transaction id.

**Returns**

int OK or NOT\_OK based on success of finding the specific element in the list.

**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**

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	length of commandArray
<i>transactionId</i>	associated with the transaction

**Returns**

ABORT or COMMIT based on the success of the function.

**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**

<i>data</i>	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**

<i>addressList</i>	pointer to the linked list where the addresses are stored.
<i>tblName</i>	table name used in the transaction

**Returns**

OK or NOT\_OK based on the success of the function.

#### 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

<i>noticeType</i>	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**

<i>lockHolder</i>	pointer to the hash list entry that is entitled to the specific memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.
<i>lock</i>	pointer to the lock element that is being tested.

**Returns**

int PASS\_LOCK\_QUEUE or WAIT\_FOR\_UNLOCK based on the rules described inside the function.

**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**

<i>blockMemoryAddress</i>	integer representation of memory address, the hash value is calculated from this parameter.
---------------------------	---

**Returns**

integer containing the hash value of the passed memory address

**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

<i>observer</i>	<a href="#">Observer</a> type
<i>observable</i>	<a href="#">Observable</a> type
<i>type</i>	Type of observable who sent some notice

**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

<i>transaction_thread</i>	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

<i>adresses</i>	linked list of memory addresses locked by the transaction.
<i>transaction↔ Id</i>	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

<i>transaction_thread</i>	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

<i>blockAddress</i>	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**

<i>blockAddress</i>	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**

<i>memoryAddress</i>	integer representation of memory address.
<i>id</i>	integer representation of transaction id.

**Returns**

NULL pointer if the element is not found otherwise it returns a pointer to the found element

**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

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	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

<i>observable_transaction</i>	Observable type instance
<i>observer</i>	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\_transaction type.

**Author**

Ivan Pusic

**Parameters**

<a href="#"><i>observable_transaction</i></a>	<a href="#">Observable</a> type instance
<a href="#"><i>observer</i></a>	<a href="#">Observer</a> 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**

<a href="#"><i>observer_lock</i></a>	<a href="#">Observer</a> 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\\_transaction](#) type.*
- [AK\\_observable\\_transaction](#) \* [AK\\_init\\_observable\\_transaction](#) ()

*Function for initialization of [AK\\_observable\\_transaction](#) type.*
- [AK\\_observer\\_lock](#) \* [AK\\_init\\_observer\\_lock](#) ()

*Function for initialization of [AK\\_observer\\_lock](#) type.*

### 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

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.

## Returns

OK or NOT\_OK based on the success of the function.

## Author

Frane Jakelić updated by Ivan Pusic

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

## Parameters

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.

## Returns

OK or NOT\_OK based on the success of the function.

```
#####\n# Lock Granted after wait\n#-----#\n# Lock ID:lu TYPE:i #\n#-----\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#\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

<i>blockAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.

## Returns

pointer to the newly created doubly linked element.

**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

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.

## Returns

pointer to the newly created Locks doubly linked element.

**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

<i>memoryAddress</i>	integer representation of memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transactionId</i>	integer representation of transaction id.

##### Returns

pointer to the newly created Locks doubly linked element.

#### 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

<i>transaction_data</i>	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

<i>blockAddress</i>	integer representation of memory address.
---------------------	---

##### Returns

integer OK or NOT\_OK based on success of finding the specific element in the list.

#### 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

<i>blockAddress</i>	integer representation of memory address.
<i>id</i>	integer representation of transaction id.

##### Returns

int OK or NOT\_OK based on success of finding the specific element in the list.

#### 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**

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	length of commandArray
<i>transactionId</i>	associated with the transaction

**Returns**

ABORT or COMMIT based on the success of the function.

**Author**

Frane Jakelić updated by Ivan Pusic

**Todo** Check multithreading, check if it's working correctly

**Parameters**

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	length of commandArray
<i>transactionId</i>	associated with the transaction

**Returns**

ABORT or COMMIT based on the success of the function.

**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

<i>data</i>	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

<i>addressList</i>	pointer to the linked list where the addresses are stored.
<i>tblName</i>	table name used in the transaction

## Returns

OK or NOT\_OK based on the success of the function.

**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

<i>noticeType</i>	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**

<i>lockHolder</i>	pointer to the hash list entry that is entitled to the specific memory address.
<i>type</i>	of lock issued to the provided memory address.
<i>transaction↔ Id</i>	integer representation of transaction id.
<i>lock</i>	pointer to the lock element that is being tested.

**Returns**

int PASS\_LOCK\_QUEUE or WAIT\_FOR\_UNLOCK based on the rules described inside the function.

**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**

<i>blockMemoryAddress</i>	integer representation of memory address, the hash value is calculated from this parameter.
---------------------------	---



**Returns**

integer containing the hash value of the passed memory address

**Author**

Frane Jakelić

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

**Parameters**

<i>blockMemoryAddress</i>	integer representation of memory address, the hash value is calculated from this parameter.
---------------------------	---

**Returns**

integer containing the hash value of the passed memory address

**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_ObservableTypeEnum type )
```

Function for handling notify from some observable type.

**Author**

Ivan Pusic

## Parameters

<i>observer</i>	<a href="#">Observer</a> type
<i>observable</i>	<a href="#">Observable</a> type
<i>type</i>	Type of observable who sent some notice

**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

<i>transaction_thread</i>	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

<i>adresses</i>	linked list of memory addresses locked by the transaction.
<i>transaction↔ Id</i>	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

<i>transaction_thread</i>	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

<i>blockAddress</i>	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**

<i>blockAddress</i>	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**

<i>memoryAddress</i>	integer representation of memory address.
<i>id</i>	integer representation of transaction id.

**Returns**

NULL pointer if the element is not found otherwise it returns a pointer to the found element

**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

<i>commandArray</i>	array filled with commands that need to be secured using transactions
<i>lengthOfArray</i>	length of <i>commandArray</i>

#### 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

<i>observable_transaction</i>	<i>Observable</i> type instance
<i>observer</i>	<i>Observer</i> 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\_transaction* type.

**Author**

Ivan Pusic

**Parameters**

<a href="#"><i>observable_transaction</i></a>	<a href="#">Observable</a> type instance
<i>observer</i>	<a href="#">Observer</a> instance

**Returns**

Exit status (OK or NOT\_OK)

**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**

<a href="#"><i>observer_lock</i></a>	<a href="#">Observer</a> 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](#)
- [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\_attribute\_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\_attributes\_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](#)
- ASCIIINESZ
  - 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

## BITSLOTS

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](#)
  - BITSLLOT, [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.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\_getseckey, [149](#)
- iniparser\_getsecname, [150](#)
- iniparser\_getseckeys, [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\_freelock, [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\_getseckey, [159](#)
  - iniparser\_getsecname, [159](#)
  - iniparser\_getseckeys, [160](#)
  - iniparser\_getstring, [160](#)
  - iniparser\_load, [160](#)
  - iniparser\_set, [161](#)
  - iniparser\_unset, [161](#)
- iniparser\_AK\_freelock
  - 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\_getseckey
  - iniparser.c, [149](#)
  - iniparser.h, [159](#)
- iniparser\_getsecname
  - iniparser.c, [150](#)
  - iniparser.h, [159](#)
- iniparser\_getseckeys
  - 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](#)
- nextSucesor
  - Sucesor, [72](#)
  - Vertex, [82](#)
- nextThread
  - threadContainer, [74](#)
- nextVertex
  - Vertex, [82](#)
- nnull.c
  - AK\_check\_constraint\_not\_null, [575](#)
  - AK\_delete\_constraint\_not\_null, [576](#)

- AK\_nullable\_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\_nullable\_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/nnnull.c, [575](#)
- sql/cs/nnnull.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\_attribute\_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\_attribute\_types, [214](#)
  - BLACK, [209](#)
  - BLUE, [209](#)
  - BOLDBLACK, [209](#)
  - BOLDBLUE, [210](#)
  - BOLD CYAN, [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](#)
- testSucceeded
  - 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](#)