

Jazzyk
1.20

Generated by Doxygen 1.5.6

Sat Jan 10 22:01:55 2009

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Class Index	3
2.1	Class Hierarchy	3
3	Class Index	7
3.1	Class List	7
4	File Index	9
4.1	File List	9
5	Namespace Documentation	11
5.1	jazzyk Namespace Reference	11
5.1.1	Detailed Description	12
5.1.2	Typedef Documentation	12
5.1.2.1	TJzModuleApiRegisterRoutine	12
5.1.2.2	TJzModuleApiVersionRoutine	12
5.1.2.3	TKRVarSubstitution	12
5.1.3	Enumeration Type Documentation	12
5.1.3.1	EKRModuleError	12
5.2	jazzyk_private Namespace Reference	13
5.2.1	Detailed Description	13
5.3	yy Namespace Reference	14
5.3.1	Detailed Description	14
6	Class Documentation	15
6.1	jazzyk::AbstractKRModule Class Reference	15
6.1.1	Detailed Description	15
6.1.2	Member Function Documentation	15

6.1.2.1	initialize	15
6.1.2.2	finalize	16
6.1.2.3	cycle	16
6.2	jazyk::AbstractKRModuleInterface< TConcreteKRModule > Class Template Reference	17
6.2.1	Detailed Description	19
6.2.2	Member Typedef Documentation	19
6.2.2.1	TPtrQueryHandler	19
6.2.2.2	TPtrUpdateHandler	19
6.2.2.3	TQueryRegister	19
6.2.2.4	TUpdateRegister	20
6.2.3	Constructor & Destructor Documentation	20
6.2.3.1	AbstractKRModuleInterface	20
6.2.3.2	~AbstractKRModuleInterface	20
6.2.4	Member Function Documentation	20
6.2.4.1	initialize	20
6.2.4.2	finalize	20
6.2.4.3	cycle	21
6.2.4.4	query	21
6.2.4.5	update	21
6.2.4.6	getQueryInterface	22
6.2.4.7	getUpdateInterface	22
6.2.4.8	free_resource	22
6.2.4.9	registerQuery	23
6.2.4.10	registerUpdate	23
6.2.4.11	destroy	23
6.2.4.12	operator delete	23
6.2.4.13	map2raw	24
6.2.4.14	raw2map	24
6.2.5	Member Data Documentation	24
6.2.5.1	m_concreteKRModule	24
6.2.5.2	m_QueryOperations	24
6.2.5.3	m_UpdateOperations	25
6.3	base_visitor Class Reference	26
6.3.1	Detailed Description	26
6.3.2	Constructor & Destructor Documentation	26
6.3.2.1	~base_visitor	26

6.4	CAndQueryNode Class Reference	27
6.4.1	Detailed Description	28
6.4.2	Constructor & Destructor Documentation	28
6.4.2.1	CAndQueryNode	28
6.4.2.2	~CAndQueryNode	28
6.4.2.3	CAndQueryNode	28
6.4.3	Member Function Documentation	28
6.4.3.1	getLeftNode	28
6.4.3.2	getRightNode	28
6.4.3.3	operator=	29
6.4.4	Member Data Documentation	29
6.4.4.1	m_pLeftNode	29
6.4.4.2	m_pRightNode	29
6.5	CCodeBlock Class Reference	30
6.5.1	Detailed Description	30
6.5.2	Constructor & Destructor Documentation	31
6.5.2.1	CCodeBlock	31
6.5.2.2	CCodeBlock	31
6.5.2.3	~CCodeBlock	31
6.5.2.4	CCodeBlock	31
6.5.3	Member Function Documentation	31
6.5.3.1	operator const char *	31
6.5.3.2	append	31
6.5.3.3	operator+=	31
6.5.3.4	operator=	32
6.5.4	Member Data Documentation	32
6.5.4.1	m_szCode	32
6.6	CCompiler Class Reference	33
6.6.1	Detailed Description	35
6.6.2	Member Enumeration Documentation	35
6.6.2.1	"@0	35
6.6.3	Constructor & Destructor Documentation	35
6.6.3.1	CCompiler	35
6.6.3.2	~CCompiler	36
6.6.3.3	CCompiler	36
6.6.4	Member Function Documentation	36

6.6.4.1	setIncludePaths	36
6.6.4.2	preprocess	36
6.6.4.3	preprocess	36
6.6.4.4	parse	37
6.6.4.5	compile	37
6.6.4.6	getProgram	37
6.6.4.7	PreAction	37
6.6.4.8	PreAction	38
6.6.4.9	preprocess	38
6.6.4.10	runChildProcess	38
6.6.4.11	runParentProcess	38
6.6.4.12	readPipe	39
6.6.4.13	fillIncludePaths	39
6.6.4.14	operator=	39
6.6.5	Member Data Documentation	39
6.6.5.1	m_pProgram	39
6.6.5.2	m_vsIncludePaths	39
6.7	CCompiler::SMPInvocationSettings Struct Reference	41
6.7.1	Detailed Description	41
6.7.2	Constructor & Destructor Documentation	41
6.7.2.1	SMPInvocationSettings	41
6.7.3	Member Function Documentation	42
6.7.3.1	configure	42
6.7.4	Member Data Documentation	42
6.7.4.1	child_stdin	42
6.7.4.2	child_stdout	42
6.7.4.3	child_stderr	42
6.7.4.4	child_pid	42
6.7.4.5	cmd	42
6.7.4.6	argv	43
6.8	CElseRule Class Reference	44
6.8.1	Detailed Description	45
6.8.2	Constructor & Destructor Documentation	45
6.8.2.1	CElseRule	45
6.8.2.2	~CElseRule	45
6.8.2.3	CElseRule	45

6.8.3	Member Function Documentation	45
6.8.3.1	getElseTransformer	45
6.8.3.2	operator=	45
6.8.4	Member Data Documentation	45
6.8.4.1	m_pElseTransformer	45
6.9	CError Class Reference	47
6.9.1	Detailed Description	48
6.9.2	Member Typedef Documentation	48
6.9.2.1	TSrcLocation	48
6.9.3	Member Enumeration Documentation	49
6.9.3.1	EErrorType	49
6.9.4	Constructor & Destructor Documentation	49
6.9.4.1	CError	49
6.9.4.2	CError	49
6.9.4.3	~CError	49
6.9.5	Member Function Documentation	50
6.9.5.1	report	50
6.9.5.2	resolveErrorType	50
6.9.5.3	what	50
6.9.5.4	operator=	50
6.9.6	Member Data Documentation	50
6.9.6.1	m_location	50
6.9.6.2	m_errorType	50
6.9.6.3	m_szReason	50
6.9.6.4	m_dictErrorMsgs	51
6.9.6.5	m_szErrMsgText	51
6.10	CExitUpdate Class Reference	52
6.10.1	Detailed Description	52
6.10.2	Constructor & Destructor Documentation	52
6.10.2.1	CExitUpdate	52
6.10.2.2	~CExitUpdate	53
6.10.2.3	CExitUpdate	53
6.10.3	Member Function Documentation	53
6.10.3.1	operator=	53
6.11	CExplicitQuery Class Reference	54
6.11.1	Detailed Description	55

6.11.2	Constructor & Destructor Documentation	55
6.11.2.1	CExplicitQuery	55
6.11.2.2	~CExplicitQuery	55
6.11.2.3	CExplicitQuery	55
6.11.3	Member Function Documentation	55
6.11.3.1	getModuleID	55
6.11.3.2	getOperation	56
6.11.3.3	getFreeVariables	56
6.11.3.4	getCodeBlock	56
6.11.3.5	operator=	56
6.11.4	Member Data Documentation	56
6.11.4.1	m_pModuleID	56
6.11.4.2	m_pOperation	56
6.11.4.3	m_pVariables	57
6.11.4.4	m_pCodeBlock	57
6.12	CExplicitUpdate Class Reference	58
6.12.1	Detailed Description	59
6.12.2	Constructor & Destructor Documentation	59
6.12.2.1	CExplicitUpdate	59
6.12.2.2	~CExplicitUpdate	59
6.12.2.3	CExplicitUpdate	59
6.12.3	Member Function Documentation	59
6.12.3.1	getModuleID	59
6.12.3.2	getOperation	60
6.12.3.3	getUsedVariables	60
6.12.3.4	getCodeBlock	60
6.12.3.5	operator=	60
6.12.4	Member Data Documentation	60
6.12.4.1	m_pModuleID	60
6.12.4.2	m_pOperation	60
6.12.4.3	m_pVariables	61
6.12.4.4	m_pCodeBlock	61
6.13	CExpression Class Reference	62
6.13.1	Detailed Description	63
6.13.2	Constructor & Destructor Documentation	63
6.13.2.1	CExpression	63

6.13.2.2	~CExpression	63
6.13.2.3	CExpression	63
6.13.3	Member Function Documentation	63
6.13.3.1	getSrcPosition	63
6.13.3.2	operator=	64
6.13.4	Member Data Documentation	64
6.13.4.1	m_srcLocation	64
6.14	CFalseQuery Class Reference	65
6.14.1	Detailed Description	65
6.14.2	Constructor & Destructor Documentation	65
6.14.2.1	CFalseQuery	65
6.14.2.2	~CFalseQuery	66
6.14.2.3	CFalseQuery	66
6.14.3	Member Function Documentation	66
6.14.3.1	operator=	66
6.15	CIdentifier Class Reference	67
6.15.1	Detailed Description	67
6.15.2	Constructor & Destructor Documentation	68
6.15.2.1	CIdentifier	68
6.15.2.2	CIdentifier	68
6.15.2.3	~CIdentifier	68
6.15.2.4	CIdentifier	68
6.15.3	Member Function Documentation	68
6.15.3.1	operator const std::string &	68
6.15.3.2	operator const char *	68
6.15.3.3	get	68
6.15.3.4	operator=	68
6.15.4	Member Data Documentation	69
6.15.4.1	m_szIdentifier	69
6.16	CInterpreter Class Reference	70
6.16.1	Detailed Description	72
6.16.2	Constructor & Destructor Documentation	72
6.16.2.1	CInterpreter	72
6.16.2.2	~CInterpreter	72
6.16.2.3	CInterpreter	72
6.16.3	Member Function Documentation	72

6.16.3.1	run	72
6.16.3.2	configure	73
6.16.3.3	Visit	73
6.16.3.4	PostAction	73
6.16.3.5	PostAction	73
6.16.3.6	PostAction	73
6.16.3.7	PostAction	74
6.16.3.8	PostAction	74
6.16.3.9	doCycle	74
6.16.3.10	operator=	74
6.16.4	Member Data Documentation	74
6.16.4.1	m_settings	74
6.16.4.2	m_pProgram	74
6.16.4.3	m_pfRandomGenerator	75
6.17	CInterpreter::SSettings Struct Reference	76
6.17.1	Detailed Description	76
6.17.2	Member Enumeration Documentation	76
6.17.2.1	"@1	76
6.17.3	Constructor & Destructor Documentation	77
6.17.3.1	SSettings	77
6.17.4	Member Data Documentation	77
6.17.4.1	nCycles	77
6.17.4.2	eTransformerSetPolicy	77
6.18	CIpcPtrDeleter< T > Class Template Reference	78
6.18.1	Detailed Description	78
6.18.2	Member Typedef Documentation	79
6.18.2.1	TSegmentManager	79
6.18.2.2	TStoredPtr	79
6.18.3	Constructor & Destructor Documentation	79
6.18.3.1	CIpcPtrDeleter	79
6.18.4	Member Function Documentation	79
6.18.4.1	operator()	79
6.18.5	Member Data Documentation	79
6.18.5.1	m_pSegmentManager	79
6.19	CJazzykApp Class Reference	80
6.19.1	Detailed Description	81

6.19.2	Constructor & Destructor Documentation	81
6.19.2.1	CJazzykApp	81
6.19.2.2	~CJazzykApp	81
6.19.2.3	CJazzykApp	81
6.19.3	Member Function Documentation	81
6.19.3.1	Instance	81
6.19.3.2	run	82
6.19.3.3	error	82
6.19.3.4	processCmdLine	82
6.19.3.5	version	82
6.19.3.6	license	82
6.19.3.7	read	83
6.19.3.8	operator=	83
6.19.4	Member Data Documentation	83
6.19.4.1	m_settings	83
6.20	CJazzykApp::SSettings Struct Reference	84
6.20.1	Detailed Description	85
6.20.2	Constructor & Destructor Documentation	85
6.20.2.1	SSettings	85
6.20.3	Member Data Documentation	85
6.20.3.1	files	85
6.20.3.2	includePaths	85
6.20.3.3	modulePaths	85
6.20.3.4	bReadStdIn	85
6.20.3.5	bPrettyPrint	85
6.20.3.6	bPreprocessOnly	86
6.20.3.7	bCompileOnly	86
6.20.3.8	bBypassMP	86
6.20.3.9	nCycles	86
6.20.3.10	bSetFirst	86
6.20.3.11	nMemSize	86
6.20.3.12	bCheckQuery	86
6.21	CJazzykLexer Class Reference	88
6.21.1	Detailed Description	88
6.21.2	Constructor & Destructor Documentation	89
6.21.2.1	CJazzykLexer	89

6.21.2.2	~CJazzykLexer	89
6.21.2.3	CJazzykLexer	89
6.21.3	Member Function Documentation	89
6.21.3.1	yylex	89
6.21.3.2	operator=	89
6.21.4	Member Data Documentation	89
6.21.4.1	m_pyylval	89
6.21.4.2	m_pyylloc	89
6.22	CLocationTracker Class Reference	91
6.22.1	Detailed Description	91
6.22.2	Constructor & Destructor Documentation	92
6.22.2.1	CLocationTracker	92
6.22.2.2	~CLocationTracker	92
6.22.2.3	CLocationTracker	92
6.22.3	Member Function Documentation	92
6.22.3.1	Instance	92
6.22.3.2	set	92
6.22.3.3	get	92
6.22.3.4	operator=	92
6.22.4	Member Data Documentation	93
6.22.4.1	m_srcLocation	93
6.23	CModuleDeclaration Class Reference	94
6.23.1	Detailed Description	95
6.23.2	Constructor & Destructor Documentation	95
6.23.2.1	CModuleDeclaration	95
6.23.2.2	~CModuleDeclaration	95
6.23.2.3	CModuleDeclaration	96
6.23.3	Member Function Documentation	96
6.23.3.1	getIdentifier	96
6.23.3.2	getType	96
6.23.3.3	addNotification	96
6.23.3.4	addQueryOperation	96
6.23.3.5	addUpdateOperation	96
6.23.3.6	isValidQueryOperation	97
6.23.3.7	isValidUpdateOperation	97
6.23.3.8	getNotificationCode	97

6.23.3.9	getSrcPosition	97
6.23.3.10	operator=	97
6.23.4	Member Data Documentation	97
6.23.4.1	m_pIdentifier	97
6.23.4.2	m_pType	98
6.23.4.3	m_pInitCode	98
6.23.4.4	m_pFinalizeCode	98
6.23.4.5	m_pCycleCode	98
6.23.4.6	m_srcLocation	98
6.23.4.7	m_queryOperations	98
6.23.4.8	m_updateOperations	98
6.24	CModuleManager Class Reference	100
6.24.1	Detailed Description	102
6.24.2	Constructor & Destructor Documentation	102
6.24.2.1	CModuleManager	102
6.24.2.2	~CModuleManager	102
6.24.2.3	CModuleManager	102
6.24.3	Member Function Documentation	102
6.24.3.1	Instance	102
6.24.3.2	initialize	102
6.24.3.3	finalize	103
6.24.3.4	load	103
6.24.3.5	unload	103
6.24.3.6	query	103
6.24.3.7	update	104
6.24.3.8	cycle	104
6.24.3.9	load	104
6.24.3.10	unload	104
6.24.3.11	writeRequest	105
6.24.3.12	extractResponse	105
6.24.3.13	communicate	105
6.24.3.14	killModule	106
6.24.3.15	filterVariables	106
6.24.3.16	operator=	106
6.24.4	Member Data Documentation	106
6.24.4.1	m_mapPIDSubprocesses	106

6.24.4.2	m_vecSearchPaths	106
6.24.4.3	m_bCheckQuery	107
6.25	CModuleNotification Class Reference	108
6.25.1	Detailed Description	109
6.25.2	Member Enumeration Documentation	109
6.25.2.1	ENotificationType	109
6.25.3	Constructor & Destructor Documentation	109
6.25.3.1	CModuleNotification	109
6.25.3.2	~CModuleNotification	109
6.25.3.3	CModuleNotification	110
6.25.4	Member Function Documentation	110
6.25.4.1	getType	110
6.25.4.2	getCode	110
6.25.4.3	getIdentifer	110
6.25.4.4	getSrcPosition	110
6.25.4.5	operator=	110
6.25.5	Member Data Documentation	110
6.25.5.1	m_eType	110
6.25.5.2	m_pIdentifier	111
6.25.5.3	m_pCode	111
6.25.5.4	m_srcLocation	111
6.26	CNopUpdate Class Reference	112
6.26.1	Detailed Description	112
6.26.2	Constructor & Destructor Documentation	112
6.26.2.1	CNopUpdate	112
6.26.2.2	~CNopUpdate	113
6.26.2.3	CNopUpdate	113
6.26.3	Member Function Documentation	113
6.26.3.1	operator=	113
6.27	CNotQueryNode Class Reference	114
6.27.1	Detailed Description	114
6.27.2	Constructor & Destructor Documentation	115
6.27.2.1	CNotQueryNode	115
6.27.2.2	~CNotQueryNode	115
6.27.2.3	CNotQueryNode	115
6.27.3	Member Function Documentation	115

6.27.3.1	getNode	115
6.27.3.2	operator=	115
6.27.4	Member Data Documentation	115
6.27.4.1	m_pNegatedNode	115
6.28	COrQueryNode Class Reference	116
6.28.1	Detailed Description	117
6.28.2	Constructor & Destructor Documentation	117
6.28.2.1	COrQueryNode	117
6.28.2.2	~COrQueryNode	117
6.28.2.3	COrQueryNode	117
6.28.3	Member Function Documentation	117
6.28.3.1	getLeftNode	117
6.28.3.2	getRightNode	117
6.28.3.3	operator=	118
6.28.4	Member Data Documentation	118
6.28.4.1	m_pLeftNode	118
6.28.4.2	m_pRightNode	118
6.29	CPrettyPrinter Class Reference	119
6.29.1	Detailed Description	120
6.29.2	Constructor & Destructor Documentation	121
6.29.2.1	CPrettyPrinter	121
6.29.2.2	~CPrettyPrinter	121
6.29.2.3	CPrettyPrinter	121
6.29.3	Member Function Documentation	121
6.29.3.1	dump	121
6.29.3.2	PreAction	121
6.29.3.3	PreAction	121
6.29.3.4	PreAction	121
6.29.3.5	PreAction	122
6.29.3.6	PreAction	122
6.29.3.7	PreAction	122
6.29.3.8	PreAction	122
6.29.3.9	PreAction	122
6.29.3.10	PreAction	122
6.29.3.11	PreAction	123
6.29.3.12	PreAction	123

6.29.3.13	PreAction	123
6.29.3.14	DefaultPostAction	123
6.29.3.15	indent	123
6.29.3.16	operator=	123
6.29.4	Member Data Documentation	123
6.29.4.1	m_pProgram	123
6.29.4.2	m_outStream	124
6.29.4.3	m_level	124
6.30	CProgram Class Reference	125
6.30.1	Detailed Description	126
6.30.2	Constructor & Destructor Documentation	126
6.30.2.1	CProgram	126
6.30.2.2	~CProgram	126
6.30.2.3	CProgram	126
6.30.3	Member Function Documentation	127
6.30.3.1	add	127
6.30.3.2	add	127
6.30.3.3	add	127
6.30.3.4	isModuleDeclared	127
6.30.3.5	getModuleDeclaration	127
6.30.3.6	getTransformer	127
6.30.3.7	loadModules	128
6.30.3.8	unloadModules	128
6.30.3.9	cycle	128
6.30.3.10	retrieveInterfaces	128
6.30.3.11	operator=	128
6.30.4	Member Data Documentation	128
6.30.4.1	m_moduleDeclarations	128
6.30.4.2	m_pTransformerCollection	129
6.31	CQueryContext Class Reference	130
6.31.1	Detailed Description	131
6.31.2	Member Enumeration Documentation	131
6.31.2.1	EQueryReturnValue	131
6.31.3	Constructor & Destructor Documentation	132
6.31.3.1	CQueryContext	132
6.31.3.2	~CQueryContext	132

6.31.3.3	CQueryContext	132
6.31.4	Member Function Documentation	132
6.31.4.1	operator=	132
6.31.4.2	getResult	133
6.31.4.3	setResult	133
6.31.4.4	resetResult	133
6.31.4.5	isTrue	133
6.31.4.6	isTrueW	133
6.31.4.7	isFalse	133
6.31.4.8	isFalseW	134
6.31.4.9	operator &=	134
6.31.4.10	operator " ="	134
6.31.4.11	negate	134
6.31.5	Member Data Documentation	134
6.31.5.1	m_result	134
6.32	CQueryNode Class Reference	136
6.32.1	Detailed Description	136
6.32.2	Constructor & Destructor Documentation	136
6.32.2.1	CQueryNode	136
6.32.2.2	~CQueryNode	137
6.32.2.3	CQueryNode	137
6.32.3	Member Function Documentation	137
6.32.3.1	operator=	137
6.33	CRule Class Reference	138
6.33.1	Detailed Description	139
6.33.2	Constructor & Destructor Documentation	139
6.33.2.1	CRule	139
6.33.2.2	~CRule	139
6.33.2.3	CRule	139
6.33.3	Member Function Documentation	139
6.33.3.1	getQuery	139
6.33.3.2	getThenTransformer	139
6.33.3.3	operator=	139
6.33.4	Member Data Documentation	140
6.33.4.1	m_pQueryList	140
6.33.4.2	m_pThenTransformer	140

6.34	CShmClientMgr Class Reference	141
6.34.1	Detailed Description	142
6.34.2	Constructor & Destructor Documentation	142
6.34.2.1	CShmClientMgr	142
6.34.2.2	~CShmClientMgr	142
6.34.2.3	CShmClientMgr	142
6.34.3	Member Function Documentation	142
6.34.3.1	Instance	142
6.34.3.2	initialize	142
6.34.3.3	finalize	143
6.34.3.4	notifyReady	143
6.34.3.5	notifyResponse	143
6.34.3.6	waitForRequest	143
6.34.3.7	retrieveRequest	143
6.34.3.8	createResponse	144
6.34.3.9	operator=	144
6.34.4	Member Data Documentation	144
6.34.4.1	m_pModuleSync	144
6.35	CShmemManager Class Reference	145
6.35.1	Detailed Description	147
6.35.2	Member Typedef Documentation	148
6.35.2.1	TPtrSModuleSync	148
6.35.3	Constructor & Destructor Documentation	148
6.35.3.1	CShmemManager	148
6.35.3.2	~CShmemManager	148
6.35.3.3	CShmemManager	148
6.35.4	Member Function Documentation	148
6.35.4.1	configure	148
6.35.4.2	getMemoryID	149
6.35.4.3	getMutexID	149
6.35.4.4	getMutex	149
6.35.4.5	Instance	149
6.35.4.6	initializeMainProcess	149
6.35.4.7	initializeChildProcess	150
6.35.4.8	finalizeMainProcess	150
6.35.4.9	finalizeChildProcess	150

6.35.4.10	allocateRequest	150
6.35.4.11	allocateResponse	150
6.35.4.12	retrieveRequest	151
6.35.4.13	retrieveResponse	151
6.35.4.14	getRequestDeleter	151
6.35.4.15	getResponseDeleter	151
6.35.4.16	createShmem	151
6.35.4.17	deleteShmem	152
6.35.4.18	openShmem	152
6.35.4.19	closeShmem	152
6.35.4.20	initializeHelpers	152
6.35.4.21	getRequestID	152
6.35.4.22	getResponseID	153
6.35.4.23	operator=	153
6.35.5	Friends And Related Function Documentation	153
6.35.5.1	ShmemLock	153
6.35.5.2	CShmClientMgr	153
6.35.5.3	CShmServerMgr	153
6.35.6	Member Data Documentation	153
6.35.6.1	m_pShMemory	153
6.35.6.2	m_pShMutex	153
6.35.6.3	m_pCharAllocator	153
6.35.6.4	m_pSubstAllocator	154
6.35.6.5	m_pRequestDeleter	154
6.35.6.6	m_pResponseDeleter	154
6.35.6.7	m_szPrefix	154
6.36	CShmemManager::SModuleSync Struct Reference	155
6.36.1	Detailed Description	155
6.36.2	Constructor & Destructor Documentation	156
6.36.2.1	SModuleSync	156
6.36.3	Member Function Documentation	156
6.36.3.1	create	156
6.36.3.2	open	156
6.36.3.3	close	156
6.36.3.4	destroy	156
6.36.3.5	getRequestConditionID	157

6.36.3.6	getResponseConditionID	157
6.36.3.7	getReadySemaphoreID	157
6.36.4	Member Data Documentation	157
6.36.4.1	m_szModuleID	157
6.36.4.2	m_pShReqCondition	157
6.36.4.3	m_pShRespCondition	157
6.36.4.4	m_pShReadySemaphore	158
6.37	CShmServerMgr Class Reference	159
6.37.1	Detailed Description	160
6.37.2	Member Typedef Documentation	160
6.37.2.1	TModuleSyncMap	160
6.37.3	Constructor & Destructor Documentation	161
6.37.3.1	CShmServerMgr	161
6.37.3.2	~CShmServerMgr	161
6.37.3.3	CShmServerMgr	161
6.37.4	Member Function Documentation	161
6.37.4.1	Instance	161
6.37.4.2	initialize	161
6.37.4.3	finalize	161
6.37.4.4	addModuleSync	162
6.37.4.5	waitForReady	162
6.37.4.6	waitForResponse	162
6.37.4.7	waitForResponse	162
6.37.4.8	notifyRequest	162
6.37.4.9	createRequest	163
6.37.4.10	retrieveResponse	163
6.37.4.11	freeSyncs	163
6.37.4.12	operator=	163
6.37.5	Member Data Documentation	163
6.37.5.1	m_mapModuleSync	163
6.38	CSingleModule Class Reference	165
6.38.1	Detailed Description	166
6.38.2	Member Typedef Documentation	167
6.38.2.1	TPtrModule	167
6.38.3	Constructor & Destructor Documentation	167
6.38.3.1	CSingleModule	167

6.38.3.2	~CSingleModule	167
6.38.3.3	CSingleModule	167
6.38.4	Member Function Documentation	167
6.38.4.1	Instance	167
6.38.4.2	run	167
6.38.4.3	extractRequest	168
6.38.4.4	writeResponse	168
6.38.4.5	processRequest	168
6.38.4.6	loadDll	169
6.38.4.7	unloadDll	169
6.38.4.8	extract_names_values	169
6.38.4.9	convert_operations	169
6.38.4.10	free_names_values	169
6.38.4.11	handleModuleError	170
6.38.4.12	operator=	170
6.38.5	Member Data Documentation	170
6.38.5.1	m_szModuleType	170
6.38.5.2	m_szModuleID	170
6.38.5.3	m_vecPaths	170
6.38.5.4	m_hLibrary	171
6.38.5.5	m_pModule	171
6.39	CTransformer Class Reference	172
6.39.1	Detailed Description	172
6.39.2	Constructor & Destructor Documentation	172
6.39.2.1	CTransformer	172
6.39.2.2	~CTransformer	173
6.39.2.3	CTransformer	173
6.39.3	Member Function Documentation	173
6.39.3.1	operator=	173
6.40	CTransformerChain Class Reference	174
6.40.1	Detailed Description	175
6.40.2	Constructor & Destructor Documentation	175
6.40.2.1	CTransformerChain	175
6.40.2.2	CTransformerChain	175
6.40.2.3	~CTransformerChain	175
6.40.2.4	CTransformerChain	175

6.40.3	Member Function Documentation	175
6.40.3.1	push_front	175
6.40.3.2	push_back	176
6.40.3.3	getTransformers	176
6.40.3.4	operator=	176
6.40.4	Member Data Documentation	176
6.40.4.1	m_pTransformers	176
6.41	CTransformerContext Class Reference	177
6.41.1	Detailed Description	178
6.41.2	Constructor & Destructor Documentation	178
6.41.2.1	CTransformerContext	178
6.41.2.2	~CTransformerContext	178
6.41.3	Member Function Documentation	178
6.41.3.1	isModuleDeclared	178
6.41.3.2	getModuleDeclaration	179
6.41.3.3	unify	179
6.41.3.4	unify	179
6.41.3.5	operator=	179
6.41.4	Friends And Related Function Documentation	179
6.41.4.1	CQueryContext	179
6.41.5	Member Data Documentation	179
6.41.5.1	m_pProgram	179
6.42	CTransformerSet Class Reference	181
6.42.1	Detailed Description	182
6.42.2	Constructor & Destructor Documentation	182
6.42.2.1	CTransformerSet	182
6.42.2.2	CTransformerSet	182
6.42.2.3	~CTransformerSet	182
6.42.2.4	CTransformerSet	182
6.42.3	Member Function Documentation	182
6.42.3.1	push_front	182
6.42.3.2	push_back	183
6.42.3.3	getTransformers	183
6.42.3.4	operator=	183
6.42.4	Member Data Documentation	183
6.42.4.1	m_pTransformers	183

6.43 CTraverser Class Reference	184
6.43.1 Detailed Description	186
6.43.2 Constructor & Destructor Documentation	187
6.43.2.1 CTraverser	187
6.43.2.2 ~CTraverser	187
6.43.2.3 CTraverser	187
6.43.3 Member Function Documentation	187
6.43.3.1 Visit	187
6.43.3.2 Visit	188
6.43.3.3 Visit	188
6.43.3.4 Visit	188
6.43.3.5 Visit	188
6.43.3.6 Visit	188
6.43.3.7 Visit	189
6.43.3.8 Visit	189
6.43.3.9 Visit	189
6.43.3.10 Visit	189
6.43.3.11 Visit	189
6.43.3.12 Visit	190
6.43.3.13 Visit	190
6.43.3.14 PreAction	190
6.43.3.15 PreAction	190
6.43.3.16 PreAction	190
6.43.3.17 PreAction	191
6.43.3.18 PreAction	191
6.43.3.19 PreAction	191
6.43.3.20 PreAction	191
6.43.3.21 PreAction	191
6.43.3.22 PreAction	191
6.43.3.23 PreAction	192
6.43.3.24 PreAction	192
6.43.3.25 PreAction	192
6.43.3.26 PreAction	192
6.43.3.27 PostAction	192
6.43.3.28 PostAction	192
6.43.3.29 PostAction	193

6.43.3.30 PostAction	193
6.43.3.31 PostAction	193
6.43.3.32 PostAction	193
6.43.3.33 PostAction	193
6.43.3.34 PostAction	193
6.43.3.35 PostAction	193
6.43.3.36 PostAction	194
6.43.3.37 PostAction	194
6.43.3.38 PostAction	194
6.43.3.39 PostAction	194
6.43.3.40 RuleMidAction	194
6.43.3.41 RuleMidAction	194
6.43.3.42 DefaultPreAction	195
6.43.3.43 DefaultPreAction	195
6.43.3.44 DefaultPostAction	195
6.43.3.45 DefaultPostAction	195
6.43.3.46 DefaultRuleMidAction	195
6.43.3.47 DefaultPreAction	195
6.43.3.48 DefaultPostAction	195
6.43.3.49 DefaultRuleMidAction	196
6.43.3.50 traverse	196
6.43.3.51 operator=	196
6.44 CTrueQuery Class Reference	197
6.44.1 Detailed Description	197
6.44.2 Constructor & Destructor Documentation	197
6.44.2.1 CTrueQuery	197
6.44.2.2 ~CTrueQuery	198
6.44.2.3 CTrueQuery	198
6.44.3 Member Function Documentation	198
6.44.3.1 operator=	198
6.45 jazzyk::KRModuleInterface Class Reference	199
6.45.1 Detailed Description	200
6.45.2 Member Function Documentation	200
6.45.2.1 initialize	200
6.45.2.2 finalize	200
6.45.2.3 cycle	200

6.45.2.4	getQueryInterface	201
6.45.2.5	getUpdateInterface	201
6.45.2.6	query	201
6.45.2.7	update	201
6.45.2.8	free_resource	202
6.45.2.9	destroy	202
6.45.2.10	operator delete	202
6.46	ShmemLock Struct Reference	203
6.46.1	Detailed Description	203
6.46.2	Constructor & Destructor Documentation	203
6.46.2.1	ShmemLock	203
6.47	SIdentifierCompare Struct Reference	204
6.47.1	Detailed Description	204
6.47.2	Member Function Documentation	204
6.47.2.1	operator()	204
6.48	SIdentifierEqual Struct Reference	205
6.48.1	Detailed Description	205
6.48.2	Member Function Documentation	205
6.48.2.1	operator()	205
6.49	SIpcStringCompare Struct Reference	206
6.49.1	Detailed Description	206
6.49.2	Member Function Documentation	206
6.49.2.1	operator()	206
6.50	SModuleRequest Struct Reference	207
6.50.1	Detailed Description	207
6.50.2	Constructor & Destructor Documentation	208
6.50.2.1	SModuleRequest	208
6.50.3	Member Function Documentation	208
6.50.3.1	addVariableSubst	208
6.50.3.2	addFreeVariable	208
6.50.4	Member Data Documentation	208
6.50.4.1	m_eType	208
6.50.4.2	m_szOperation	208
6.50.4.3	m_szCodeBlock	209
6.50.4.4	m_mapSubstitution	209
6.50.4.5	m_mapFreeVariables	209

6.50.4.6	m_charAllocator	209
6.51	SModuleResponse Struct Reference	210
6.51.1	Detailed Description	210
6.51.2	Constructor & Destructor Documentation	210
6.51.2.1	SModuleResponse	210
6.51.3	Member Function Documentation	211
6.51.3.1	addVariableSubst	211
6.51.4	Member Data Documentation	211
6.51.4.1	m_eError	211
6.51.4.2	m_mapSubstitution	211
6.51.4.3	m_bQueryResult	211
6.51.4.4	m_charAllocator	211
6.52	jazzyk_private::StringCompare Struct Reference	212
6.52.1	Detailed Description	212
6.52.2	Member Function Documentation	212
6.52.2.1	operator()	212
6.53	TIdentifierMap< T > Class Template Reference	213
6.53.1	Detailed Description	213
6.54	visitable< R > Class Template Reference	214
6.54.1	Detailed Description	214
6.54.2	Member Typedef Documentation	214
6.54.2.1	TReturnType	214
6.54.3	Constructor & Destructor Documentation	215
6.54.3.1	~visitable	215
6.54.4	Member Function Documentation	215
6.54.4.1	Accept	215
6.54.4.2	AcceptImpl	215
6.55	visitable_arg< A, R > Class Template Reference	216
6.55.1	Detailed Description	216
6.55.2	Member Typedef Documentation	216
6.55.2.1	TReturnType	216
6.55.2.2	TArgType	217
6.55.3	Constructor & Destructor Documentation	217
6.55.3.1	~visitable_arg	217
6.55.4	Member Function Documentation	217
6.55.4.1	Accept	217

6.55.4.2	AcceptImpl	217
6.56	visitor< T, R > Class Template Reference	218
6.56.1	Detailed Description	218
6.56.2	Member Typedef Documentation	218
6.56.2.1	TReturnType	218
6.56.3	Constructor & Destructor Documentation	218
6.56.3.1	~visitor	218
6.56.4	Member Function Documentation	219
6.56.4.1	Visit	219
6.57	visitor_arg< T, A, R > Class Template Reference	220
6.57.1	Detailed Description	220
6.57.2	Member Typedef Documentation	220
6.57.2.1	TReturnType	220
6.57.2.2	TArgType	221
6.57.3	Constructor & Destructor Documentation	221
6.57.3.1	~visitor_arg	221
6.57.4	Member Function Documentation	221
6.57.4.1	Visit	221
7	File Documentation	223
7.1	AndQueryNode.hpp File Reference	223
7.2	CodeBlock.cpp File Reference	224
7.3	CodeBlock.hpp File Reference	225
7.4	Compiler.cpp File Reference	226
7.5	Compiler.hpp File Reference	227
7.6	Context.cpp File Reference	228
7.7	Context.hpp File Reference	229
7.8	ElseRule.hpp File Reference	230
7.9	Error.cpp File Reference	231
7.9.1	Function Documentation	231
7.9.1.1	operator<<	231
7.10	Error.hpp File Reference	232
7.10.1	Function Documentation	232
7.10.1.1	operator<<	232
7.11	ExplicitQuery.hpp File Reference	233
7.12	ExplicitUpdate.hpp File Reference	234
7.13	Expression.cpp File Reference	235

7.14	Expression.hpp File Reference	236
7.14.1	Typedef Documentation	236
7.14.1.1	TSrcLocation	236
7.15	ExpressionsFwd.hpp File Reference	237
7.15.1	Typedef Documentation	237
7.15.1.1	TIdentifiers	237
7.15.1.2	TIdentifierSet	238
7.15.1.3	TModuleDeclCollection	238
7.15.1.4	TPtrCodeBlock	238
7.15.1.5	TPtrExitUpdate	238
7.15.1.6	TPtrIdentifier	238
7.15.1.7	TPtrIdentifiers	238
7.15.1.8	TPtrModuleDeclaration	238
7.15.1.9	TPtrModuleDeclCollection	238
7.15.1.10	TPtrModuleNotification	238
7.15.1.11	TPtrNopUpdate	238
7.15.1.12	TPtrPlugin	238
7.15.1.13	TPtrProgram	239
7.15.1.14	TPtrQueryNode	239
7.15.1.15	TPtrTransformer	239
7.15.1.16	TPtrTransformerChain	239
7.15.1.17	TPtrTransformerContext	239
7.15.1.18	TPtrTransformers	239
7.15.1.19	TPtrTransformerSet	239
7.15.1.20	TPtrVariableSubstitution	239
7.15.1.21	TPtrVariableValue	239
7.15.1.22	TSrcLocation	239
7.15.1.23	TTransformers	239
7.15.1.24	TVariableSubstitution	240
7.15.1.25	TVariableValue	240
7.15.1.26	TVecStrings	240
7.15.1.27	TVecTransformers	240
7.16	Identifier.hpp File Reference	241
7.17	IdentifierMap.hpp File Reference	242
7.18	Interpreter.cpp File Reference	243
7.18.1	Function Documentation	243

7.18.1.1	basic_random	243
7.19	Interpreter.hpp File Reference	244
7.20	jazzyk.hpp File Reference	245
7.20.1	Define Documentation	246
7.20.1.1	CALL_MEMBER_FN	246
7.20.1.2	JZMODULE_MANIFEST_BEGIN	246
7.20.1.3	JZMODULE_MANIFEST_END	246
7.20.1.4	REGISTER_QUERY	247
7.20.1.5	REGISTER_UPDATE	247
7.21	jazzyk_common.hpp File Reference	248
7.21.1	Define Documentation	249
7.21.1.1	__JZDLL_CALL	249
7.21.1.2	__JZDLL_EXPORT	249
7.21.1.3	JZMODULE_API_REGISTER_ROUTINE	249
7.21.1.4	JZMODULE_API_REGISTER_ROUTINE_ID	249
7.21.1.5	JZMODULE_API_VERSION_ROUTINE	249
7.21.1.6	JZMODULE_API_VERSION_ROUTINE_ID	249
7.21.2	Variable Documentation	249
7.21.2.1	JAZZYK_API_VERSION	249
7.22	JazzykApp.cpp File Reference	250
7.22.1	Define Documentation	250
7.22.1.1	MOTTO	250
7.22.2	Variable Documentation	250
7.22.2.1	yydebug	250
7.23	JazzykApp.hpp File Reference	251
7.23.1	Typedef Documentation	251
7.23.1.1	JazzykApp	251
7.24	Lexer.l++ File Reference	252
7.24.1	Define Documentation	252
7.24.1.1	This	252
7.24.1.2	ThisYylloc	252
7.24.1.3	ThisYylval	252
7.24.1.4	YY_USER_ACTION	252
7.24.2	Function Documentation	252
7.24.2.1	step	252
7.25	LocationTracker.hpp File Reference	253

7.26	Logging.hpp File Reference	254
7.26.1	Define Documentation	254
7.26.1.1	LOG	254
7.27	main.cpp File Reference	255
7.27.1	Function Documentation	255
7.27.1.1	main	255
7.28	ModuleCommData.hpp File Reference	256
7.28.1	Define Documentation	258
7.28.1.1	g_JZAPI_QUERY	258
7.28.1.2	g_JZAPI_UPDATE	258
7.28.2	Typedef Documentation	258
7.28.2.1	TIPCCharAllocator	258
7.28.2.2	TIPCString	258
7.28.2.3	TIPCSubst	258
7.28.2.4	TIPCSubstAllocator	259
7.28.2.5	TIPCSubstValueType	259
7.28.2.6	TPtrIPCCharAllocator	259
7.28.2.7	TPtrIPCCondition	259
7.28.2.8	TPtrIPCManagedSharedMemory	259
7.28.2.9	TPtrIPCMutex	259
7.28.2.10	TPtrIPCSemaphore	259
7.28.2.11	TPtrIPCSubstAllocator	259
7.28.2.12	TPtrSModuleRequest	260
7.28.2.13	TPtrSModuleRequestDeleter	260
7.28.2.14	TPtrSModuleResponse	260
7.28.2.15	TPtrSModuleResponseDeleter	260
7.28.2.16	TSMModuleRequestDeleter	260
7.28.2.17	TSMModuleResponseDeleter	260
7.28.3	Enumeration Type Documentation	260
7.28.3.1	EModuleErrCode	260
7.28.3.2	EModuleRequestType	260
7.29	ModuleDeclaration.cpp File Reference	262
7.30	ModuleDeclaration.hpp File Reference	263
7.31	ModuleManager.cpp File Reference	264
7.32	ModuleManager.hpp File Reference	265
7.33	ModuleNotification.hpp File Reference	266

7.34	NotQueryNode.hpp File Reference	267
7.35	OrQueryNode.hpp File Reference	268
7.36	Parser.y++ File Reference	269
7.36.1	Define Documentation	269
7.36.1.1	yylex	269
7.36.2	Function Documentation	269
7.36.2.1	push_back	269
7.36.2.2	TPtrCodeBlock	269
7.36.2.3	TPtrIdentifier	269
7.36.2.4	TPtrIdentifiers	270
7.36.2.5	TPtrTransformer	270
7.36.3	Variable Documentation	270
7.36.3.1	__pad0__	270
7.36.3.2	__pad10__	270
7.36.3.3	__pad11__	270
7.36.3.4	__pad12__	270
7.36.3.5	__pad13__	270
7.36.3.6	__pad1__	270
7.36.3.7	__pad2__	270
7.36.3.8	__pad3__	270
7.36.3.9	__pad4__	270
7.36.3.10	__pad5__	271
7.36.3.11	__pad6__	271
7.36.3.12	__pad7__	271
7.36.3.13	__pad8__	271
7.36.3.14	__pad9__	271
7.37	ParserAux.cpp File Reference	272
7.38	ParserAux.hpp File Reference	273
7.38.1	Typedef Documentation	274
7.38.1.1	TPtrLexer	274
7.39	PrettyPrinter.cpp File Reference	275
7.40	PrettyPrinter.hpp File Reference	276
7.41	Program.cpp File Reference	277
7.42	Program.hpp File Reference	278
7.42.1	Typedef Documentation	278
7.42.1.1	TPtrProgram	278

7.43	QueryNode.hpp File Reference	279
7.44	Rule.hpp File Reference	280
7.45	ShmClientMgr.cpp File Reference	281
7.46	ShmClientMgr.hpp File Reference	282
7.47	ShmemManager.cpp File Reference	283
7.47.1	Function Documentation	283
7.47.1.1	g_szMemorySuffix	283
7.47.1.2	g_szMutexSuffix	283
7.47.1.3	g_szReadySemaphoreSuffix	283
7.47.1.4	g_szRequestConditionSuffix	283
7.47.1.5	g_szRequestSuffix	284
7.47.1.6	g_szResponseConditionSuffix	284
7.47.1.7	g_szResponseSuffix	284
7.48	ShmemManager.hpp File Reference	285
7.49	ShmServerMgr.cpp File Reference	286
7.49.1	Function Documentation	286
7.49.1.1	g_szSharedMemoryPrefix	286
7.50	ShmServerMgr.hpp File Reference	287
7.51	SingleModule.cpp File Reference	288
7.51.1	Function Documentation	288
7.51.1.1	cszLibNamePrefix	288
7.51.1.2	cszLibNameSuffix	288
7.52	SingleModule.hpp File Reference	289
7.53	Transformer.hpp File Reference	290
7.54	TransformerChain.cpp File Reference	291
7.55	TransformerChain.hpp File Reference	292
7.56	TransformerSet.cpp File Reference	293
7.57	TransformerSet.hpp File Reference	294
7.58	Traverser.cpp File Reference	295
7.59	Traverser.hpp File Reference	296
7.60	TrivialQuery.hpp File Reference	297
7.61	TrivialUpdate.hpp File Reference	298
7.62	visitor.hpp File Reference	299
7.62.1	Define Documentation	299
7.62.1.1	DECLARE_VISITABLE	299
7.62.1.2	DECLARE_VISITABLE	300

7.62.1.3	DECLARE_VISITABLE	300
7.62.1.4	DECLARE_VISITABLE_ARG	300
7.62.1.5	DECLARE_VISITABLE_ARG	300
7.62.1.6	DECLARE_VISITABLE_ARG	300

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

jazzyk (Namespace defining Jazzyk API for KR modules)	11
jazzyk_private (Private namespace to be used in jazzyk (p.11))	13
yy	14

Chapter 2

Class Index

2.1 Class Hierarchy

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

jazyk::AbstractKRModule	15
base_visitor	26
CTraverser	184
CCompiler	33
CInterpreter	70
CPrettyPrinter	119
CCodeBlock	30
CCompiler::SMPInvocationSettings	41
CError	47
CExpression	62
CQueryNode	136
CAndQueryNode	27
CExplicitQuery	54
CFalseQuery	65
CNotQueryNode	114
COrQueryNode	116
CTrueQuery	197
CTransformer	172
CExitUpdate	52
CExplicitUpdate	58
CNopUpdate	112
CRule	138
CElseRule	44
CTransformerChain	174
CTransformerSet	181
CIdentifier	67
CInterpreter::SSettings	76
CIPCPtrDeleter< T >	78
CJazykApp	80
CJazykApp::SSettings	84
CJazykLexer	88
CLocationTracker	91

CModuleDeclaration	94
CModuleManager	100
CModuleNotification	108
CProgram	125
CShmClientMgr	141
CShmemManager	145
CShmemManager::SModuleSync	155
CShmServerMgr	159
CSingleModule	165
jazyk::KRModuleInterface	199
jazyk::AbstractKRModuleInterface< TConcreteKRModule >	17
ShmemLock	203
SIdentifierCompare	204
SIdentifierEqual	205
SIPCStringCompare	206
SModuleRequest	207
SModuleResponse	210
jazyk_private::StringCompare	212
TIdentifierMap< T >	213
CTransformerContext	177
CQueryContext	130
TIdentifierMap< boost::shared_ptr< TVariableValue > >	213
visitable< R >	214
visitable_arg< A, R >	216
visitable_arg< CQueryContext, CQueryContext >	216
CQueryNode	136
visitable_arg< CTransformerContext, bool >	216
CTransformer	172
visitor< T, R >	218
visitor_arg< T, A, R >	220
visitor_arg< CAndQueryNode, CQueryContext, CQueryContext >	220
CTraverser	184
visitor_arg< CElseRule, CTransformerContext, bool >	220
CTraverser	184
visitor_arg< CExitUpdate, CTransformerContext, bool >	220
CTraverser	184
visitor_arg< CExplicitQuery, CQueryContext, CQueryContext >	220
CTraverser	184
visitor_arg< CExplicitUpdate, CTransformerContext, bool >	220
CTraverser	184
visitor_arg< CFalseQuery, CQueryContext, CQueryContext >	220
CTraverser	184
visitor_arg< CNopUpdate, CTransformerContext, bool >	220
CTraverser	184
visitor_arg< CNotQueryNode, CQueryContext, CQueryContext >	220
CTraverser	184
visitor_arg< COrQueryNode, CQueryContext, CQueryContext >	220
CTraverser	184
visitor_arg< CRule, CTransformerContext, bool >	220

CTraverser	184
visitor_arg< CTransformerChain, CTransformerContext, bool >	220
CTraverser	184
visitor_arg< CTransformerSet, CTransformerContext, bool >	220
CTraverser	184
visitor_arg< CTrueQuery, CQueryContext, CQueryContext >	220
CTraverser	184

Chapter 3

Class Index

3.1 Class List

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

jazyk::AbstractKRModule (Concrete knowledge representation module base class)	15
jazyk::AbstractKRModuleInterface < TConcreteKRModule > (The abstract knowledge representation module interface adapter)	17
base_visitor (Visitor pattern library)	26
CAndQueryNode (Query node for AND logical connector of queries)	27
CCodeBlock (Code of block holder)	30
CCompiler (Jazyk programming language compiler)	33
CCompiler::SMPInvocationSettings (Configuration structure for the macro preprocessor invocation)	41
CElseRule (If-then-else type rule)	44
CError (Error handler for the application)	47
CExitUpdate (Exit command mental state transformer implementation)	52
CExplicitQuery (Explicit query class)	54
CExplicitUpdate (Explicit update class)	58
CExpression (Base abstract class for all the query and update expressions)	62
CFalseQuery (Trivial False query)	65
CIdentifier (Identifier holder)	67
CInterpreter (Interpreter engine of the Jazyk language interpreter)	70
CInterpreter::SSettings (Interpreter configuration)	76
CIpcPtrDeleter < T > (A functor that destroys the shared memory object of an associated smart pointer)	78
CJazykApp (Main Jazyk application class)	80
CJazykApp::SSettings (Settings wrapper)	84
CJazykLexer (Specialisation of the standard Flex C++ yyFlexLexer class)	88
CLocationTracker (Small singleton for communication between the Bison location and CExpressions)	91
CModuleDeclaration (Module declaration and link holder)	94
CModuleManager (External plug-ins/modules manager)	100
CModuleNotification (Wrapper class for the module notification declaration)	108
CNoUpdate (Empty update class implementation)	112
CNotQueryNode (Query node for NOT logical modifier of a query expression)	114
COrQueryNode (Query node for OR logical connector of queries)	116
CPrettyPrinter (Jazyk program pretty printig service class)	119

CProgram (The Jazzyk program backend structure)	125
CQueryContext (Interpreter context used while traversing the query subtree)	130
CQueryNode (Query node abstract base class)	136
CRule (If-then rule mental state transformer)	138
CShmClientMgr (Client side interprocess communication manager)	141
CShmMemManager (Service managing the shared memory between the main process and plug-in processes)	145
CShmMemManager::SModuleSync (Synchronisation conditions structure)	155
CShmServerMgr (Server side interprocess communication manager)	159
CSingleModule (Wrapper for a KR module)	165
CTransformer (Abstract class for all the various types of mental state transformers) .	172
CTransformerChain (Chain of mental state transformers)	174
CTransformerContext (Interpreter context used while traversing the transformer tree structure)	177
CTransformerSet (Set of mental state transformers)	181
CTraverser (Program transformer tree traversal abstract class)	184
CTrueQuery (Trivial True query)	197
jazzyk::KRModuleInterface (Abstract base class for defining knowledge representa- tion modules)	199
ShmemLock (Specialised shared memory lock)	203
SIdentifierCompare (Comparison operator)	204
SIdentifierEqual (Equality operator)	205
SIPCStringCompare (String comparison operator)	206
SModuleRequest (Plug-in request structure)	207
SModuleResponse (Plug-in response structure)	210
jazzyk_private::StringCompare (String comparison operator)	212
TIdentifierMap< T > (CIdentifier (p. 67) map)	213
visitable< R > (Base class for visitable (p. 214) classes)	214
visitable_arg< A, R > (Version of visitable (p. 214) class with argument)	216
visitor< T, R > (Visitor class template)	218
visitor_arg< T, A, R > (Visitor passing an argument class template)	220

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

AndQueryNode.hpp	223
CodeBlock.cpp	224
CodeBlock.hpp	225
Compiler.cpp	226
Compiler.hpp	227
Context.cpp	228
Context.hpp	229
ElseRule.hpp	230
Error.cpp	231
Error.hpp	232
ExplicitQuery.hpp	233
ExplicitUpdate.hpp	234
Expression.cpp	235
Expression.hpp	236
ExpressionsFwd.hpp	237
Identifier.hpp	241
IdentifierMap.hpp	242
Interpreter.cpp	243
Interpreter.hpp	244
jazyk.hpp	245
jazyk_common.hpp	248
JazykApp.cpp	250
JazykApp.hpp	251
Lexer.l++	252
LocationTracker.hpp	253
Logging.hpp	254
main.cpp	255
ModuleCommData.hpp	256
ModuleDeclaration.cpp	262
ModuleDeclaration.hpp	263
ModuleManager.cpp	264
ModuleManager.hpp	265
ModuleNotification.hpp	266

NotQueryNode.hpp	267
OrQueryNode.hpp	268
Parser.y++	269
ParserAux.cpp	272
ParserAux.hpp	273
PrettyPrinter.cpp	275
PrettyPrinter.hpp	276
Program.cpp	277
Program.hpp	278
QueryNode.hpp	279
Rule.hpp	280
ShmClientMgr.cpp	281
ShmClientMgr.hpp	282
ShmemManager.cpp	283
ShmemManager.hpp	285
ShmServerMgr.cpp	286
ShmServerMgr.hpp	287
SingleModule.cpp	288
SingleModule.hpp	289
Transformer.hpp	290
TransformerChain.cpp	291
TransformerChain.hpp	292
TransformerSet.cpp	293
TransformerSet.hpp	294
Traverser.cpp	295
Traverser.hpp	296
TrivialQuery.hpp	297
TrivialUpdate.hpp	298
visitor.hpp	299

Chapter 5

Namespace Documentation

5.1 jazzyk Namespace Reference

Namespace defining Jazzyk API for KR modules.

Classes

- class **AbstractKRModuleInterface**
The abstract knowledge representation module interface adapter.
- class **AbstractKRModule**
Concrete knowledge representation module base class.
- class **KRModuleInterface**
Abstract base class for defining knowledge representation modules.

Typedefs

- typedef std::map< std::string, std::string, jazzyk_private::StringCompare > **TKRVarSubstitution**
Type definition for variable substitution map.
- typedef unsigned int(* **TJzModuleApiVersionRoutine**)()
Signature of the version API routine.
- typedef **jazzyk::KRModuleInterface** *(* **TJzModuleApiRegisterRoutine**)()
Signature of the KR module registration routine.

Enumerations

- enum **EKRModuleError** { **OK**, **FAIL**, **INVALID_OPERATION** }
Error codes returned from the KR module to the main process.

5.1.1 Detailed Description

Namespace defining Jazzyk API for KR modules.

5.1.2 Typedef Documentation

5.1.2.1 `typedef jazzyk::KRModuleInterface*(* jazzyk::TJzModuleApiRegisterRoutine)()`

Signature of the KR module registration routine.

Definition at line 200 of file `jazzyk_common.hpp`.

5.1.2.2 `typedef unsigned int(* jazzyk::TJzModuleApiVersionRoutine)()`

Signature of the version API routine.

Definition at line 197 of file `jazzyk_common.hpp`.

5.1.2.3 `typedef std::map<std::string, std::string, jazzyk_private::StringCompare> jazzyk::TKRVarSubstitution`

Type definition for variable substitution map.

Definition at line 67 of file `jazzyk.hpp`.

5.1.3 Enumeration Type Documentation

5.1.3.1 `enum jazzyk::EKRRModuleError`

Error codes returned from the KR module to the main process.

Enumerator:

OK

FAIL

INVALID_OPERATION

Definition at line 73 of file `jazzyk_common.hpp`.

5.2 jazzyk_private Namespace Reference

Private namespace to be used in **jazzyk** (p. 11).

Classes

- struct **StringCompare**
String comparison operator.

5.2.1 Detailed Description

Private namespace to be used in **jazzyk** (p. 11).

Contains internal typedefs etc.

5.3 `yy` Namespace Reference

5.3.1 Detailed Description

Forward declarations of the whole **CExpression** (p. 62) hierarchy.

Remark: introduced to speed up the compilation and avoid unnecessary multiple header inclusion in other headers.

Chapter 6

Class Documentation

6.1 jazzyk::AbstractKRModule Class Reference

Concrete knowledge representation module base class.

```
#include <jazzyk.hpp>
```

Public Member Functions

- virtual **EKRModuleError initialize** (const std::string &szCodeBlock)=0
Interface to override in the subclasses.
- virtual **EKRModuleError finalize** (const std::string &szCodeBlock)=0
KR module finalization notification.
- virtual **EKRModuleError cycle** (const std::string &szCodeBlock)=0
Interpreter cycle notification.

6.1.1 Detailed Description

Concrete knowledge representation module base class.

The concrete module implementation should inherit from this base class. It has to implement the abstract methods, plus provide a query/update interface which then has to be registered using JZMODULE_MANIFEST macro family.

Definition at line 206 of file jazzyk.hpp.

6.1.2 Member Function Documentation

6.1.2.1 virtual EKRModuleError jazzyk::AbstractKRModule::initialize (const std::string & szCodeBlock) [pure virtual]

Interface to override in the subclasses.

KR module initial notification

The method is invoked right after the KR module is loaded.

It gets an initialization code supposed to contain preparation for the Jazzyk program interpretation.

6.1.2.2 virtual EKRModuleError jazzyk::AbstractKRModule::finalize (const std::string & *szCodeBlock*) [pure virtual]

KR module finalization notification.

The method is called right before the KR module is about to be closed and unloaded.

It gets the finalization code supposed to clean up and release all the resources possibly held, or allocated by the module during the initialization and module lifetime.

6.1.2.3 virtual EKRModuleError jazzyk::AbstractKRModule::cycle (const std::string & *szCodeBlock*) [pure virtual]

Interpreter cycle notification.

After each deliberation cycle finishes (an update operation was performed in one of the program's modules), all modules receive a cycle notification.

The provided code block is supposed to free all resources possibly allocated during the last deliberation cycle (e.g. cache, etc.).

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

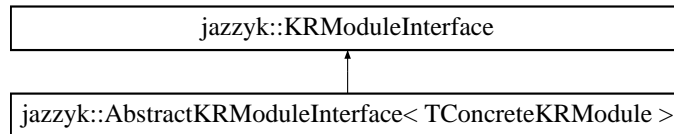
- **jazzyk.hpp**

6.2 jazzyk::AbstractKRModuleInterface< TConcreteKRModule > Class Template Reference

The abstract knowledge representation module interface adapter.

```
#include <jazzyk.hpp>
```

Inheritance diagram for jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::



Public Member Functions

- **AbstractKRModuleInterface** ()
Constructor.
- virtual **~AbstractKRModuleInterface** ()
Virtual destructor.
- virtual **EKRModuleError initialize** (const char *szCodeBlock)
Implementations of the virtual superclass interface.
- virtual **EKRModuleError finalize** (const char *szCodeBlock)
Invokes the finalize function.
- virtual **EKRModuleError cycle** (const char *szCodeBlock)
Invokes the cycle function.
- virtual **EKRModuleError query** (const char *szOperation, const char *szQueryCode, const char **arrInVarID, const char **arrInVarValue, const unsigned int nInArrSize, char **&arrOutVarID, char **&arrOutVarValue, unsigned int &nOutArrSize, bool &bResult)
Query to the KR module.
- virtual **EKRModuleError update** (const char *szOperation, const char *szUpdateCode, const char **arrVarID, const char **arrVarValue, const unsigned int nArrSize)
Update the KR module.
- virtual **EKRModuleError getQueryInterface** (char **&arrQueryOperations, unsigned int &nSize)
Fills an array of available query operations into the argument empty pointer.
- virtual **EKRModuleError getUpdateInterface** (char **&arrUpdateOperations, unsigned int &nSize)
Fills an array of available update operations into the argument empty pointer.
- virtual void **free_resource** (char **&array, const unsigned int nSize)

Helper method for correct variable substitution deallocation.

- void **registerQuery** (const std::string szOperation, **TPtrQueryHandler** ptrQueryFunc)
Query/Update interface registration.
- void **registerUpdate** (const std::string szOperation, **TPtrUpdateHandler** ptrUpdateFunc)
- virtual void **__JZDLL_CALL destroy** ()
Artificial virtual destruction.
- void **operator delete** (void *ptr)
Custom delete of the __JZDLL interface.

Private Types

- typedef **EKRModuleError**(TConcreteKRModule::* **TPtrQueryHandler**)(const std::string &szQueryCode, const **TKRVarSubstitution** &mapInSubst, **TKRVarSubstitution** &mapOutSubst, bool &bResult)
Type definition for pointer-to-member to the adaptee class query function.
- typedef **EKRModuleError**(TConcreteKRModule::* **TPtrUpdateHandler**)(const std::string &szUpdateCode, const **TKRVarSubstitution** &mapInSubst)
Type definition for pointer-to-member to the adaptee update function.
- typedef std::map< std::string, **TPtrQueryHandler**, jazzyk_private::StringCompare > **TQueryRegister**
Type definitions for query/update register.
- typedef std::map< std::string, **TPtrUpdateHandler**, jazzyk_private::StringCompare > **TUpdateRegister**

Private Member Functions

- void **map2raw** (const **TKRVarSubstitution** &mapSubstitution, char **&arrVarID, char **&arrVarValue, unsigned int &nSize)
Converts the variable substitution in the form of STL container into raw data.
- void **raw2map** (const char **&arrVarID, const char **&arrVarValue, const unsigned int nSize, **jazzyk::TKRVarSubstitution** &mapSubstitution)
Converts the variable substitution in the raw form into an STL container.

Private Attributes

- TConcreteKRModule **m_concreteKRModule**
Instance of the adaptee class.
- **TQueryRegister** **m_QueryOperations**

Register of query operations.

- **TUpdateRegister m_UpdateOperations**

Register of update operations.

6.2.1 Detailed Description

`template<class TConcreteKRModule> class jazzyk::AbstractKRModuleInterface< TConcreteKRModule >`

The abstract knowledge representation module interface adapter.

The template should be parametrized with the particular KR module interface implementation. It holds pointers-to-members of the parameter class and serves as an adapter between the main interpreter and the concrete KR module implementation.

The instance is used on the "server" side, i.e. in the main interpreter.

The destruction implementation complements the one in the RawKRModuleInterface and is inspired by the article <http://aegisknight.org/cppinterface.html> by Chad Austin.

Definition at line 84 of file jazzyk.hpp.

6.2.2 Member Typedef Documentation

6.2.2.1 `template<class TConcreteKRModule> typedef EKRRModuleError(TConcreteKRModule::* jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::TPtrQueryHandler)(const std::string &szQueryCode, const TKRVarSubstitution &mapInSubst, TKRVarSubstitution &mapOutSubst, bool &bResult) [private]`

Type definition for pointer-to-member to the adaptee class query function.

6.2.2.2 `template<class TConcreteKRModule> typedef EKRRModuleError(TConcreteKRModule::* jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::TPtrUpdateHandler)(const std::string &szUpdateCode, const TKRVarSubstitution &mapInSubst) [private]`

Type definition for pointer-to-member to the adaptee update function.

6.2.2.3 `template<class TConcreteKRModule> typedef std::map<std::string, TPtrQueryHandler, jazzyk_private::StringCompare> jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::TQueryRegister [private]`

Type definitions for query/update register.

Definition at line 101 of file jazzyk.hpp.

```

6.2.2.4 template<class TConcreteKRModule> typedef std::map<std::string,
TPtrUpdateHandler, jazzyk_private::StringCompare>
jazzyk::AbstractKRModuleInterface< TConcreteKRModule
>::TUpdateRegister [private]

```

Definition at line 102 of file jazzyk.hpp.

6.2.3 Constructor & Destructor Documentation

```

6.2.3.1 template<class TConcreteKRModule>
jazzyk::AbstractKRModuleInterface< TConcreteKRModule
>::AbstractKRModuleInterface () [inline]

```

Constructor.

Definition at line 340 of file jazzyk.hpp.

```

6.2.3.2 template<class TConcreteKRModule>
jazzyk::AbstractKRModuleInterface< TConcreteKRModule
>::~~AbstractKRModuleInterface () [inline, virtual]

```

Virtual destructor.

Definition at line 349 of file jazzyk.hpp.

6.2.4 Member Function Documentation

```

6.2.4.1 template<class TConcreteKRModule> jazzyk::EKRRModuleError
jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::initialize
(const char * szCodeBlock) [inline, virtual]

```

Implementations of the virtual superclass interface.

Forwarding the call to the adaptee instance.

Implements **jazzyk::KRModuleInterface** (p.200).

Definition at line 369 of file jazzyk.hpp.

References **jazzyk::FAIL**, and **jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_concreteKRModule**.

```

6.2.4.2 template<class TConcreteKRModule> jazzyk::EKRRModuleError
jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::finalize
(const char * szCodeBlock) [inline, virtual]

```

Invokes the finalize function.

To be called when the module is being closed and de-initialized. It is supposed to provide an opportunity to clean up all the held resources.

Implements **jazzyk::KRModuleInterface** (p.200).

Definition at line 380 of file jazzyk.hpp.

References jazzyk::FAIL, and jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_concreteKRModule.

6.2.4.3 `template<class TConcreteKRModule> jazzyk::EKRRModuleError
jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::cycle (const
char * szCodeBlock) [inline, virtual]`

Invokes the cycle function.

It is invoked at the end of each deliberation cycle of the main interpreter. It is supposed to clean up all the held resources which were acquired during the deliberation cycle execution (possible caching).

Implements **jazzyk::KRModuleInterface** (p.200).

Definition at line 391 of file jazzyk.hpp.

References jazzyk::FAIL, and jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_concreteKRModule.

6.2.4.4 `template<class TConcreteKRModule> jazzyk::EKRRModuleError
jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::query
(const char * szOperation, const char * szQueryCode, const char **
arrInVarID, const char ** arrInVarValue, const unsigned int nInArrSize,
char **& arrOutVarID, char **& arrOutVarValue, unsigned int &
nOutArrSize, bool & bResult) [inline, virtual]`

Query to the KR module.

Takes a query formula (code), an input variable substitution for this formula, and a query operation to perform. Returns a truth value of the query operation and a new variable substitution (result of the query).

IMPORTANT: After successful processing of the output substitution, the output substitution has to be correctly de-allocated using `free_resource(.)` method below.

Implements **jazzyk::KRModuleInterface** (p.201).

Definition at line 402 of file jazzyk.hpp.

References `CALL_MEMBER_FN`, `jazzyk::FAIL`, `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::free_resource()`, `jazzyk::INVALID_OPERATION`, `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_QueryOperations`, `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::map2raw()`, and `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::raw2map()`.

6.2.4.5 `template<class TConcreteKRModule> jazzyk::EKRRModuleError
jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::update
(const char * szOperation, const char * szQueryCode, const char **
arrVarID, const char ** arrVarValue, const unsigned int nArrSize)
[inline, virtual]`

Update the KR module.

Takes an update formula, a variable substitution for this formula and an update operation to perform.

Implements **jazzyk::KRModuleInterface** (p.201).

Definition at line 444 of file jazzyk.hpp.

References `CALL_MEMBER_FN`, `jazzyk::FAIL`, `jazzyk::INVALID_OPERATION`, `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_UpdateOperations`, and `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::raw2map()`.

6.2.4.6 `template<class TConcreteKRModule> jazzyk::EKRModuleError
jazzyk::AbstractKRModuleInterface< TConcreteKRModule
>::getQueryInterface (char **& arrQueryOperations, unsigned int &
nSize) [inline, virtual]`

Fills an array of available query operations into the argument empty pointer.

The method **replaces** the argument pointer, so the memory gets lost! Use empty pointers only and then do not forget to free the allocated memory using `free_resource(...)`.

The second argument receives the size of the returned array.

Implements `jazzyk::KRModuleInterface` (p.201).

Definition at line 469 of file jazzyk.hpp.

References `jazzyk::FAIL`, `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_QueryOperations`, and `jazzyk::OK`.

6.2.4.7 `template<class TConcreteKRModule> jazzyk::EKRModuleError
jazzyk::AbstractKRModuleInterface< TConcreteKRModule
>::getUpdateInterface (char **& arrUpdateOperations, unsigned int &
nSize) [inline, virtual]`

Fills an array of available update operations into the argument empty pointer.

The method **replaces** the argument pointer, so the memory gets lost! Use empty pointers only and then do not forget to free the allocated memory using `free_resource(...)`.

The second argument receives the size of the returned array.

Implements `jazzyk::KRModuleInterface` (p.201).

Definition at line 492 of file jazzyk.hpp.

References `jazzyk::FAIL`, `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_UpdateOperations`, and `jazzyk::OK`.

6.2.4.8 `template<class TConcreteKRModule> void
jazzyk::AbstractKRModuleInterface< TConcreteKRModule
>::free_resource (char **& array, const unsigned int nSize) [inline,
virtual]`

Helper method for correct variable substitution deallocation.

Implements `jazzyk::KRModuleInterface` (p.202).

Definition at line 515 of file jazzyk.hpp.

Referenced by `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::query()`.

6.2.4.9 `template<class TConcreteKRModule> void jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::registerQuery (const std::string szOperation, TPtrQueryHandler ptrQueryFunc) [inline]`

Query/Update interface registration.

The operation identifier is associated with the provided module member function.

It is not used directly, but rather auto-generated by the JZMODULE_MANIFEST macro family.

Definition at line 573 of file jazzyk.hpp.

References jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_QueryOperations.

6.2.4.10 `template<class TConcreteKRModule> void jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::registerUpdate (const std::string szOperation, TPtrUpdateHandler ptrUpdateFunc) [inline]`

Definition at line 580 of file jazzyk.hpp.

References jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_UpdateOperations.

6.2.4.11 `template<class TConcreteKRModule> void jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::destroy () [inline, virtual]`

Artificial virtual destruction.

Implements the correct __JZDLL destruction mechanism. The user should NOT reimplement this method!!! For more see the article by Chad Austin mentioned above.

Implements jazzyk::KRModuleInterface (p.202).

Definition at line 355 of file jazzyk.hpp.

6.2.4.12 `template<class TConcreteKRModule> void jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::operator delete (void * ptr) [inline]`

Custom delete of the __JZDLL interface.

Complements the polymorphic destruction technique for __JZDLL interface. See comments above.

Reimplemented from jazzyk::KRModuleInterface (p.202).

Definition at line 362 of file jazzyk.hpp.

6.2.4.13 `template<class TConcreteKRModule> void jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::map2raw (const TKRVarSubstitution & mapSubstitution, char **& arrVarID, char **& arrVarValue, unsigned int & nSize) [inline, private]`

Converts the variable substitution in the form of STL container into raw data.

The output arrays have to be correctly de-allocated using `free_resource` after processing! New resources are allocated here.

Definition at line 536 of file `jazzyk.hpp`.

Referenced by `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::query()`.

6.2.4.14 `template<class TConcreteKRModule> void jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::raw2map (const char ** arrVarID, const char ** arrVarValue, const unsigned int nSize, jazzyk::TKRVarSubstitution & mapSubstitution) [inline, private]`

Converts the variable substitution in the raw form into an STL container.

Definition at line 557 of file `jazzyk.hpp`.

Referenced by `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::query()`, and `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::update()`.

6.2.5 Member Data Documentation

6.2.5.1 `template<class TConcreteKRModule> TConcreteKRModule jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_concreteKRModule [private]`

Instance of the adaptee class.

All the calls of the interface are forwarded to this adaptee instance as callbacks.

Definition at line 189 of file `jazzyk.hpp`.

Referenced by `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::cycle()`, `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::finalize()`, and `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::initialize()`.

6.2.5.2 `template<class TConcreteKRModule> TQueryRegister jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::m_QueryOperations [private]`

Register of query operations.

Definition at line 192 of file `jazzyk.hpp`.

Referenced by `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::getQueryInterface()`, `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::query()`, and `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::registerQuery()`.

6.2.5.3 `template<class TConcreteKRModule> TUpdateRegister
jazzyk::AbstractKRModuleInterface< TConcreteKRModule
>::m_UpdateOperations [private]`

Register of update operations.

Definition at line 195 of file jazzyk.hpp.

Referenced by `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::getUpdateInterface()`, `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::registerUpdate()`, and `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::update()`.

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

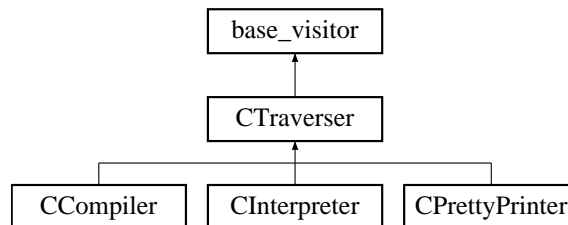
- **jazzyk.hpp**

6.3 base_visitor Class Reference

Visitor pattern library.

```
#include <visitor.hpp>
```

Inheritance diagram for base_visitor::



Public Member Functions

- `virtual ~base_visitor ()`
Empty virtual destructor.

6.3.1 Detailed Description

Visitor pattern library.

Visitor pattern implementation according to Andrei Alexandrescu: Modern C++ Design.

The implemented Visitor is a generic implementation of the Visitor pattern using `dynamic_cast` for dispatch. Base **visitor** (p.218) class

Definition at line 48 of file `visitor.hpp`.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 `virtual base_visitor::~~base_visitor ()` [inline, virtual]

Empty virtual destructor.

Only types with at least one virtual method support `dynamic_cast` for RTTI.

Definition at line 56 of file `visitor.hpp`.

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

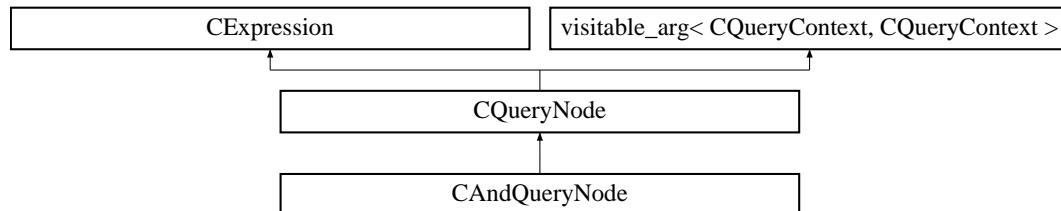
- `visitor.hpp`

6.4 CAndQueryNode Class Reference

Query node for AND logical connector of queries.

```
#include <AndQueryNode.hpp>
```

Inheritance diagram for CAndQueryNode::



Public Member Functions

- **CAndQueryNode** (TPtrQueryNode pLeftNode, TPtrQueryNode pRightNode)
Default constructor.
- virtual **~CAndQueryNode** ()
Virtual destructor.
- **TPtrQueryNode** getLeftNode () const
Returns the left node.
- **TPtrQueryNode** getRightNode () const
Returns rthe right subnode.

Private Member Functions

- **CAndQueryNode** (const CAndQueryNode &)
No copying defined.
- **CAndQueryNode & operator=** (const CAndQueryNode &)
No assignment defined.

Private Attributes

- **TPtrQueryNode** m_pLeftNode
Left hand query node.
- **TPtrQueryNode** m_pRightNode
Right hand query node.

6.4.1 Detailed Description

Query node for AND logical connector of queries.

Holds two query subnodes.

The node evaluates to TRUE when both subnodes evaluate to true as well. The resulting substitution is the combination of the substitution set of the right and left subnodes.

Definition at line 46 of file AndQueryNode.hpp.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 CAndQueryNode::CAndQueryNode (TPtrQueryNode *pLeftNode*, TPtrQueryNode *pRightNode*) [inline]

Default constructor.

Definition at line 102 of file AndQueryNode.hpp.

6.4.2.2 CAndQueryNode::~~CAndQueryNode () [inline, virtual]

Virtual destructor.

Definition at line 109 of file AndQueryNode.hpp.

6.4.2.3 CAndQueryNode::CAndQueryNode (const CAndQueryNode &) [private]

No copying defined.

6.4.3 Member Function Documentation

6.4.3.1 TPtrQueryNode CAndQueryNode::getLeftNode () const [inline]

Returns the left node.

Definition at line 114 of file AndQueryNode.hpp.

References m_pLeftNode.

Referenced by CTraverser::Visit().

6.4.3.2 TPtrQueryNode CAndQueryNode::getRightNode () const [inline]

Returns rthe right subnode.

Definition at line 120 of file AndQueryNode.hpp.

References m_pRightNode.

Referenced by CTraverser::Visit().

6.4.3.3 CAndQueryNode& CAndQueryNode::operator= (const CAndQueryNode &) [private]

No assignment defined.

6.4.4 Member Data Documentation

6.4.4.1 TPtrQueryNode CAndQueryNode::m_pLeftNode [private]

Left hand query node.

Definition at line 79 of file AndQueryNode.hpp.

Referenced by getLeftNode().

6.4.4.2 TPtrQueryNode CAndQueryNode::m_pRightNode [private]

Right hand query node.

Definition at line 82 of file AndQueryNode.hpp.

Referenced by getRightNode().

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

- **AndQueryNode.hpp**

6.5 CCodeBlock Class Reference

Code of block holder.

```
#include <CodeBlock.hpp>
```

Public Member Functions

- **CCodeBlock** ()
Default constructor.
- **CCodeBlock** (const char *szText)
Constructor from a plain string.
- virtual **~CCodeBlock** ()
Virtual destructor.
- **operator const char *** ()
*Automatic conversion to char *.*
- void **append** (const char *szText)
Append a code chunk.
- **CCodeBlock & operator+=** (char *szText)
Append operator.

Private Member Functions

- **CCodeBlock** (const **CCodeBlock** &)
Strictly "no copying".
- **CCodeBlock & operator=** (const **CCodeBlock** &)
Strict "no copying".

Private Attributes

- std::string **m_szCode**
Code block container.

6.5.1 Detailed Description

Code of block holder.

Holds a block of code, which is either part of query, or update expression.

One has to be carefull, so that this object will never copy, because it can possibly contain a large portion of data.

Definition at line 46 of file CodeBlock.hpp.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 CCodeBlock::CCodeBlock ()

Default constructor.

Definition at line 39 of file CodeBlock.cpp.

6.5.2.2 CCodeBlock::CCodeBlock (const char * *szText*)

Constructor from a plain string.

Definition at line 45 of file CodeBlock.cpp.

6.5.2.3 CCodeBlock::~~CCodeBlock () [virtual]

Virtual destructor.

Definition at line 51 of file CodeBlock.cpp.

6.5.2.4 CCodeBlock::CCodeBlock (const CCodeBlock &) [private]

Strictly "no copying".

6.5.3 Member Function Documentation

6.5.3.1 CCodeBlock::operator const char * () [inline]

Automatic conversion to char *.

Definition at line 114 of file CodeBlock.hpp.

References `m_szCode`.

6.5.3.2 void CCodeBlock::append (const char * *szText*) [inline]

Append a code chunk.

Definition at line 100 of file CodeBlock.hpp.

References `m_szCode`.

Referenced by `operator+=()`.

6.5.3.3 CCodeBlock & CCodeBlock::operator+= (char * *szText*) [inline]

Append operator.

Definition at line 106 of file CodeBlock.hpp.

References `append()`.

6.5.3.4 CCodeBlock& CCodeBlock::operator= (const CCodeBlock &) [private]

Strict "no copying".

6.5.4 Member Data Documentation

6.5.4.1 std::string CCodeBlock::m_szCode [private]

Code block container.

Definition at line 80 of file CodeBlock.hpp.

Referenced by append(), and operator const char *().

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

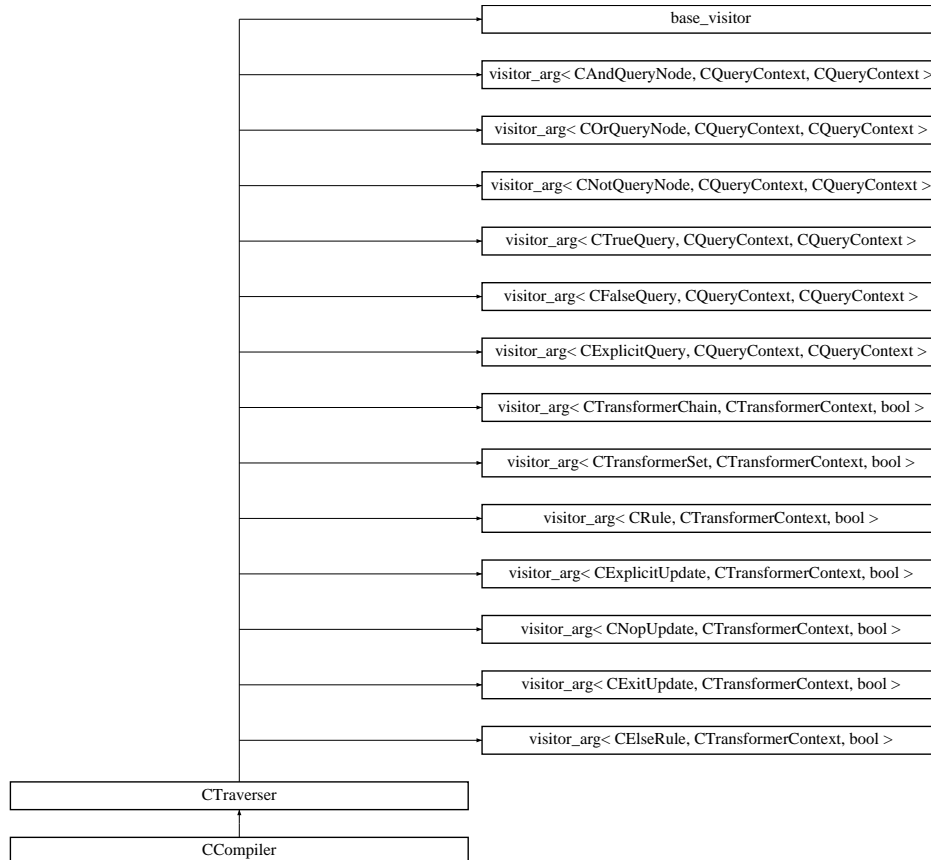
- **CodeBlock.hpp**
- **CodeBlock.cpp**

6.6 CCompiler Class Reference

Jazzyk programming language compiler.

```
#include <Compiler.hpp>
```

Inheritance diagram for CCompiler::



Public Member Functions

- **CCompiler** (TPtrProgram pProgram)
Default constructor.
- virtual **~CCompiler** ()
Virtual destructor.
- void **setIncludePaths** (const std::vector< std::string > &vszPaths)
Adds a new path to the list of include paths.
- virtual void **preprocess** (std::istream &inputStream, std::stringstream &preprocessed-Stream)
Runs the macro preprocessor on the input stream.

- virtual void **preprocess** (const std::string &szFileName, std::stringstream &preprocessed-Stream)
Runs the macro preprocessor on the input file.
- virtual void **parse** (std::istream &inputStream=std::cin)
Parses the input from the input stream.
- virtual void **compile** ()
Validates and compiles the parse tree to an executable program tree.
- virtual **TPtrProgram** **getProgram** () const
Returns the pointer to the program parse tree stored.

Protected Member Functions

- virtual **CQueryContext** & **PreAction** (**CExplicitQuery** &, **CQueryContext** &)
*Visitor/Traverser methods for the **CExpression** (p. 62) hierarchy.*
- virtual void **PreAction** (**CExplicitUpdate** &, **CTransformerContext** &)

Private Types

- enum { **PIPE_READ_END** = 0, **PIPE_WRITE_END** = 1 }
Constants for pipe ends.

Private Member Functions

- void **preprocess** (**SMPInvocationSettings** &settings, std::istream *inStream, std::stringstream &outStream)
Internal wrapper for launching the macro preprocessor.
- void **runChildProcess** (const **SMPInvocationSettings** &settings)
Wrapper for the forked child process.
- void **runParentProcess** (const **SMPInvocationSettings** &settings, std::istream *inStream, std::stringstream &outStream)
Wrapper for the parent process handler.
- void **readPipe** (const int fd, std::stringstream &outStream)
Reads a char stream from file descriptor to a stringstream.
- void **fillIncludePaths** (**SMPInvocationSettings** &settings)
Fills include paths into the macro preprocessor invocation settings.
- **CCompiler** (const **CCompiler** &)
No copying defined.

- **CCompiler & operator=** (const CCompiler &)
No assignment defined.

Private Attributes

- **TPtrProgram m_pProgram**
Pointer to the parsed program.
- **std::vector< std::string > m_vsIncludePaths**
Macro preprocessor include paths.

Classes

- **struct SMPInvocationSettings**
Configuration structure for the macro preprocessor invocation.

6.6.1 Detailed Description

Jazzyk programming language compiler.

Provides a wrapper for parser and compiles the input source code into a interpretable structure.

Parser calls the bison/flex generated parser. The result is stored in **CProgram** (p. 125) parse tree structure. Compiler then provides interfaces to compile and manipulate the structure.

In order to compile/validate the parsed program, the **CCompiler** (p. 33) visits the whole **CExpression** (p. 62) hierarchy, hence it is a special kind of the program tree traverser.

Definition at line 60 of file Compiler.hpp.

6.6.2 Member Enumeration Documentation

6.6.2.1 anonymous enum [private]

Constants for pipe ends.

Enumerator:

PIPE_READ_END
PIPE_WRITE_END

Definition at line 194 of file Compiler.hpp.

6.6.3 Constructor & Destructor Documentation

6.6.3.1 CCompiler::CCompiler (TPtrProgram *pProgram*)

Default constructor.

Note: Compiler does not take the ownership of the pProgram.

Definition at line 89 of file Compiler.cpp.

6.6.3.2 CCompiler::~~CCompiler () [virtual]

Virtual destructor.

Definition at line 95 of file Compiler.cpp.

6.6.3.3 CCompiler::CCompiler (const CCompiler &) [private]

No copying defined.

6.6.4 Member Function Documentation

6.6.4.1 void CCompiler::setIncludePaths (const std::vector< std::string > & vszPaths)

Adds a new path to the list of include paths.

These paths are passed further to the internal macro preprocessor invocation via -I option.

Definition at line 121 of file Compiler.cpp.

References m_vszIncludePaths.

Referenced by CJazzykApp::run().

6.6.4.2 void CCompiler::preprocess (std::istream & inputStream, std::stringstream & preprocessedStream) [virtual]

Runs the macro preprocessor on the input stream.

Feeds the inputStream content into GNU M4 macro preprocessor and returns the resulting output stream which can be used later as an input stream for other compiler stages.

Definition at line 274 of file Compiler.cpp.

Referenced by preprocess(), and CJazzykApp::run().

6.6.4.3 void CCompiler::preprocess (const std::string & szFileName, std::stringstream & preprocessedStream) [virtual]

Runs the macro preprocessor on the input file.

The GNU M4 macro preprocessor is here responsible for opening, reading and preprocessing the input file. The output stream is returned.

Note: The macro preprocessor respects only its own include markers in the input file. It means that it changes the internal line counters only when it really performs an include operation. It ignores markers added to the input file before and treats them as a plain text. This would lead to incorrect error reporting from within the GNU M4 preprocessor. Therefore it's better to let the preprocessor completely read and process the files itself, because then the error output from it is usefull and truthfull in the error reporting of the Jazzyk interpreter.

Definition at line 287 of file Compiler.cpp.

References CCompiler::SMPInvocationSettings::argy, and preprocess().

6.6.4.4 void CCompiler::parse (std::istream & *inputStream* = std::cin) [virtual]

Parses the input from the input stream.

In the case of an error occuring during parsing, method throws an **CError** (p. 47) exception.

Definition at line 100 of file Compiler.cpp.

References CError::JZPARSER_PARSE_ERROR, and m_pProgram.

Referenced by CJazzykApp::run().

6.6.4.5 void CCompiler::compile () [virtual]

Validates and compiles the parse tree to an executable program tree.

Program is valid when all the 'called' mental state transformers were also defined and all the references to modules link to a declared module as well. Finally, free variables used in explicit query/update expressions should be already defined sometime above in the parse tree in query expressions.

In the case the program is invalid, **compile()** (p. 37) throws an exception **CError** (p. 47).

Definition at line 115 of file Compiler.cpp.

References m_pProgram, and CTraverser::traverse().

Referenced by CJazzykApp::run().

6.6.4.6 TPtrProgram CCompiler::getProgram () const [virtual]

Returns the pointer to the program parse tree stored.

Definition at line 338 of file Compiler.cpp.

References m_pProgram.

6.6.4.7 CQueryContext & CCompiler::PreAction (CExplicitQuery & *quest*, CQueryContext & *ctx*) [protected, virtual]

Visitor/Traverser methods for the **CExpression** (p. 62) hierarchy.

CCompiler (p. 33) visits **CExpression** (p. 62) hierarchy in order to examine the program for validity.

For details on validity of a program see comments by compile(...) method.

PreAction() (p. 37) methods of traverser specialisation

Reimplemented from **CTraverser** (p. 191).

Definition at line 344 of file Compiler.cpp.

References CTransformerContext::getModuleDeclaration(), CExplicitQuery::getModuleID(), CExplicitQuery::getOperation(), CExpression::getSrcPosition(), CTransformerContext::isModuleDeclared(), CError::JZCOMPILER_VALIDITY_INVALID_OPERATION, and CError::JZCOMPILER_VALIDITY_MODULE_UNDECLARED.

6.6.4.8 void CCompiler::PreAction (CExplicitUpdate & *guest*, CTransformerContext & *ctx*) [protected, virtual]

Reimplemented from CTraverser (p.192).

Definition at line 373 of file Compiler.cpp.

References CTransformerContext::getModuleDeclaration(), CExplicitUpdate::getModuleID(), CExplicitUpdate::getOperation(), CExpression::getSrcPosition(), CExplicitUpdate::getUsedVariables(), CTransformerContext::isModuleDeclared(), CError::JZCOMPILER_VALIDITY_INVALID_OPERATION, CError::JZCOMPILER_VALIDITY_MODULE_UNDECLARED, and CError::JZCOMPILER_VARIABLE_UNDECLARED.

6.6.4.9 void CCompiler::preprocess (SMPInvocationSettings & *settings*, std::istream * *inStream*, std::stringstream & *outStream*) [private]

Internal wrapper for launching the macro preprocessor.

According to provided settings, it forks a child process and invokes child and parent process handlers.

In the case *inStream* is non-zero, it's contents are fed to the child process stdin stream.

Definition at line 299 of file Compiler.cpp.

References CCompiler::SMPInvocationSettings::child_pid, CCompiler::SMPInvocationSettings::child_stderr, CCompiler::SMPInvocationSettings::child_stdin, CCompiler::SMPInvocationSettings::child_stdout, fillIncludePaths(), CError::JZMPREPROCESSOR_ERROR, runChildProcess(), and runParentProcess().

6.6.4.10 void CCompiler::runChildProcess (const SMPInvocationSettings & *settings*) [private]

Wrapper for the forked child process.

Prepares the GNU M4 invocation and calls exec(). Also redirects the stdin/stdout/stderr streams to the parent process.

Definition at line 160 of file Compiler.cpp.

References CCompiler::SMPInvocationSettings::argv, CCompiler::SMPInvocationSettings::child_stderr, CCompiler::SMPInvocationSettings::child_stdin, CCompiler::SMPInvocationSettings::child_stdout, CCompiler::SMPInvocationSettings::cmd, CError::JZMPREPROCESSOR_ERROR, PIPE_READ_END, and PIPE_WRITE_END.

Referenced by preprocess().

6.6.4.11 void CCompiler::runParentProcess (const SMPInvocationSettings & *settings*, std::istream * *inStream*, std::stringstream & *outStream*) [private]

Wrapper for the parent process handler.

Wrapper for communication with the parent process running the macro preprocessor. By default it feeds *inStream* to the stdin stream of the child process and captures its stdout stream. In the case of any error, it throws a CError (p.47) exception.

In the case `inStream==0`, it does not feed any input to the child process and only captures its output.

Definition at line 220 of file `Compiler.cpp`.

References `CCompiler::SMPInvocationSettings::child_pid`, `CCompiler::SMPInvocationSettings::child_stderr`, `CCompiler::SMPInvocationSettings::child_stdin`, `CCompiler::SMPInvocationSettings::child_stdout`, `CError::JZMPREPROCESSOR_ERROR`, `PIPE_READ_END`, `PIPE_WRITE_END`, and `readPipe()`.

Referenced by `preprocess()`.

6.6.4.12 `void CCompiler::readPipe (const int fd, std::stringstream & outStream) [private]`

Reads a char stream from file descriptor to a stringstream.

Definition at line 139 of file `Compiler.cpp`.

References `CError::JZMPREPROCESSOR_ERROR`.

Referenced by `runParentProcess()`.

6.6.4.13 `void CCompiler::fillIncludePaths (CCompiler::SMPInvocationSettings & settings) [private]`

Fills include paths into the macro preprocessor invocation settings.

Definition at line 127 of file `Compiler.cpp`.

References `CCompiler::SMPInvocationSettings::argv`, and `m_vszIncludePaths`.

Referenced by `preprocess()`.

6.6.4.14 `CCompiler& CCompiler::operator= (const CCompiler &) [private]`

No assignment defined.

6.6.5 Member Data Documentation

6.6.5.1 `TPtrProgram CCompiler::m_pProgram [private]`

Pointer to the parsed program.

Definition at line 156 of file `Compiler.hpp`.

Referenced by `compile()`, `getProgram()`, and `parse()`.

6.6.5.2 `std::vector<std::string> CCompiler::m_vszIncludePaths [private]`

Macro preprocessor include paths.

Definition at line 159 of file `Compiler.hpp`.

Referenced by `fillIncludePaths()`, and `setIncludePaths()`.

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

- `Compiler.hpp`
- `Compiler.cpp`

6.7 CCompiler::SMPInvocationSettings Struct Reference

Configuration structure for the macro preprocessor invocation.

Public Member Functions

- **SMPInvocationSettings** ()
Default constructor.
- void **configure** (const std::string &szCmd, const std::vector< std::string > &vArgv)
Reconfiguration of the macro preprocessor call.

Public Attributes

- int **child_stdin** [2]
Pipe file descriptors for stdin, stdout and stderr.
- int **child_stdout** [2]
- int **child_stderr** [2]
- pid_t **child_pid**
Child process PID.
- std::string **cmd**
Macro preprocessor command.
- std::vector< std::string > **argv**
Macro preprocessor command line arguments.

6.7.1 Detailed Description

Configuration structure for the macro preprocessor invocation.

Definition at line 162 of file Compiler.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 CCompiler::SMPInvocationSettings::SMPInvocationSettings ()

Default constructor.

Definition at line 60 of file Compiler.cpp.

References argv.

6.7.3 Member Function Documentation

6.7.3.1 `void CCompiler::SMPInvocationSettings::configure (const std::string & szCmd, const std::vector< std::string > & vArgv)`

Reconfiguration of the macro preprocessor call.

Allows to implement invocation of other than default GNU M4 macro preprocessor.

Definition at line 79 of file Compiler.cpp.

References `argv`, and `cmd`.

6.7.4 Member Data Documentation

6.7.4.1 `int CCompiler::SMPInvocationSettings::child_stdin[2]`

Pipe file descriptors for stdin, stdout and stderr.

Definition at line 175 of file Compiler.hpp.

Referenced by `CCompiler::preprocess()`, `CCompiler::runChildProcess()`, and `CCompiler::runParentProcess()`.

6.7.4.2 `int CCompiler::SMPInvocationSettings::child_stdout[2]`

Definition at line 176 of file Compiler.hpp.

Referenced by `CCompiler::preprocess()`, `CCompiler::runChildProcess()`, and `CCompiler::runParentProcess()`.

6.7.4.3 `int CCompiler::SMPInvocationSettings::child_stderr[2]`

Definition at line 177 of file Compiler.hpp.

Referenced by `CCompiler::preprocess()`, `CCompiler::runChildProcess()`, and `CCompiler::runParentProcess()`.

6.7.4.4 `pid_t CCompiler::SMPInvocationSettings::child_pid`

Child process PID.

Definition at line 180 of file Compiler.hpp.

Referenced by `CCompiler::preprocess()`, and `CCompiler::runParentProcess()`.

6.7.4.5 `std::string CCompiler::SMPInvocationSettings::cmd`

Macro preprocessor command.

By default we invoke GNU M4.

Definition at line 186 of file Compiler.hpp.

Referenced by `configure()`, and `CCompiler::runChildProcess()`.

6.7.4.6 `std::vector<std::string> CCompiler::SMPInvocationSettings::argv`

Macro preprocessor command line arguments.

Definition at line 189 of file `Compiler.hpp`.

Referenced by `configure()`, `CCompiler::fillIncludePaths()`, `CCompiler::preprocess()`, `CCompiler::runChildProcess()`, and `SMPInvocationSettings()`.

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

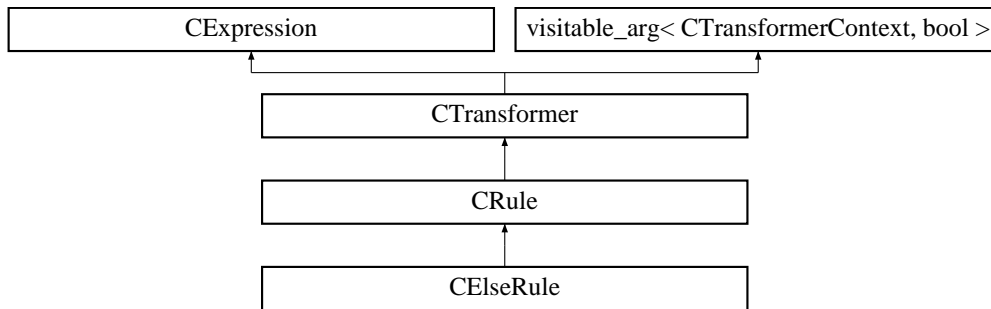
- `Compiler.hpp`
- `Compiler.cpp`

6.8 CElseRule Class Reference

If-then-else type rule.

```
#include <ElseRule.hpp>
```

Inheritance diagram for CElseRule::



Public Member Functions

- **CElseRule** (TPtrQueryNode pLHS, TPtrTransformer pThenRHS, TPtrTransformer pElseRHS)
Default constructor.
- virtual **~CElseRule** ()
Virtual destructor.
- TPtrTransformer **getElseTransformer** () const
Returns the second right hand side transformer.

Private Member Functions

- **CElseRule** (const CElseRule &)
No copying defined.
- **CElseRule & operator=** (const CElseRule &)
No assignment defined.

Private Attributes

- TPtrTransformer **m_pElseTransformer**
Else right hand side transformer.

6.8.1 Detailed Description

If-then-else type rule.

CElseRule (p. 44) is a special type of **CRule** (p. 138) with two right hand sides. If the query evaluates to **FALSE**, the second right hand mental state transformer is executed.

Definition at line 45 of file ElseRule.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 CElseRule::CElseRule (TPtrQueryNode *pLHS*, TPtrTransformer *pThenRHS*, TPtrTransformer *pElseRHS*) [inline]

Default constructor.

Definition at line 93 of file ElseRule.hpp.

6.8.2.2 CElseRule::~~CElseRule () [inline, virtual]

Virtual destructor.

Definition at line 100 of file ElseRule.hpp.

6.8.2.3 CElseRule::CElseRule (const CElseRule &) [private]

No copying defined.

6.8.3 Member Function Documentation

6.8.3.1 TPtrTransformer CElseRule::getElseTransformer () const [inline]

Returns the second right hand side transformer.

Definition at line 105 of file ElseRule.hpp.

References `m_pElseTransformer`.

Referenced by `CTraverser::Visit()`.

6.8.3.2 CElseRule& CElseRule::operator= (const CElseRule &) [private]

No assignment defined.

6.8.4 Member Data Documentation

6.8.4.1 TPtrTransformer CElseRule::m_pElseTransformer [private]

Else right hand side transformer.

Definition at line 74 of file ElseRule.hpp.

Referenced by `getElseTransformer()`.

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

- `ElseRule.hpp`

6.9 CError Class Reference

Error handler for the application.

```
#include <Error.hpp>
```

Inherits std::exception.

Public Types

- enum **EErrorType** {
JZNOERROR = 0, **JZPARSER_PARSE_ERROR** = 1, **JZCOMPILER_VALIDITY_MODULE_REDEFINED** = 2, **JZCOMPILER_VALIDITY_TRANSFORMER_REDEFINED** = 3,
JZCOMPILER_VALIDITY_TRANSFORMER_UNDEFINED = 4, **JZCOMPILER_VALIDITY_MODULE_UNDECLARED** = 5,
JZCOMPILER_VALIDITY_INVALID_OPERATION = 6, **JZCOMPILER_VARIABLE_UNDECLARED** = 7,
JZINTERPRETER_VARIABLE_REWRITE = 8, **JZMODULE_ERROR** = 9,
JZLIBTOOL_ERROR = 10, **JZMPREPROCESSOR_ERROR** = 11,
JZOTHER_ERROR = 12, **JZUNKNOWN_ERROR** = 13 }

Enumeration of possible error types.

- typedef yy::location **TSrcLocation**
Type to be used as a position marker of errors.

Public Member Functions

- **CError** (const **EErrorType** eErrType, const std::string &szReason)
Default constructor.
- **CError** (const **TSrcLocation** &pos, const **EErrorType** eErrType, const std::string &szReason)
Constructor with error position.
- virtual **~CError** () throw ()
Copy constructor.
- void **report** () const
Dumps the error to the output.
- std::string & **resolveErrorType** () const
Resolves the error type code to a reportable string.
- virtual const char * **what** () const throw ()
Virtual overwrite of the parent reporting method.

Public Attributes

- **TSrcLocation m_location**
Error position.
- **EErrorType m_errorType**
Error type.
- **std::string m_szReason**
Error reason.
- **std::string m_szErrMsg**
Full error message holder.

Static Public Attributes

- **static std::string m_dictErrMsgs []**
Dictionary of error messages.

Private Member Functions

- **CError & operator= (const CError &)**
No assignment defined.

6.9.1 Detailed Description

Error handler for the application.

Centralized error handler. All errors are directed to this class. Exceptions throw also objects of **CError** (p. 47) class.

An error holds an error type, reason as a string and an error source code position.

Error messages are held in a dictionary which has to be initialized before use of the error handler.

CError (p. 47) is also used as an exception which is thrown during program/parse tree parsing, validation, interpretation etc.

Definition at line 57 of file Error.hpp.

6.9.2 Member Typedef Documentation

6.9.2.1 typedef yy::location CError::TSrcLocation

Type to be used as a position marker of errors.

Definition at line 84 of file Error.hpp.

6.9.3 Member Enumeration Documentation

6.9.3.1 enum CError::EErrorType

Enumeration of possible error types.

Enumerator:

JZNOERROR
JZPARSER_PARSE_ERROR
JZCOMPILER_VALIDITY_MODULE_REDEFINED
JZCOMPILER_VALIDITY_TRANSFORMER_REDEFINED
JZCOMPILER_VALIDITY_TRANSFORMER_UNDEFINED
JZCOMPILER_VALIDITY_MODULE_UNDECLARED
JZCOMPILER_VALIDITY_INVALID_OPERATION
JZCOMPILER_VARIABLE_UNDECLARED
JZINTERPRETER_VARIABLE_REWRITE
JZMODULE_ERROR
JZLIBTOOL_ERROR
JZMPREPROCESSOR_EROR
JZOTHER_ERROR
JZUNKNOWN_ERROR

Definition at line 64 of file Error.hpp.

6.9.4 Constructor & Destructor Documentation

6.9.4.1 CError::CError (const EErrorType *eErrType*, const std::string & *szReason*)

Default constructor.

Definition at line 61 of file Error.cpp.

References `m_location`, `m_szErrText`, `m_szReason`, and `resolveErrorType()`.

6.9.4.2 CError::CError (const TSrcLocation & *pos*, const EErrorType *eErrType*, const std::string & *szReason*)

Constructor with error position.

Definition at line 81 of file Error.cpp.

References `m_location`, `m_szErrText`, `m_szReason`, and `resolveErrorType()`.

6.9.4.3 CError::~CError () throw () [virtual]

Copy constructor.

Left for the compiler to create a default copy constructor. Virtual destructor

Definition at line 102 of file Error.cpp.

6.9.5 Member Function Documentation

6.9.5.1 void CError::report () const

Dumps the error to the output.

Definition at line 107 of file Error.cpp.

6.9.5.2 std::string & CError::resolveErrorType () const

Resolves the error type code to a reportable string.

Definition at line 113 of file Error.cpp.

References JZUNKNOWN_ERROR, m_dictErrorMsgs, and m_errorType.

Referenced by CError().

6.9.5.3 const char * CError::what () const throw () [virtual]

Virtual overwrite of the parent reporting method.

Definition at line 128 of file Error.cpp.

References m_szErrMsg.

Referenced by CJazzykApp::run().

6.9.5.4 CError& CError::operator= (const CError &) [private]

No assignment defined.

6.9.6 Member Data Documentation

6.9.6.1 TSrcLocation CError::m_location

Error position.

Position where the error occurred.

We are using the standard Bison C++ yy::location class for tracking location.

Definition at line 132 of file Error.hpp.

Referenced by CError().

6.9.6.2 EErrorType CError::m_errorType

Error type.

Definition at line 135 of file Error.hpp.

Referenced by CInterpreter::PostAction(), and resolveErrorType().

6.9.6.3 std::string CError::m_szReason

Error reason.

Definition at line 138 of file Error.hpp.

Referenced by CError(), and CInterpreter::PostAction().

6.9.6.4 std::string CError::m_dictErrorMsgs [static]

Initial value:

```
{
    "ok",
    "parse error",
    "module declared twice",
    "transformer declared twice",
    "undefined transformer identifier",
    "referenced module undeclared",
    "invalid query/update module interface operation attempted",
    "variable used, but not instantiated before",

    "variable substitution inconsistency - value overwritten",
    "module error",
    "libtool error",
    "macro preprocessor error",

    "error",
    "unknown error"
}
```

Dictionary of error messages.

Definition at line 141 of file Error.hpp.

Referenced by resolveErrorType().

6.9.6.5 std::string CError::m_szErrMsg

Full error message holder.

Definition at line 144 of file Error.hpp.

Referenced by CError(), operator<<(), and what().

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

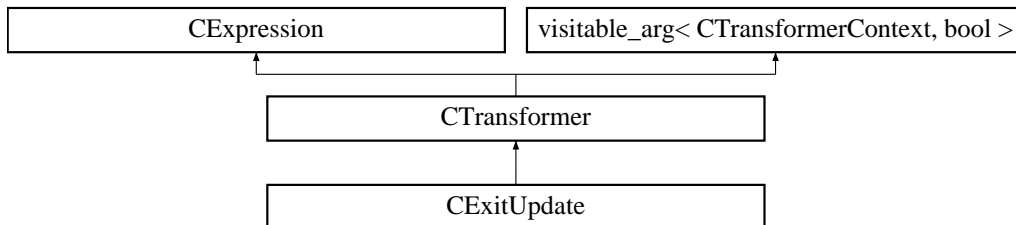
- **Error.hpp**
- **Error.cpp**

6.10 CExitUpdate Class Reference

Exit command mental state transformer implementation.

```
#include <TrivialUpdate.hpp>
```

Inheritance diagram for CExitUpdate::



Public Member Functions

- **CExitUpdate ()**
Default constructor.
- **virtual ~CExitUpdate ()**
Virtual destructor.

Private Member Functions

- **CExitUpdate (const CExitUpdate &)**
No copying defined.
- **CExitUpdate & operator= (const CExitUpdate &)**
No assignment defined.

6.10.1 Detailed Description

Exit command mental state transformer implementation.

Upon executing the exit command, the program should stop and terminate correctly.

Definition at line 82 of file TrivialUpdate.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 CExitUpdate::CExitUpdate () [inline]

Default constructor.

Definition at line 140 of file TrivialUpdate.hpp.

6.10.2.2 CExitUpdate::~~CExitUpdate () [inline, virtual]

Virtual destructor.

Definition at line 145 of file TrivialUpdate.hpp.

6.10.2.3 CExitUpdate::CExitUpdate (const CExitUpdate &) [private]

No copying defined.

6.10.3 Member Function Documentation

6.10.3.1 CExitUpdate& CExitUpdate::operator= (const CExitUpdate &) [private]

No assignment defined.

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

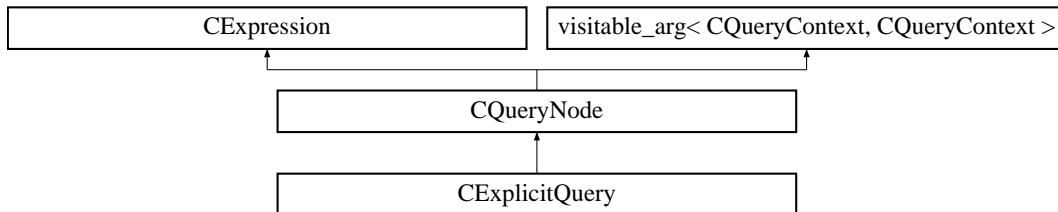
- **TrivialUpdate.hpp**

6.11 CExplicitQuery Class Reference

Explicit query class.

```
#include <ExplicitQuery.hpp>
```

Inheritance diagram for CExplicitQuery::



Public Member Functions

- **CExplicitQuery** (const TPtrIdentifier &pModuleID, const TPtrIdentifier &pOperation, const TPtrIdentifiers &pVariables, const TPtrCodeBlock &pCodeBlock)
Default constructor.
- virtual **~CExplicitQuery** ()
Virtual destructor.
- const TPtrIdentifier & **getModuleID** () const
Returns the module identifier.
- const TPtrIdentifier & **getOperation** () const
Returns the pointer to the operation identifier.
- const TPtrIdentifiers & **getFreeVariables** () const
Returns the pointer to free variables container.
- const TPtrCodeBlock & **getCodeBlock** () const
Returns the pointer to code block.

Private Member Functions

- **CExplicitQuery** (const CExplicitQuery &)
No copying defined.
- **CExplicitQuery** & **operator=** (const CExplicitQuery &)
No assignment defined.

Private Attributes

- TPtrIdentifier **m_pModuleID**
Associated module identifier.
- TPtrIdentifier **m_pOperation**
Query operation identifier.
- TPtrIdentifiers **m_pVariables**
List of free variables.
- TPtrCodeBlock **m_pCodeBlock**
Query code block.

6.11.1 Detailed Description

Explicit query class.

Explicit query is so to say "normal" query comprising of the full code block and a link to an associated external module which is being queries by it.

Definition at line 45 of file ExplicitQuery.hpp.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 CExplicitQuery::CExplicitQuery (const TPtrIdentifier & *pModuleID*, const TPtrIdentifier & *pOperation*, const TPtrIdentifiers & *pVariables*, const TPtrCodeBlock & *pCodeBlock*) [inline]

Default constructor.

Definition at line 115 of file ExplicitQuery.hpp.

6.11.2.2 CExplicitQuery::~CExplicitQuery () [inline, virtual]

Virtual destructor.

Definition at line 128 of file ExplicitQuery.hpp.

6.11.2.3 CExplicitQuery::CExplicitQuery (const CExplicitQuery &) [private]

No copying defined.

6.11.3 Member Function Documentation

6.11.3.1 const TPtrIdentifier & CExplicitQuery::getModuleID () const [inline]

Returns the module identifier.

Definition at line 133 of file ExplicitQuery.hpp.

References `m_pModuleID`.

Referenced by `CInterpreter::PostAction()`, and `CCompiler::PreAction()`.

6.11.3.2 `const TPtrIdentifier & CExplicitQuery::getOperation () const [inline]`

Returns the pointer to the operation identifier.

Definition at line 139 of file `ExplicitQuery.hpp`.

References `m_pOperation`.

Referenced by `CInterpreter::PostAction()`, and `CCompiler::PreAction()`.

6.11.3.3 `const TPtrIdentifiers & CExplicitQuery::getFreeVariables () const [inline]`

Returns the pointer to free variables container.

Definition at line 145 of file `ExplicitQuery.hpp`.

References `m_pVariables`.

Referenced by `CInterpreter::PostAction()`, `CPrettyPrinter::PreAction()`, and `CTraverser::Visit()`.

6.11.3.4 `const TPtrCodeBlock & CExplicitQuery::getCodeBlock () const [inline]`

Returns the pointer to code block.

Definition at line 151 of file `ExplicitQuery.hpp`.

References `m_pCodeBlock`.

Referenced by `CInterpreter::PostAction()`.

6.11.3.5 `CExplicitQuery& CExplicitQuery::operator= (const CExplicitQuery &) [private]`

No assignment defined.

6.11.4 Member Data Documentation

6.11.4.1 `TPtrIdentifier CExplicitQuery::m_pModuleID [private]`

Associated module identifier.

Definition at line 87 of file `ExplicitQuery.hpp`.

Referenced by `getModuleID()`.

6.11.4.2 `TPtrIdentifier CExplicitQuery::m_pOperation [private]`

Query operation identifier.

Definition at line 90 of file `ExplicitQuery.hpp`.

Referenced by `getOperation()`.

6.11.4.3 TPtrIdentifiers CExplicitQuery::m_pVariables [private]

List of free variables.

Definition at line 93 of file ExplicitQuery.hpp.

Referenced by getFreeVariables().

6.11.4.4 TPtrCodeBlock CExplicitQuery::m_pCodeBlock [private]

Query code block.

Definition at line 96 of file ExplicitQuery.hpp.

Referenced by getCodeBlock().

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

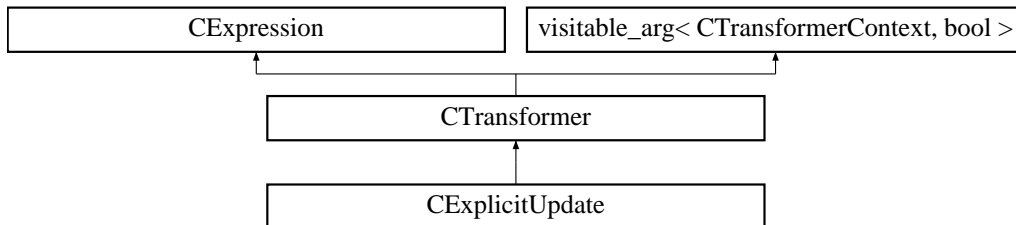
- **ExplicitQuery.hpp**

6.12 CExplicitUpdate Class Reference

Explicit update class.

```
#include <ExplicitUpdate.hpp>
```

Inheritance diagram for CExplicitUpdate::



Public Member Functions

- **CExplicitUpdate** (const TPtrIdentifier &pModuleID, const TPtrIdentifier &pOperation, const TPtrIdentifiers &variables, const TPtrCodeBlock &pCodeBlock)

Default constructor.

- virtual **~CExplicitUpdate** ()

Virtual destructor.

- const TPtrIdentifier & **getModuleID** () const

Returns the referred module ID.

- const TPtrIdentifier & **getOperation** () const

Returns the update operation ID.

- const TPtrIdentifiers & **getUsedVariables** () const

Returns the free variables list.

- const TPtrCodeBlock & **getCodeBlock** () const

Returnd the code block.

Private Member Functions

- **CExplicitUpdate** (const CExplicitUpdate &)

No copying defined.

- **CExplicitUpdate** & **operator=** (const CExplicitUpdate &)

No assignment defined.

Private Attributes

- TPtrIdentifier **m_pModuleID**
Associated module identifier.
- TPtrIdentifier **m_pOperation**
Update operation to invoke identifier.
- TPtrIdentifiers **m_pVariables**
List of free variables.
- TPtrCodeBlock **m_pCodeBlock**
Code block.

6.12.1 Detailed Description

Explicit update class.

Explicit update is so to say "normal" update comprising of the full code block and a link to an associated external module on which is it is executed.

Definition at line 46 of file ExplicitUpdate.hpp.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 CExplicitUpdate::CExplicitUpdate (const TPtrIdentifier & *pModuleID*, const TPtrIdentifier & *pOperation*, const TPtrIdentifiers & *variables*, const TPtrCodeBlock & *pCodeBlock*) [inline]

Default constructor.

Definition at line 117 of file ExplicitUpdate.hpp.

6.12.2.2 CExplicitUpdate::~CExplicitUpdate () [inline, virtual]

Virtual destructor.

Definition at line 130 of file ExplicitUpdate.hpp.

6.12.2.3 CExplicitUpdate::CExplicitUpdate (const CExplicitUpdate &) [private]

No copying defined.

6.12.3 Member Function Documentation

6.12.3.1 const TPtrIdentifier & CExplicitUpdate::getModuleID () const [inline]

Returns the referred module ID.

Definition at line 135 of file ExplicitUpdate.hpp.

References `m_pModuleID`.

Referenced by `CInterpreter::PostAction()`, and `CCompiler::PreAction()`.

6.12.3.2 `const TPtrIdentifier & CExplicitUpdate::getOperation () const [inline]`

Returns the update operation ID.

Definition at line 141 of file `ExplicitUpdate.hpp`.

References `m_pOperation`.

Referenced by `CInterpreter::PostAction()`, and `CCompiler::PreAction()`.

6.12.3.3 `const TPtrIdentifiers & CExplicitUpdate::getUsedVariables () const [inline]`

Returns the free variables list.

Definition at line 147 of file `ExplicitUpdate.hpp`.

References `m_pVariables`.

Referenced by `CInterpreter::PostAction()`, `CPrettyPrinter::PreAction()`, and `CCompiler::PreAction()`.

6.12.3.4 `const TPtrCodeBlock & CExplicitUpdate::getCodeBlock () const [inline]`

Returnd the code block.

Definition at line 152 of file `ExplicitUpdate.hpp`.

References `m_pCodeBlock`.

Referenced by `CInterpreter::PostAction()`.

6.12.3.5 `CExplicitUpdate& CExplicitUpdate::operator= (const CExplicitUpdate &) [private]`

No assignment defined.

6.12.4 Member Data Documentation

6.12.4.1 `TPtrIdentifier CExplicitUpdate::m_pModuleID [private]`

Associated module identifier.

Definition at line 89 of file `ExplicitUpdate.hpp`.

Referenced by `getModuleID()`.

6.12.4.2 `TPtrIdentifier CExplicitUpdate::m_pOperation [private]`

Update operation to invoke identifier.

Definition at line 92 of file ExplicitUpdate.hpp.

Referenced by getOperation().

6.12.4.3 TPtrIdentifiers CExplicitUpdate::m_pVariables [private]

List of free variables.

Definition at line 95 of file ExplicitUpdate.hpp.

Referenced by getUsedVariables().

6.12.4.4 TPtrCodeBlock CExplicitUpdate::m_pCodeBlock [private]

Code block.

Definition at line 98 of file ExplicitUpdate.hpp.

Referenced by getCodeBlock().

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

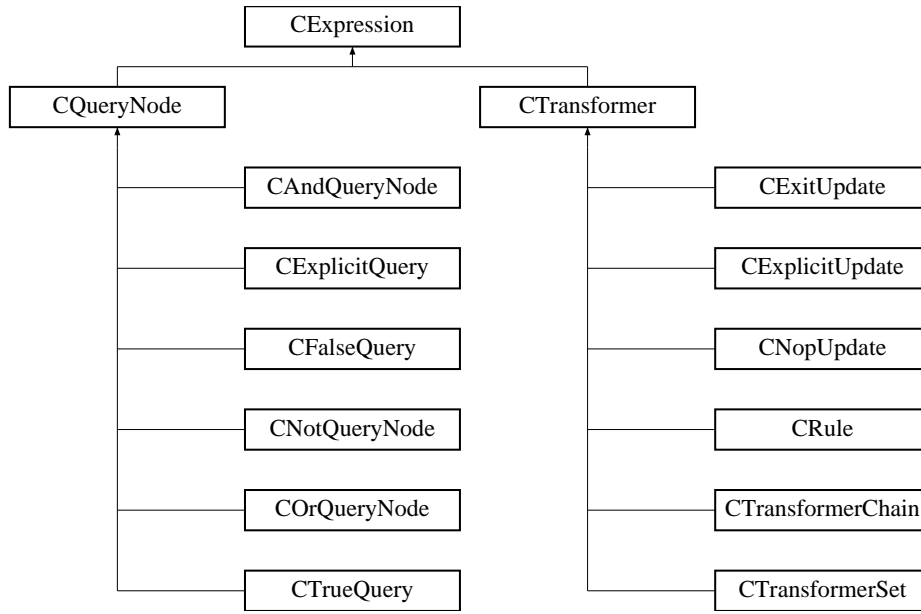
- **ExplicitUpdate.hpp**

6.13 CExpression Class Reference

Base abstract class for all the query and update expressions.

```
#include <Expression.hpp>
```

Inheritance diagram for CExpression::



Public Member Functions

- **CExpression ()**
Constructor.
- **virtual ~CExpression ()=0**
Virtual destructor.
- **const TSrcLocation & getSrcPosition () const**
Returns the source position where the expression was defined.

Private Member Functions

- **CExpression (const CExpression &)**
No copying defined.
- **CExpression & operator= (const CExpression &)**
No assignment defined.

Private Attributes

- **TSrcLocation m_srcLocation**

Source file definition position.

6.13.1 Detailed Description

Base abstract class for all the query and update expressions.

Abstract class for other types of query and update expressions.

All the manipulations on the **CExpression** (p. 62) hierarchy are done using a Visitor pattern. Hence **CExpression** (p. 62) class hierarchy is **visitable** (p. 214), all the **visitable** (p. 214) subclasses have to declare themselves to be such, i.e. use **DECLARE_VISITABLE()** (p. 300). Abstract subclasses do not have to declare their visitability. For details see the documentation of subclasses (**CTransformer** (p. 172) and **CQueryNode** (p. 136)).

To facilitate an elegant error reporting, each expressions holds a position where it was defined in the source file (stdin).

Note: In order to optimize the code, it might be useful in the future to redefine the whole **CExpression** (p. 62) hierarchy to struct, rather than classes and open the private members. Currently, most members of the hierarchy have a trivial implementation. This was achieved by the use of Visitor pattern.

Definition at line 62 of file Expression.hpp.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 CExpression::CExpression ()

Constructor.

Definition at line 41 of file Expression.cpp.

6.13.2.2 CExpression::~CExpression () [pure virtual]

Virtual destructor.

Definition at line 47 of file Expression.cpp.

6.13.2.3 CExpression::CExpression (const CExpression &) [private]

No copying defined.

6.13.3 Member Function Documentation

6.13.3.1 const TSrcLocation & CExpression::getSrcPosition () const [inline]

Returns the source position where the expression was defined.

Definition at line 108 of file Expression.hpp.

References `m_srcLocation`.

Referenced by CInterpreter::PostAction(), and CCompiler::PreAction().

6.13.3.2 CExpression& CExpression::operator= (const CExpression &) [private]

No assignment defined.

6.13.4 Member Data Documentation

6.13.4.1 TSrcLocation CExpression::m_srcLocation [private]

Source file definition position.

Definition at line 89 of file Expression.hpp.

Referenced by getSrcPosition().

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

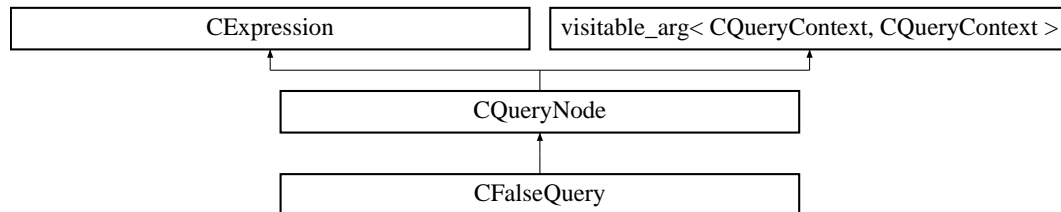
- **Expression.hpp**
- **Expression.cpp**

6.14 CFalseQuery Class Reference

Trivial False query.

```
#include <TrivialQuery.hpp>
```

Inheritance diagram for CFalseQuery::



Public Member Functions

- **CFalseQuery** ()
Default constructor.
- **virtual ~CFalseQuery** ()
Virtual destructor.

Private Member Functions

- **CFalseQuery** (const **CFalseQuery** &)
No copying defined.
- **CFalseQuery** & **operator=** (const **CFalseQuery** &)
No assignment defined.

6.14.1 Detailed Description

Trivial False query.

Query execution always returns false.

Definition at line 72 of file TrivialQuery.hpp.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 CFalseQuery::CFalseQuery () [inline]

Default constructor.

Definition at line 118 of file TrivialQuery.hpp.

6.14.2.2 `CFalseQuery::~~CFalseQuery ()` [inline, virtual]

Virtual destructor.

Definition at line 123 of file TrivialQuery.hpp.

6.14.2.3 `CFalseQuery::CFalseQuery (const CFalseQuery &)` [private]

No copying defined.

6.14.3 Member Function Documentation**6.14.3.1** `CFalseQuery& CFalseQuery::operator= (const CFalseQuery &)` [private]

No assignment defined.

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

- **TrivialQuery.hpp**

6.15 CIdentifier Class Reference

Identifier holder.

```
#include <Identifier.hpp>
```

Public Member Functions

- **CIdentifier** ()
Default constructor.
- **CIdentifier** (const char *szText)
Construct identifier from a string.
- virtual **~CIdentifier** ()
Virtual destructor.
- **operator const std::string &** ()
Automatic conversion to std::string.
- **operator const char *** ()
*Automatic conversion to char *.*
- **const std::string & get** ()
Explicit conversion to string.

Private Member Functions

- **CIdentifier** (const **CIdentifier** &)
Copying is strictly forbidden.
- **CIdentifier & operator=** (const **CIdentifier** &)
Copying is strictly forbidden.

Private Attributes

- **std::string m_szIdentifier**
Identifier string.

6.15.1 Detailed Description

Identifier holder.

Holds an identifier.

When it comes to copying, the same handling is applied as to **CCodeBlock** (p. 30) class.

Definition at line 45 of file Identifier.hpp.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 CIdentifier::CIdentifier () [inline]

Default constructor.

Definition at line 111 of file Identifier.hpp.

6.15.2.2 CIdentifier::CIdentifier (const char * *szText*) [inline]

Construct identifier from a string.

Definition at line 117 of file Identifier.hpp.

6.15.2.3 CIdentifier::~~CIdentifier () [inline, virtual]

Virtual destructor.

Definition at line 123 of file Identifier.hpp.

6.15.2.4 CIdentifier::CIdentifier (const CIdentifier &) [private]

Copying is strictly forbidden.

6.15.3 Member Function Documentation

6.15.3.1 CIdentifier::operator const std::string & () [inline]

Automatic conversion to std::string.

Definition at line 105 of file Identifier.hpp.

References m_szIdentifier.

6.15.3.2 CIdentifier::operator const char * () [inline]

Automatic conversion to char *.

Definition at line 99 of file Identifier.hpp.

References m_szIdentifier.

6.15.3.3 const std::string & CIdentifier::get () [inline]

Explicit conversion to string.

Definition at line 128 of file Identifier.hpp.

6.15.3.4 CIdentifier& CIdentifier::operator= (const CIdentifier &) [private]

Copying is strictly forbidden.

6.15.4 Member Data Documentation

6.15.4.1 `std::string CIdentifier::m_szIdentifier` [private]

Identifier string.

Definition at line 79 of file Identifier.hpp.

Referenced by operator `const char *()`, and operator `const std::string &()`.

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

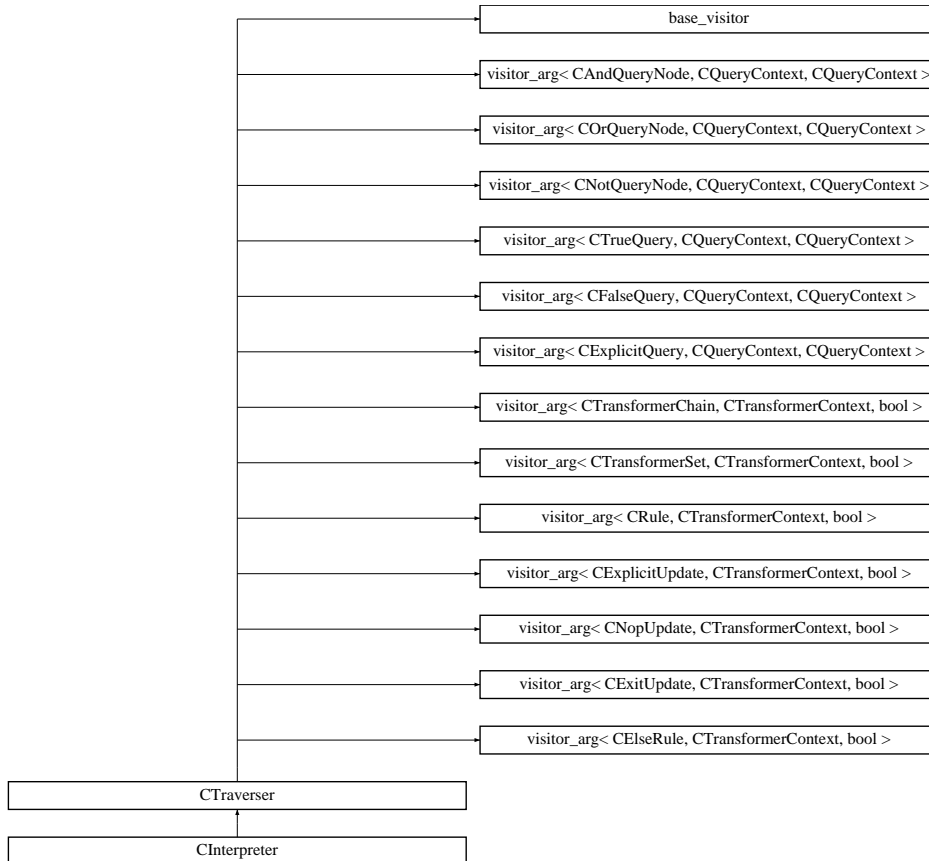
- **Identifier.hpp**

6.16 CInterpreter Class Reference

Interpreter engine of the Jazzyk language interpreter.

```
#include <Interpreter.hpp>
```

Inheritance diagram for CInterpreter::



Public Member Functions

- **CInterpreter** (TPtrProgram pProgram)
Constructor with the program to be interpreted.
- virtual ~**CInterpreter** ()
Virtual destructor.
- virtual void **run** ()
Main interface method for running the program.
- void **configure** (unsigned int nCycles=0, bool bFirst=false)
Configuration method.

Protected Member Functions

- virtual bool **Visit** (**CTransformerSet** &guest, **CTransformerContext** ctx)
*Overriden inherited parts of the **visitor** (p. 218) interface.*
- virtual **CQueryContext** & **PostAction** (**CTrueQuery** &guest, **CQueryContext** &ctx)
- virtual **CQueryContext** & **PostAction** (**CFalseQuery** &guest, **CQueryContext** &ctx)
- virtual **CQueryContext** & **PostAction** (**CExplicitQuery** &guest, **CQueryContext** &ctx)
- virtual void **PostAction** (**CExplicitUpdate** &guest, **CTransformerContext** &ctx)
- virtual void **PostAction** (**CExitUpdate** &guest, **CTransformerContext** &ctx)

Protected Attributes

- struct **CInterpreter::SSettings** m_settings
Interpreter configuration.

Private Member Functions

- void **doCycle** ()
Performs a single interpreter deliberation cycle.
- **CInterpreter** (const **CInterpreter** &)
No copying defined.
- **CInterpreter** & **operator=** (const **CInterpreter** &)
No assignment defined.

Private Attributes

- **TPtrProgram** m_pProgram
Pointer to the program to be interpreted.
- `std::pointer_to_unary_function< int, int >` m_pfRandomGenerator
Pointer to the random generator functor.

Classes

- struct **SSettings**
Interpreter configuration.

6.16.1 Detailed Description

Interpreter engine of the Jazzyk language interpreter.

Implementation of the core functionality of the Jazzyk programming language interpreter.

Interpreter service class is a Traverser-type **visitor** (p. 218) of the **CExpression** (p. 62) hierarchy.

Interpreter has few parameters to configure:

- number of cycles: by default the interpreter is supposed to perform infinite number of interpretation cycles. The number of cycles performed can be set.
- policy for transformer sets: by default, the interpreter chooses a random transformer from the transformer set. It can be changed to consider transformers in random order.

Definition at line 58 of file Interpreter.hpp.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 CInterpreter::CInterpreter (TPtrProgram *pProgram*)

Constructor with the program to be interpreted.

Definition at line 72 of file Interpreter.cpp.

6.16.2.2 CInterpreter::~CInterpreter () [virtual]

Virtual destructor.

Definition at line 81 of file Interpreter.cpp.

6.16.2.3 CInterpreter::CInterpreter (const CInterpreter &) [private]

No copying defined.

6.16.3 Member Function Documentation

6.16.3.1 void CInterpreter::run () [virtual]

Main interface method for running the program.

Program is interpreted in an infinite cycle until exit command is found, or an error occurs.

In the case of an error, or exit command execution, the interpreter throws **CError** (p. 47) exception.

Definition at line 86 of file Interpreter.cpp.

References doCycle(), m_settings, and CInterpreter::SSettings::nCycles.

Referenced by CJazzykApp::run().

6.16.3.2 void CInterpreter::configure (unsigned int *nCycles* = 0, bool *bFirst* = false)

Configuration method.

In the case, the interpreter is supposed to run in a configured mode, this method should be called to change the default settings.

By default, the interpreter performs infinite number of steps and chooses random transformer from the set.

Definition at line 214 of file Interpreter.cpp.

References CInterpreter::SSettings::eTransformerSetPolicy, m_settings, CInterpreter::SSettings::nCycles, CInterpreter::SSettings::TS_FIRST, and CInterpreter::SSettings::TS_RANDOM.

Referenced by CJazzykApp::run().

6.16.3.3 bool CInterpreter::Visit (CTransformerSet & *guest*, CTransformerContext *ctx*) [protected, virtual]

Overriden inherited parts of the **visitor** (p. 218) interface.

Reimplemented from CTraverser (p. 189).

Definition at line 109 of file Interpreter.cpp.

References CInterpreter::SSettings::eTransformerSetPolicy, CTransformerSet::getTransformers(), m_pfRandomGenerator, m_settings, PostAction(), CTraverser::PreAction(), and CInterpreter::SSettings::TS_RANDOM.

6.16.3.4 CQueryContext & CInterpreter::PostAction (CTrueQuery & *guest*, CQueryContext & *ctx*) [protected, virtual]

Reimplemented from CTraverser (p. 193).

Definition at line 147 of file Interpreter.cpp.

References CQueryContext::QTRUE, and CQueryContext::setResult().

Referenced by Visit().

6.16.3.5 CQueryContext & CInterpreter::PostAction (CFalseQuery & *guest*, CQueryContext & *ctx*) [protected, virtual]

Reimplemented from CTraverser (p. 193).

Definition at line 156 of file Interpreter.cpp.

References CQueryContext::QFALSE, and CQueryContext::setResult().

6.16.3.6 CQueryContext & CInterpreter::PostAction (CExplicitQuery & *guest*, CQueryContext & *ctx*) [protected, virtual]

Reimplemented from CTraverser (p. 193).

Definition at line 165 of file Interpreter.cpp.

References CExplicitQuery::getCodeBlock(), CExplicitQuery::getFreeVariables(), CExplicitQuery::getModuleID(), CExplicitQuery::getOperation(), CExpression::getSrcPosition(), CModuleManager::Instance(), CError::m_errorType, CError::m_szReason, CQueryContext::QFALSE, CQueryContext::QTRUE, and CQueryContext::setResult().

6.16.3.7 void CInterpreter::PostAction (CExplicitUpdate & *guest*, CTransformerContext & *ctx*) [protected, virtual]

Reimplemented from CTraverser (p. 194).

Definition at line 188 of file Interpreter.cpp.

References CExplicitUpdate::getCodeBlock(), CExplicitUpdate::getModuleID(), CExplicitUpdate::getOperation(), CExpression::getSrcPosition(), CExplicitUpdate::getUsedVariables(), CModuleManager::Instance(), CError::m_errorType, CError::m_szReason, and CModuleManager::update().

6.16.3.8 void CInterpreter::PostAction (CExitUpdate & *guest*, CTransformerContext & *ctx*) [protected, virtual]

Reimplemented from CTraverser (p. 194).

Definition at line 207 of file Interpreter.cpp.

References CExpression::getSrcPosition(), and CError::JZNOERROR.

6.16.3.9 void CInterpreter::doCycle () [private]

Performs a single interpreter deliberation cycle.

Definition at line 102 of file Interpreter.cpp.

References m_pProgram, and CTraverser::traverse().

Referenced by run().

6.16.3.10 CInterpreter& CInterpreter::operator= (const CInterpreter &) [private]

No assignment defined.

6.16.4 Member Data Documentation

6.16.4.1 struct CInterpreter::SSettings CInterpreter::m_settings [protected]

Interpreter configuration.

Referenced by configure(), run(), and Visit().

6.16.4.2 TPtrProgram CInterpreter::m_pProgram [private]

Pointer to the program to be interpreted.

Definition at line 136 of file Interpreter.hpp.

Referenced by doCycle().

6.16.4.3 `std::pointer_to_unary_function<int, int> CInterpreter::m_pfRandomGenerator` [private]

Pointer to the random generator functor.

Definition at line 139 of file Interpreter.hpp.

Referenced by Visit().

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

- **Interpreter.hpp**
- **Interpreter.cpp**

6.17 CInterpreter::SSettings Struct Reference

Interpreter configuration.

```
#include <Interpreter.hpp>
```

Public Types

- enum { **TS_RANDOM**, **TS_FIRST** }
Transformer set interpretation policy.

Public Member Functions

- **SSettings** ()
Default constructor.

Public Attributes

- unsigned int **nCycles**
Number of interpretation cycles to perform.
- enum CInterpreter::SSettings:: { ... } **eTransformerSetPolicy**
Transformer set interpretation policy.

6.17.1 Detailed Description

Interpreter configuration.

Definition at line 109 of file Interpreter.hpp.

6.17.2 Member Enumeration Documentation

6.17.2.1 anonymous enum

Transformer set interpretation policy.

Enumerator:

TS_RANDOM
TS_FIRST

Definition at line 118 of file Interpreter.hpp.

6.17.3 Constructor & Destructor Documentation

6.17.3.1 CInterpreter::SSettings::SSettings ()

Default constructor.

Definition at line 62 of file Interpreter.cpp.

6.17.4 Member Data Documentation

6.17.4.1 unsigned int CInterpreter::SSettings::nCycles

Number of interpretation cycles to perform.

Definition at line 115 of file Interpreter.hpp.

Referenced by CInterpreter::configure(), and CInterpreter::run().

6.17.4.2 enum { ... } CInterpreter::SSettings::eTransformerSetPolicy

Transformer set interpretation policy.

Referenced by CInterpreter::configure(), and CInterpreter::Visit().

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

- **Interpreter.hpp**
- **Interpreter.cpp**

6.18 CIPCPtrDeleter< T > Class Template Reference

A functor that destroys the shared memory object of an associated smart pointer.

```
#include <ModuleCommData.hpp>
```

Public Types

- typedef T * **TStoredPtr**
Smart pointer type to store.

Public Member Functions

- **CIPCPtrDeleter** (**TSegmentManager** *manager)
Constructor.
- void **operator()** (**TStoredPtr** pObject)
Deleter operator.

Private Types

- typedef boost::interprocess::managed_shared_memory::segment_manager **TSegmentManager**
Typedef of the associated shared memory segment manager.

Private Attributes

- **TSegmentManager** * **m_pSegmentManager**
Associated shared memory segment manager.

6.18.1 Detailed Description

```
template<class T> class CIPCPtrDeleter< T >
```

A functor that destroys the shared memory object of an associated smart pointer.

The deleter object is used for specialised smart pointers and correctly destroys the pointer in the shared memory the associated pointer points to.

Definition at line 279 of file ModuleCommData.hpp.

6.18.2 Member Typedef Documentation

6.18.2.1 `template<class T> typedef boost::interprocess::managed_shared_memory::segment_manager CIpcPtrDeleter< T >::TSegmentManager [private]`

Typedef of the associated shared memory segment manager.

Definition at line 283 of file ModuleCommData.hpp.

6.18.2.2 `template<class T> typedef T* CIpcPtrDeleter< T >::TStoredPtr`

Smart pointer type to store.

Definition at line 290 of file ModuleCommData.hpp.

6.18.3 Constructor & Destructor Documentation

6.18.3.1 `template<class T> CIpcPtrDeleter< T >::CIpcPtrDeleter (TSegmentManager * manager) [inline]`

Constructor.

Definition at line 293 of file ModuleCommData.hpp.

6.18.4 Member Function Documentation

6.18.4.1 `template<class T> void CIpcPtrDeleter< T >::operator() (TStoredPtr pObject) [inline]`

Deleter operator.

Definition at line 298 of file ModuleCommData.hpp.

References CIpcPtrDeleter< T >::m_pSegmentManager.

6.18.5 Member Data Documentation

6.18.5.1 `template<class T> TSegmentManager* CIpcPtrDeleter< T >::m_pSegmentManager [private]`

Associated shared memory segment manager.

Definition at line 286 of file ModuleCommData.hpp.

Referenced by CIpcPtrDeleter< T >::operator()().

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

- **ModuleCommData.hpp**

6.19 CJazzykApp Class Reference

Main Jazzyk application class.

```
#include <JazzykApp.hpp>
```

Public Member Functions

- `int run (int argc, char *argv[])`
Main application entry point.

Static Public Member Functions

- `static CJazzykApp & Instance ()`
Lazy singleton instance creation.

Protected Member Functions

- `void error ()`
Application error handler.

Private Member Functions

- `bool processCmdLine (int argc, char *argv[])`
Processes the command line and produces a configuration.
- `void version ()`
Prints the version information for the program.
- `void license ()`
Prints the warranty information for the program.
- `void read (std::istream &inStream, std::stringstream &outStream)`
Reads the input from inStream and appends it to the output stream.
- `CJazzykApp ()`
Hidden default constructor.
- `virtual ~CJazzykApp ()`
Hidden virtual destructor.
- `CJazzykApp (const CJazzykApp &)`
No copying defined.
- `CJazzykApp & operator= (const CJazzykApp &)`
No assignment defined.

Private Attributes

- struct **CJazzykApp::SSettings** `m_settings`
Settings wrapper.

Classes

- struct **SSettings**
Settings wrapper.

6.19.1 Detailed Description

Main Jazzyk application class.

CJazzykApp (p. 80) is the class representing the main Jazzyk application. It is the system's entry point.

CJazzykApp (p. 80) is a singleton.

Definition at line 48 of file JazzykApp.hpp.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 CJazzykApp::CJazzykApp () [private]

Hidden default constructor.

Definition at line 86 of file JazzykApp.cpp.

6.19.2.2 CJazzykApp::~CJazzykApp () [private, virtual]

Hidden virtual destructor.

Definition at line 91 of file JazzykApp.cpp.

6.19.2.3 CJazzykApp::CJazzykApp (const CJazzykApp &) [private]

No copying defined.

6.19.3 Member Function Documentation

6.19.3.1 CJazzykApp & CJazzykApp::Instance () [static]

Lazy singleton instance creation.

Definition at line 96 of file JazzykApp.cpp.

Referenced by `main()`.

6.19.3.2 int CJazzykApp::run (int argc, char * argv[])

Main application entry point.

Takes the command line as an argument.

Definition at line 243 of file JazzykApp.cpp.

References CJazzykApp::SSettings::bBypassMP, CJazzykApp::SSettings::bCheckQuery, CJazzykApp::SSettings::bCompileOnly, CJazzykApp::SSettings::bPreprocessOnly, CJazzykApp::SSettings::bPrettyPrint, CJazzykApp::SSettings::bReadStdIn, CJazzykApp::SSettings::bSetFirst, CCompiler::compile(), CInterpreter::configure(), CJazzykApp::SSettings::files, CModuleManager::finalize(), CJazzykApp::SSettings::includePaths, CModuleManager::initialize(), CModuleManager::Instance(), CError::JZOTHER_ERROR, m_settings, CJazzykApp::SSettings::modulePaths, CJazzykApp::SSettings::nCycles, CJazzykApp::SSettings::nMemSize, CCompiler::parse(), CCompiler::preprocess(), processCmdLine(), read(), CInterpreter::run(), CCompiler::setIncludePaths(), and CError::what().

Referenced by main(), and processCmdLine().

6.19.3.3 void CJazzykApp::error () [protected]

Application error handler.

6.19.3.4 bool CJazzykApp::processCmdLine (int argc, char * argv[]) [private]

Processes the command line and produces a configuration.

As an argument takes the command line. Returns true when the command line was processed properly. Otherwise the application is supposed to exit immediately - the method returns false.

The resulting configuration is returned in the last output argument.

Definition at line 113 of file JazzykApp.cpp.

References CJazzykApp::SSettings::bBypassMP, CJazzykApp::SSettings::bCheckQuery, CJazzykApp::SSettings::bCompileOnly, CJazzykApp::SSettings::bPreprocessOnly, CJazzykApp::SSettings::bPrettyPrint, CJazzykApp::SSettings::bReadStdIn, CJazzykApp::SSettings::bSetFirst, CJazzykApp::SSettings::files, CJazzykApp::SSettings::includePaths, license(), m_settings, CJazzykApp::SSettings::modulePaths, MOTTO, CJazzykApp::SSettings::nCycles, CJazzykApp::SSettings::nMemSize, run(), and version().

Referenced by run().

6.19.3.5 void CJazzykApp::version () [private]

Prints the version information for the program.

Definition at line 213 of file JazzykApp.cpp.

References JAZZYK_API_VERSION.

Referenced by processCmdLine().

6.19.3.6 void CJazzykApp::license () [private]

Prints the warranty information for the program.

Definition at line 221 of file JazzykApp.cpp.

Referenced by processCmdLine().

6.19.3.7 void CJazzykApp::read (std::istream & *inStream*, std::stringstream & *outStream*) [private]

Reads the input from *inStream* and appends it to the output stream.

Reads always till the very end of the input stream.

Definition at line 104 of file JazzykApp.cpp.

Referenced by run().

6.19.3.8 CJazzykApp& CJazzykApp::operator= (const CJazzykApp &) [private]

No assignment defined.

6.19.4 Member Data Documentation

6.19.4.1 struct CJazzykApp::SSettings CJazzykApp::m_settings [private]

Settings wrapper.

Referenced by processCmdLine(), and run().

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

- JazzykApp.hpp
- JazzykApp.cpp

6.20 CJazzykApp::SSettings Struct Reference

Settings wrapper.

Public Member Functions

- **SSettings** ()
Default constructor.

Public Attributes

- `std::vector< std::string >` **files**
List of files to process.
- `std::vector< std::string >` **includePaths**
List of include paths to pass to the macro processor.
- `std::vector< std::string >` **modulePaths**
List of module paths.
- `bool` **bReadStdIn**
Explicit flag to read also the stdin after the input files are read.
- `bool` **bPrettyPrint**
Pretty print the program tree structure.
- `bool` **bPreprocessOnly**
Run only macro preprocessing step.
- `bool` **bCompileOnly**
Compile only.
- `bool` **bBypassMP**
Bypass the macro preprocessor.
- `unsigned int` **nCycles**
Run only N cycles of the interpreter.
- `bool` **bSetFirst**
Choose from a set transformers in first-true manner.
- `unsigned int` **nMemSize**
Size of the shared memory between the main process and plug-ins.
- `bool` **bCheckQuery**
Check the resulting query substitutions.

6.20.1 Detailed Description

Settings wrapper.

Definition at line 75 of file JazzykApp.hpp.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 CJazzykApp::SSettings::SSettings ()

Default constructor.

Definition at line 66 of file JazzykApp.cpp.

6.20.3 Member Data Documentation

6.20.3.1 `std::vector<std::string> CJazzykApp::SSettings::files`

List of files to process.

Definition at line 81 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.2 `std::vector<std::string> CJazzykApp::SSettings::includePaths`

List of include paths to pass to the macro processor.

Definition at line 84 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.3 `std::vector<std::string> CJazzykApp::SSettings::modulePaths`

List of module paths.

Definition at line 87 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.4 `bool CJazzykApp::SSettings::bReadStdIn`

Explicit flag to read also the stdin after the input files are read.

Definition at line 90 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.5 `bool CJazzykApp::SSettings::bPrettyPrint`

Pretty print the program tree structure.

Definition at line 93 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.6 `bool CJazzykApp::SSettings::bPreprocessOnly`

Run only macro preprocessing step.

Definition at line 96 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.7 `bool CJazzykApp::SSettings::bCompileOnly`

Compile only.

Definition at line 99 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.8 `bool CJazzykApp::SSettings::bBypassMP`

Bypass the macro preprocessor.

Definition at line 102 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.9 `unsigned int CJazzykApp::SSettings::nCycles`

Run only N cycles of the interpreter.

Definition at line 105 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.10 `bool CJazzykApp::SSettings::bSetFirst`

Choose from a set transformers in first-true manner.

Definition at line 108 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.11 `unsigned int CJazzykApp::SSettings::nMemSize`

Size of the shared memory between the main process and plug-ins.

Definition at line 111 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

6.20.3.12 `bool CJazzykApp::SSettings::bCheckQuery`

Check the resulting query substitutions.

Definition at line 114 of file JazzykApp.hpp.

Referenced by `CJazzykApp::processCmdLine()`, and `CJazzykApp::run()`.

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

- JazzykApp.hpp
- JazzykApp.cpp

6.21 CJazzykLexer Class Reference

Specialisation of the standard Flex C++ yyFlexLexer class.

```
#include <ParserAux.hpp>
```

Public Member Functions

- **CJazzykLexer** (std::istream *inputStream)
Default constructor with the input stream.
- virtual **~CJazzykLexer** ()
Virtual destructor.
- virtual int **yylex** (void *yylval, void *yylloc)
Added yylex specialisation with yylval for return value and yylloc for locations.

Public Attributes

- void * **m_pyylval**
Pointer to the yylval argument of CJazzykLexer::lex().
- void * **m_pyylloc**
Pointer to the yylloc argument of CJazzykLexer::lex().

Private Member Functions

- **CJazzykLexer** (const **CJazzykLexer** &)
No copying defined.
- **CJazzykLexer** & **operator=** (const **CJazzykLexer** &)
No assignment defined.

6.21.1 Detailed Description

Specialisation of the standard Flex C++ yyFlexLexer class.

We need a special subclass of the stock yyFlexLexer class generated by Flex in the C++ mode in order to correctly communicate correctly with the Bison parser in C++ mode.

The **yylex()** (p. 89) is defined using void * because at this stage the Parser.h++ is not generated yet. Actually yylval and yylloc are of internal types of JazzykParser generated by Bison.

Definition at line 102 of file ParserAux.hpp.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 CJazzykLexer::CJazzykLexer (std::istream * *inputStream*)

Default constructor with the input stream.

Definition at line 59 of file ParserAux.cpp.

6.21.2.2 CJazzykLexer::~CJazzykLexer () [virtual]

Virtual destructor.

Definition at line 65 of file ParserAux.cpp.

6.21.2.3 CJazzykLexer::CJazzykLexer (const CJazzykLexer &) [private]

No copying defined.

6.21.3 Member Function Documentation

6.21.3.1 int CJazzykLexer::yylex (void * *yylval*, void * *yylloc*) [virtual]

Added yylex specialisation with *yylval* for return value and *yylloc* for locations.

yylval is actually of the type `yy::JazzykParser::semantic_type *` *yylloc* is actually of the type `yy::JazzykParser::location_type *`

Definition at line 70 of file ParserAux.cpp.

References `m_pyylloc`, and `m_pyylval`.

6.21.3.2 CJazzykLexer& CJazzykLexer::operator= (const CJazzykLexer &) [private]

No assignment defined.

6.21.4 Member Data Documentation

6.21.4.1 void* CJazzykLexer::m_pyylval

Pointer to the *yylval* argument of `CJazzykLexer::lex()`.

Definition at line 134 of file ParserAux.hpp.

Referenced by `yylex()`.

6.21.4.2 void* CJazzykLexer::m_pyylloc

Pointer to the *yylloc* argument of `CJazzykLexer::lex()`.

Definition at line 137 of file ParserAux.hpp.

Referenced by `yylex()`.

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

- `ParserAux.hpp`
- `ParserAux.cpp`

6.22 CLocationTracker Class Reference

Small singleton for communication between the Bison location and CExpressions.

```
#include <LocationTracker.hpp>
```

Public Member Functions

- void **set** (yy::location loc)
Sets a new location.
- yy::location **get** () const
Returns the last location stored.

Static Public Member Functions

- static **CLocationTracker & Instance** ()
Returns the singleton instance.

Protected Member Functions

- **CLocationTracker** ()
Default constructor.
- virtual **~CLocationTracker** ()
Virtual destructor.

Private Member Functions

- **CLocationTracker** (const **CLocationTracker** &)
No copying defined.
- **CLocationTracker & operator=** (const **CLocationTracker** &)
No assignment defined.

Private Attributes

- yy::location **m_srcLocation**

6.22.1 Detailed Description

Small singleton for communication between the Bison location and CExpressions.

Holds the last recorded location of a token. When a new **CExpression** (p. 62) object is created, it copies the location and stores it for later verbatim error reporting.

This class is a singleton.

Definition at line 46 of file LocationTracker.hpp.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 CLocationTracker::CLocationTracker () [inline, protected]

Default constructor.

Definition at line 94 of file LocationTracker.hpp.

6.22.2.2 CLocationTracker::~~CLocationTracker () [inline, protected, virtual]

Virtual destructor.

Definition at line 100 of file LocationTracker.hpp.

6.22.2.3 CLocationTracker::CLocationTracker (const CLocationTracker &) [private]

No copying defined.

6.22.3 Member Function Documentation

6.22.3.1 CLocationTracker & CLocationTracker::Instance () [inline, static]

Returns the singleton instance.

Definition at line 105 of file LocationTracker.hpp.

6.22.3.2 void CLocationTracker::set (yy::location *loc*) [inline]

Sets a new location.

Definition at line 112 of file LocationTracker.hpp.

References `m_srcLocation`.

6.22.3.3 yy::location CLocationTracker::get () const [inline]

Returns the last location stored.

Definition at line 118 of file LocationTracker.hpp.

References `m_srcLocation`.

6.22.3.4 CLocationTracker& CLocationTracker::operator= (const CLocationTracker &) [private]

No assignment defined.

6.22.4 Member Data Documentation

6.22.4.1 yy::location CLocationTracker::m_srcLocation [private]

Definition at line 72 of file LocationTracker.hpp.

Referenced by `get()`, and `set()`.

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

- **LocationTracker.hpp**

6.23 CModuleDeclaration Class Reference

Module declaration and link holder.

```
#include <ModuleDeclaration.hpp>
```

Public Member Functions

- **CModuleDeclaration** (const TPtrIdentifier &pIdentifier, const TPtrIdentifier &pType)
Default constructor.
- virtual **~CModuleDeclaration** ()
Virtual destructor.
- const TPtrIdentifier & **getIdentifier** () const
Returns the identifier of the module.
- const TPtrIdentifier & **getType** () const
Returns the module type identifier.
- void **addNotification** (const TPtrModuleNotification &pNotification)
Adds notification code to the module.
- void **addQueryOperation** (const TPtrIdentifier &pOperationID)
Adds a module query interface operation.
- void **addUpdateOperation** (const TPtrIdentifier &pOperationID)
Adds a module update interface operation.
- bool **isValidQueryOperation** (const TPtrIdentifier &pOperationID) const
Returns true if the argument is an available module query operation.
- bool **isValidUpdateOperation** (const TPtrIdentifier &pOperationID) const
Returns true if the argument is an available module update operation.
- const TPtrCodeBlock & **getNotificationCode** (CModuleNotification::ENotificationType notifyType) const
Returns the requested notification code pointer.
- const TSrcLocation & **getSrcPosition** () const
Returns the source code location of the module declaration.

Private Member Functions

- **CModuleDeclaration** (const CModuleDeclaration &)
No copying defined.
- **CModuleDeclaration** & **operator=** (const CModuleDeclaration &)
No assignment defined.

Private Attributes

- **TPtrIdentifier m_pIdentifier**
Identifier of the associated module.
- **TPtrIdentifier m_pType**
Module type identifier.
- **TPtrCodeBlock m_pInitCode**
Codeblock for initialization of the module.
- **TPtrCodeBlock m_pFinalizeCode**
Codeblock for finalization of the module.
- **TPtrCodeBlock m_pCycleCode**
Codeblock for notification on interpretation cycle start.
- **TSrcLocation m_srcLocation**
Source location of the module declaration.
- **TIdentifierSet m_queryOperations**
List of available query module operations.
- **TIdentifierSet m_updateOperations**
List of available update module operations.

6.23.1 Detailed Description

Module declaration and link holder.

The class is a holder of module declaration specification. It is used to load and handle the external plugins of the interpreter.

Definition at line 45 of file ModuleDeclaration.hpp.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 CModuleDeclaration::CModuleDeclaration (const TPtrIdentifier & pIdentifier, const TPtrIdentifier & pType)

Default constructor.

Definition at line 43 of file ModuleDeclaration.cpp.

6.23.2.2 CModuleDeclaration::~~CModuleDeclaration () [virtual]

Virtual destructor.

Definition at line 58 of file ModuleDeclaration.cpp.

6.23.2.3 CModuleDeclaration::CModuleDeclaration (const CModuleDeclaration &) [private]

No copying defined.

6.23.3 Member Function Documentation

6.23.3.1 const TPtrIdentifier & CModuleDeclaration::getIdentifier () const [inline]

Returns the identifier of the module.

Definition at line 135 of file ModuleDeclaration.hpp.

References m_pIdentifier.

6.23.3.2 const TPtrIdentifier & CModuleDeclaration::getType () const [inline]

Returns the module type identifier.

Definition at line 148 of file ModuleDeclaration.hpp.

References m_pType.

6.23.3.3 void CModuleDeclaration::addNotification (const TPtrModuleNotification & pNotification)

Adds notification code to the module.

Definition at line 63 of file ModuleDeclaration.cpp.

References m_pCycleCode, m_pFinalizeCode, m_pInitCode, CModuleNotification::NOTIFY_CYCLE, CModuleNotification::NOTIFY_FINALIZE, and CModuleNotification::NOTIFY_INITIALIZE.

6.23.3.4 void CModuleDeclaration::addQueryOperation (const TPtrIdentifier & pOperationID)

Adds a module query interface operation.

Definition at line 86 of file ModuleDeclaration.cpp.

References m_queryOperations.

6.23.3.5 void CModuleDeclaration::addUpdateOperation (const TPtrIdentifier & pOperationID)

Adds a module update interface operation.

Definition at line 92 of file ModuleDeclaration.cpp.

References m_updateOperations.

6.23.3.6 `bool CModuleDeclaration::isValidQueryOperation (const TPtrIdentifier & pOperationID) const [inline]`

Returns true if the argument is an available module query operation.

Definition at line 154 of file ModuleDeclaration.hpp.

References `m_queryOperations`.

6.23.3.7 `bool CModuleDeclaration::isValidUpdateOperation (const TPtrIdentifier & pOperationID) const [inline]`

Returns true if the argument is an available module update operation.

Definition at line 160 of file ModuleDeclaration.hpp.

References `m_updateOperations`.

6.23.3.8 `const TPtrCodeBlock & CModuleDeclaration::getNotificationCode (CModuleNotification::ENotificationType notifyType) const`

Returns the requested notification code pointer.

Definition at line 98 of file ModuleDeclaration.cpp.

References `CError::JZOTHER_ERROR`, `m_pCycleCode`, `m_pFinalizeCode`, `m_pInitCode`, `CModuleNotification::NOTIFY_CYCLE`, `CModuleNotification::NOTIFY_FINALIZE`, and `CModuleNotification::NOTIFY_INITIALIZE`.

6.23.3.9 `const TSrcLocation & CModuleDeclaration::getSrcPosition () const [inline]`

Returns the source code location of the module declaration.

Definition at line 141 of file ModuleDeclaration.hpp.

References `m_srcLocation`.

6.23.3.10 `CModuleDeclaration& CModuleDeclaration::operator= (const CModuleDeclaration &) [private]`

No assignment defined.

6.23.4 Member Data Documentation**6.23.4.1** `TPtrIdentifier CModuleDeclaration::m_pIdentifier [private]`

Identifier of the associated module.

Definition at line 97 of file ModuleDeclaration.hpp.

Referenced by `getIdentifier()`.

6.23.4.2 TPtrIdentifier CModuleDeclaration::m_pType [private]

Module type identifier.

Definition at line 100 of file ModuleDeclaration.hpp.

Referenced by getType().

6.23.4.3 TPtrCodeBlock CModuleDeclaration::m_pInitCode [private]

Codeblock for initialization of the module.

Definition at line 103 of file ModuleDeclaration.hpp.

Referenced by addNotification(), and getNotificationCode().

6.23.4.4 TPtrCodeBlock CModuleDeclaration::m_pFinalizeCode [private]

Codeblock for finalization of the module.

Definition at line 106 of file ModuleDeclaration.hpp.

Referenced by addNotification(), and getNotificationCode().

6.23.4.5 TPtrCodeBlock CModuleDeclaration::m_pCycleCode [private]

Codeblock for notification on interpretation cycle start.

Definition at line 109 of file ModuleDeclaration.hpp.

Referenced by addNotification(), and getNotificationCode().

6.23.4.6 TSrcLocation CModuleDeclaration::m_srcLocation [private]

Source location of the module declaration.

Definition at line 112 of file ModuleDeclaration.hpp.

Referenced by getSrcPosition().

6.23.4.7 TIdentifierSet CModuleDeclaration::m_queryOperations [private]

List of available query module operations.

Definition at line 115 of file ModuleDeclaration.hpp.

Referenced by addQueryOperation(), and isValidQueryOperation().

6.23.4.8 TIdentifierSet CModuleDeclaration::m_updateOperations [private]

List of available update module operations.

Definition at line 118 of file ModuleDeclaration.hpp.

Referenced by addUpdateOperation(), and isValidUpdateOperation().

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

- [ModuleDeclaration.hpp](#)
- [ModuleDeclaration.cpp](#)

6.24 CModuleManager Class Reference

External plug-ins/modules manager.

```
#include <ModuleManager.hpp>
```

Public Member Functions

- void **initialize** (const std::vector< std::string > &vszSearchPaths, unsigned int nMemSize, bool bCheckQuery)
Configures the module manager and initializes the server side shared memory.
- void **finalize** ()
Server side deinitialization of the module manager.
- void **load** (TPtrModuleDeclaration &pDecl)
Loads a plugin according to the module declaration.
- void **unload** (const TPtrModuleDeclaration &pDecl)
Unloads a plugin according to the module declaration.
- bool **query** (const TPtrIdentifier &pModuleID, const TPtrIdentifier &pOperation, const TPtrCodeBlock &pQueryCode, const TPtrIdentifiers &pFreeVars, TVariableSubstitution &substitution)
Executes a specified query in the given module.
- void **update** (const TPtrIdentifier &pModuleID, const TPtrIdentifier &pOperation, const TPtrCodeBlock &pUpdateCode, const TPtrIdentifiers &pUsedVars, TVariableSubstitution &substitution)
Executes a specified update in the given module.
- void **cycle** (const TPtrModuleDeclaration &pDecl)
Module notification about the end of the cycle.

Static Public Member Functions

- static CModuleManager & **Instance** ()
Returns the singleton instance.

Protected Member Functions

- void **load** (const TPtrIdentifier &pType, const TPtrIdentifier &pModuleID, const TPtrCodeBlock &pInitCode)
Loads the module.
- void **unload** (const TPtrIdentifier &pModuleID, const TPtrCodeBlock &pFinalizeCode)
Unloads the module identified by the argument.

Private Member Functions

- **TPtrSModuleRequest writeRequest** (const **EModuleRequestType** eReqType, const TPtrIdentifier *pOperation, const TPtrCodeBlock *ppCode, const **TVariableSubstitution** *pSubstitution)

Creates and write the request into the memory.
- **TPtrSModuleResponse extractResponse** (const TPtrIdentifier &pIdentifier, **TVariableSubstitution** *pSubstitution, bool *pbQueryResult)

Retrieves response from the shared memory.
- void **communicate** (const TPtrIdentifier &pModuleID, const **EModuleRequestType** eReqType, const TPtrIdentifier *ppOperation, const TPtrCodeBlock *ppCode, **TVariableSubstitution** *pSubstitution, bool *pbQueryResult)

Sends the request, waits for a response and retrieves it.
- void **killModule** (const TPtrIdentifier &pModuleID, bool bHang=true)

Kills the subprocess of the given KR module.
- void **filterVariables** (const **TVariableSubstitution** &inSubstitution, const TPtrIdentifiers &pFilter, **TVariableSubstitution** &outSubstitution)

Filters a variable substitution.
- **CModuleManager ()**

Default constructor.
- virtual **~CModuleManager ()**

Virtual destructor.
- **CModuleManager (const CModuleManager &)**

No copying defined.
- **CModuleManager & operator= (const CModuleManager &)**

No assignment defined.

Private Attributes

- **TIdentifierMap< pid_t > m_mapPIDSubprocesses**

Map of module subprocess' PIDs.
- **TVecStrings m_vecSearchPaths**

DLL search paths as specified by the app user.
- bool **m_bCheckQuery**

Flag for checking sanity of resulting query variable substitutions.

6.24.1 Detailed Description

External plug-ins/modules manager.

The singleton manages loading, unloading and storing handles to external modules for various KR techniques/languages. It is also responsible for the low level query/update interfacing with the modules.

Module is identified by a unique identifier `pId` and it is a dynamically linked library identified by `pType`. One plug-in library can serve several modules with different IDs. This is because it is not possible to dynamically load the same DLL library twice.

Important remark: `ModuleManager` makes a strong assumption that the program is sane and valid. That means that it does not perform additional checking whether modules are loaded/unloaded in the correct order, or that modules are being loaded twice, or unloaded when not loaded.

Definition at line 84 of file `ModuleManager.hpp`.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 `CModuleManager::CModuleManager () [private]`

Default constructor.

Definition at line 63 of file `ModuleManager.cpp`.

6.24.2.2 `CModuleManager::~CModuleManager () [private, virtual]`

Virtual destructor.

Definition at line 71 of file `ModuleManager.cpp`.

6.24.2.3 `CModuleManager::CModuleManager (const CModuleManager &) [private]`

No copying defined.

6.24.3 Member Function Documentation

6.24.3.1 `CModuleManager & CModuleManager::Instance () [static]`

Returns the singleton instance.

Also does the lazy initialization of the Meyers-type singleton

Definition at line 54 of file `ModuleManager.cpp`.

Referenced by `CProgram::cycle()`, `CProgram::loadModules()`, `CInterpreter::PostAction()`, `CJazzykApp::run()`, and `CProgram::unloadModules()`.

6.24.3.2 `void CModuleManager::initialize (const std::vector< std::string > & vszSearchPaths, unsigned int nMemSize, bool bCheckQuery)`

Configures the module manager and initializes the server side shared memory.

Sets search paths where to look for the modules. If unsuccessful, it throws **CError** (p. 47) exception.

A flag to check that all the variables returned by queries are instantiated is also configured.

Definition at line 78 of file ModuleManager.cpp.

References CShmServerMgr::initialize(), CShmServerMgr::Instance(), m_bCheckQuery, and m_vecSearchPaths.

Referenced by CJazzykApp::run().

6.24.3.3 void CModuleManager::finalize ()

Server side deinitialization of the module manager.

Definition at line 86 of file ModuleManager.cpp.

References CShmServerMgr::finalize(), CShmServerMgr::Instance(), CError::JZMODULE_ERROR, and m_mapPIDSubprocesses.

Referenced by CJazzykApp::run().

6.24.3.4 void CModuleManager::load (TPtrModuleDeclaration & pDecl)

Loads a plugin according to the module declaration.

Also retrieves the module query/update interface information and stores it in the provided module declaration instance.

Definition at line 179 of file ModuleManager.cpp.

References communicate(), g_JZAPI_QUERY, g_JZAPI_UPDATE, GETINTERFACE, CError::JZMODULE_ERROR, LOG, and CModuleNotification::NOTIFY_INITIALIZE.

Referenced by CProgram::loadModules().

6.24.3.5 void CModuleManager::unload (const TPtrModuleDeclaration & pDecl)

Unloads a plugin according to the module declaration.

Definition at line 216 of file ModuleManager.cpp.

References CModuleNotification::NOTIFY_FINALIZE.

Referenced by CProgram::unloadModules().

6.24.3.6 bool CModuleManager::query (const TPtrIdentifier & pModuleID, const TPtrIdentifier & pOperation, const TPtrCodeBlock & pQueryCode, const TPtrIdentifiers & pFreeVars, TVariableSubstitution & substitution)

Executes a specified query in the given module.

It takes a variable substitution as an input/output argument. After a successful query evaluation, the updated variable substitution is stored back to the substitution argument.

Additional checks are made on the returned substitution:

- no new variable can be added to the substitution

- no already substituted variable can be substituted again

Definition at line 225 of file ModuleManager.cpp.

References `communicate()`, `filterVariables()`, `CError::JZMODULE_ERROR`, `m_bCheckQuery`, and `QUERY`.

6.24.3.7 `void CModuleManager::update (const TPtrIdentifier & pModuleID, const TPtrIdentifier & pOperation, const TPtrCodeBlock & pUpdateCode, const TPtrIdentifiers & pUsedVars, TVariableSubstitution & substitution)`

Executes a specified update in the given module.

Takes a variable substitution as an argument. It does not modify this substitution.

Definition at line 294 of file ModuleManager.cpp.

References `communicate()`, `filterVariables()`, and `UPDATE`.

Referenced by `CInterpreter::PostAction()`.

6.24.3.8 `void CModuleManager::cycle (const TPtrModuleDeclaration & pDecl)`

Module notification about the end of the cycle.

At each cycle end, the plugin is notified by the code specified in the code.

Definition at line 311 of file ModuleManager.cpp.

References `communicate()`, `CYCLE`, and `CModuleNotification::NOTIFY_CYCLE`.

Referenced by `CProgram::cycle()`.

6.24.3.9 `void CModuleManager::load (const TPtrIdentifier & pType, const TPtrIdentifier & pModuleID, const TPtrCodeBlock & pInitCode) [protected]`

Loads the module.

Launches a new subprocess, which starts the plug-in wrapper (`CSingleModule` (p.165)) in a separate process and let's it load the library.

The communication with the subprocess happens via shared memory channel.

Definition at line 100 of file ModuleManager.cpp.

References `CShmServerMgr::addModuleSync()`, `communicate()`, `INITIALIZE`, `CSingleModule::Instance()`, `CShmServerMgr::Instance()`, `CError::JZMODULE_ERROR`, `LOG`, `m_mapPIDSsubprocesses`, `m_vecSearchPaths`, and `CSingleModule::run()`.

6.24.3.10 `void CModuleManager::unload (const TPtrIdentifier & pModuleID, const TPtrCodeBlock & pFinalizeCode) [protected]`

Unloads the module identified by the argument.

Finishes the corresponding plug-in subprocess. Subprocess is notified and requested to finalize itself with the provided finalization code and subsequently it should quit (or is destroyed by the main process).

Definition at line 145 of file ModuleManager.cpp.

References `communicate()`, `FINALIZE`, `killModule()`, `LOG`, and `m_mapPIDSubprocesses`.

6.24.3.11 `TPtrSModuleRequest CModuleManager::writeRequest (const EModuleRequestType eReqType, const TPtrIdentifier * pOperation, const TPtrCodeBlock * ppCode, const TVariableSubstitution * pSubstitution) [private]`

Creates and write the request into the memory.

Effectively, the substitution is broke down into two parts of already substituted variables and free variables. After obtaining a response from the module child process, these are again merged into one so that the module had no chance to change the already substituted variables.

Returns a handle to the request so that the caller can keep control over the shared memory object lifetime.

Only non-null arguments are written into the request structure.

If an error occurs it throws either `interprocess_exception`, or **CError** (p. 47).

Definition at line 319 of file ModuleManager.cpp.

References `CShmServerMgr::Instance()`, and `CError::JZMODULE_ERROR`.

Referenced by `communicate()`.

6.24.3.12 `TPtrSModuleResponse CModuleManager::extractResponse (const TPtrIdentifier & pIdentifier, TVariableSubstitution * pSubstitution, bool * pbQueryResult) [private]`

Retrieves response from the shared memory.

Returns a handle to the request so that the caller can keep control over the shared memory object lifetime.

For query responses the query result is also provided. Otherwise it should be ignored. The output substitution is returned in the second argument (if present).

Definition at line 368 of file ModuleManager.cpp.

References `CShmServerMgr::Instance()`, `CError::JZMODULE_ERROR`, `LOG`, and `OK`.

Referenced by `communicate()`.

6.24.3.13 `void CModuleManager::communicate (const TPtrIdentifier & pModuleID, const EModuleRequestType eReqType, const TPtrIdentifier * ppOperation, const TPtrCodeBlock * ppCode, TVariableSubstitution * pSubstitution, bool * pbQueryResult) [private]`

Sends the request, waits for a response and retrieves it.

Implements the synchronised communication section.

Only non-null arguments are communicated and returned. What was passed as null, it will return as null.

If the underlying module reports an error on return, or other error occurs, it throws **CError** (p. 47) exception.

Definition at line 418 of file ModuleManager.cpp.

References `extractResponse()`, `CShmServerMgr::Instance()`, `LOG`, `CShmServerMgr::notifyRequest()`, `CShmServerMgr::waitForReady()`, `CShmServerMgr::waitForResponse()`, and `writeRequest()`.

Referenced by `cycle()`, `load()`, `query()`, `unload()`, and `update()`.

6.24.3.14 `void CModuleManager::killModule (const TPtrIdentifier & pModuleID, bool bHang = true) [private]`

Kills the subprocess of the given KR module.

If the `bHang` argument is set to `false`, the process will be remorselessly killed right away, without polite waiting for its exit.

Definition at line 465 of file ModuleManager.cpp.

References `CError::JZMODULE_ERROR`, and `m_mapPIDSubprocesses`.

Referenced by `unload()`.

6.24.3.15 `void CModuleManager::filterVariables (const TVariableSubstitution & inSubstitution, const TPtrIdentifiers & pFilter, TVariableSubstitution & outSubstitution) [private]`

Filters a variable substitution.

Fills the output substitution, with the variables from the input substitution, which are present in the filter list.

Definition at line 481 of file ModuleManager.cpp.

References `CError::JZMODULE_ERROR`.

Referenced by `query()`, and `update()`.

6.24.3.16 `CModuleManager& CModuleManager::operator= (const CModuleManager &) [private]`

No assignment defined.

6.24.4 Member Data Documentation

6.24.4.1 `TIdentifierMap<pid_t> CModuleManager::m_mapPIDSubprocesses [private]`

Map of module subprocess' PIDs.

Definition at line 266 of file ModuleManager.hpp.

Referenced by `finalize()`, `killModule()`, `load()`, and `unload()`.

6.24.4.2 `TVecStrings CModuleManager::m_vecSearchPaths [private]`

DLL search paths as specified by the app user.

Definition at line 269 of file ModuleManager.hpp.

Referenced by initialize(), and load().

6.24.4.3 bool CModuleManager::m_bCheckQuery [private]

Flag for checking sanity of resulting query variable substitutions.

Definition at line 272 of file ModuleManager.hpp.

Referenced by initialize(), and query().

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

- **ModuleManager.hpp**
- **ModuleManager.cpp**

6.25 CModuleNotification Class Reference

Wrapper class for the module notification declaration.

```
#include <ModuleNotification.hpp>
```

Public Types

- enum **ENotificationType** { **NOTIFY_INITIALIZE**, **NOTIFY_FINALIZE**, **NOTIFY_CYCLE** }

Notification types.

Public Member Functions

- **CModuleNotification** (**ENotificationType** type, const TPtrIdentifier &id, const TPtrCodeBlock &code)
Default constructor.
- virtual **~CModuleNotification** ()
Virtual destructor.
- **ENotificationType** **getType** () const
Returns notification type.
- const TPtrCodeBlock & **getCode** () const
Returns the contained code block.
- const TPtrIdentifier & **getIdentifier** () const
Returns the contained module identifier.
- const TSrcLocation & **getSrcPosition** () const
Returns the source code position of the notification object.

Private Member Functions

- **CModuleNotification** (const **CModuleNotification** &)
No copying defined.
- **CModuleNotification** & **operator=** (const **CModuleNotification** &)
No assignment defined.

Private Attributes

- **ENotificationType** **m_eType**
Notification type.

- TPtrIdentifier **m_pIdentifier**
Notified module identifier.
- TPtrCodeBlock **m_pCode**
Code block of the notification declaration.
- TSrcLocation **m_srcLocation**
Source location of the notification.

6.25.1 Detailed Description

Wrapper class for the module notification declaration.

Module notification declaration serves only a temporary purpose during the source code parsing. **CModuleNotification** (p. 108) object is created, then passed to **CProgram** (p. 125) and **CModuleDeclaration** (p. 94) to transfer the event notification code and destroyed upon return.

Definition at line 48 of file ModuleNotification.hpp.

6.25.2 Member Enumeration Documentation

6.25.2.1 enum CModuleNotification::ENotificationType

Notification types.

Enumerator:

NOTIFY_INITIALIZE
NOTIFY_FINALIZE
NOTIFY_CYCLE

Definition at line 54 of file ModuleNotification.hpp.

6.25.3 Constructor & Destructor Documentation

6.25.3.1 CModuleNotification::CModuleNotification (ENotificationType *type*, const TPtrIdentifier & *id*, const TPtrCodeBlock & *code*) [inline]

Default constructor.

Definition at line 122 of file ModuleNotification.hpp.

6.25.3.2 CModuleNotification::~~CModuleNotification () [inline, virtual]

Virtual destructor.

Definition at line 131 of file ModuleNotification.hpp.

6.25.3.3 CModuleNotification::CModuleNotification (const CModuleNotification &) [private]

No copying defined.

6.25.4 Member Function Documentation

6.25.4.1 CModuleNotification::ENotificationType CModuleNotification::getType () const [inline]

Returns notification type.

Definition at line 136 of file ModuleNotification.hpp.

References m_eType.

6.25.4.2 const TPtrCodeBlock & CModuleNotification::getCode () const [inline]

Returns the contained code block.

Definition at line 142 of file ModuleNotification.hpp.

References m_pCode.

6.25.4.3 const TPtrIdentifier & CModuleNotification::getIdentifier () const [inline]

Returns the contained module identifier.

Definition at line 148 of file ModuleNotification.hpp.

References m_pIdentifier.

6.25.4.4 const TSrcLocation & CModuleNotification::getSrcPosition () const [inline]

Returns the source code position of the notification object.

Definition at line 154 of file ModuleNotification.hpp.

References m_srcLocation.

6.25.4.5 CModuleNotification& CModuleNotification::operator= (const CModuleNotification &) [private]

No assignment defined.

6.25.5 Member Data Documentation

6.25.5.1 ENotificationType CModuleNotification::m_eType [private]

Notification type.

Definition at line 94 of file ModuleNotification.hpp.

Referenced by `getType()`.

6.25.5.2 TPtrIdentifier CModuleNotification::m_pIdentifier [private]

Notified module identifier.

Definition at line 97 of file `ModuleNotification.hpp`.

Referenced by `getIdentifier()`.

6.25.5.3 TPtrCodeBlock CModuleNotification::m_pCode [private]

Code block of the notification declaration.

Definition at line 100 of file `ModuleNotification.hpp`.

Referenced by `getCode()`.

6.25.5.4 TSrcLocation CModuleNotification::m_srcLocation [private]

Source location of the notification.

Definition at line 103 of file `ModuleNotification.hpp`.

Referenced by `getSrcPosition()`.

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

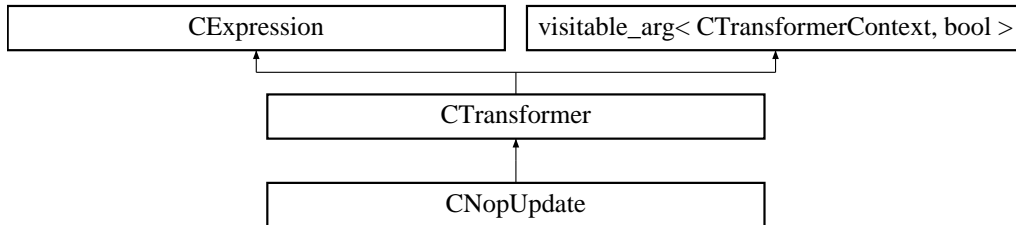
- **ModuleNotification.hpp**

6.26 CNoUpdate Class Reference

Empty update class implementation.

```
#include <TrivialUpdate.hpp>
```

Inheritance diagram for CNoUpdate::



Public Member Functions

- **CNoUpdate ()**
Default constructor.
- **virtual ~CNoUpdate ()**
Virtual destructor.

Private Member Functions

- **CNoUpdate (const CNoUpdate &)**
No copying defined.
- **CNoUpdate & operator= (const CNoUpdate &)**
No assignment defined.

6.26.1 Detailed Description

Empty update class implementation.

The NOP mental state transformer

Definition at line 42 of file TrivialUpdate.hpp.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 CNoUpdate::CNoUpdate () [inline]

Default constructor.

Definition at line 126 of file TrivialUpdate.hpp.

6.26.2.2 CNoUpdate::~~CNoUpdate () [inline, virtual]

Virtual destructor.

Definition at line 131 of file TrivialUpdate.hpp.

6.26.2.3 CNoUpdate::CNoUpdate (const CNoUpdate &) [private]

No copying defined.

6.26.3 Member Function Documentation**6.26.3.1 CNoUpdate& CNoUpdate::operator= (const CNoUpdate &) [private]**

No assignment defined.

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

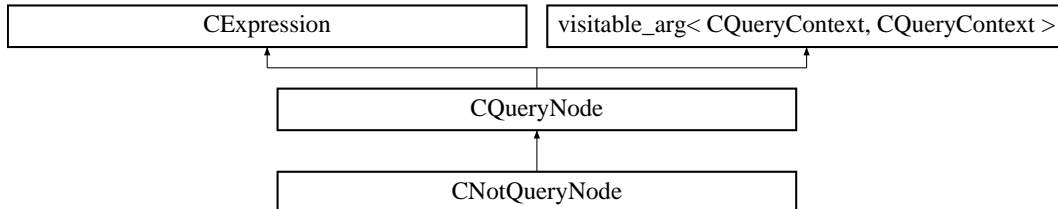
- **TrivialUpdate.hpp**

6.27 CNotQueryNode Class Reference

Query node for NOT logical modifier of a query expression.

```
#include <NotQueryNode.hpp>
```

Inheritance diagram for CNotQueryNode::



Public Member Functions

- **CNotQueryNode** (TPtrQueryNode pQueryNode)
Default constructor.
- virtual **~CNotQueryNode** ()
Virtual destructor.
- **TPtrQueryNode getNode** () const
Returns the negated node.

Private Member Functions

- **CNotQueryNode** (const CNotQueryNode &)
No copying defined.
- **CNotQueryNode & operator=** (const CNotQueryNode &)
No assignment defined.

Private Attributes

- **TPtrQueryNode m_pNegatedNode**
Modified query node.

6.27.1 Detailed Description

Query node for NOT logical modifier of a query expression.

Holds a single query subnode.

The node evaluates to TRUE when the contained subnoded evaluates to FALSE end vice versa.

Additionally, no substitution set is returned.

Definition at line 47 of file NotQueryNode.hpp.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 CNotQueryNode::CNotQueryNode (TPtrQueryNode *pQueryNode*) [inline]

Default constructor.

Definition at line 96 of file NotQueryNode.hpp.

6.27.2.2 CNotQueryNode::~CNotQueryNode () [inline, virtual]

Virtual destructor.

Definition at line 102 of file NotQueryNode.hpp.

6.27.2.3 CNotQueryNode::CNotQueryNode (const CNotQueryNode &) [private]

No copying defined.

6.27.3 Member Function Documentation

6.27.3.1 TPtrQueryNode CNotQueryNode::getNode () const [inline]

Returns the negated node.

Definition at line 107 of file NotQueryNode.hpp.

References `m_pNegatedNode`.

Referenced by `CTraverser::Visit()`.

6.27.3.2 CNotQueryNode& CNotQueryNode::operator= (const CNotQueryNode &) [private]

No assignment defined.

6.27.4 Member Data Documentation

6.27.4.1 TPtrQueryNode CNotQueryNode::m_pNegatedNode [private]

Modified query node.

Definition at line 77 of file NotQueryNode.hpp.

Referenced by `getNode()`.

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

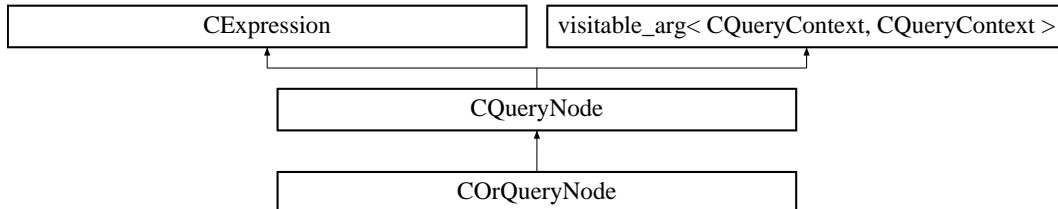
- `NotQueryNode.hpp`

6.28 COrQueryNode Class Reference

Query node for OR logical connector of queries.

```
#include <OrQueryNode.hpp>
```

Inheritance diagram for COrQueryNode::



Public Member Functions

- **COrQueryNode** (TPtrQueryNode pLeftNode, TPtrQueryNode pRightNode)
Default constructor.
- virtual **~COrQueryNode** ()
Virtual destructor.
- **TPtrQueryNode** getLeftNode () const
Returns the left node.
- **TPtrQueryNode** getRightNode () const
Returns rthe right subnode.

Private Member Functions

- **COrQueryNode** (const COrQueryNode &)
No copying defined.
- **COrQueryNode & operator=** (const COrQueryNode &)
No assignment defined.

Private Attributes

- **TPtrQueryNode** m_pLeftNode
Left hand query node.
- **TPtrQueryNode** m_pRightNode
Right hand query node.

6.28.1 Detailed Description

Query node for OR logical connector of queries.

Holds two query subnodes.

The node evaluates to TRUE when both subnodes evaluate to true as well. The resulting substitution is the constructed according to the truth value of the subnodes:

- if only one of the subnodes is TRUE, its substitution set is returned
- if both of them are TRUE then the combination substitution set of the right and the left subnodes is returned.

Additionally, the right subnode takes as input the output substitution set of the left one.

Definition at line 53 of file OrQueryNode.hpp.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 COrQueryNode::COrQueryNode (TPtrQueryNode *pLeftNode*, TPtrQueryNode *pRightNode*) [inline]

Default constructor.

Definition at line 108 of file OrQueryNode.hpp.

6.28.2.2 COrQueryNode::~~COrQueryNode () [inline, virtual]

Virtual destructor.

Definition at line 115 of file OrQueryNode.hpp.

6.28.2.3 COrQueryNode::COrQueryNode (const COrQueryNode &) [private]

No copying defined.

6.28.3 Member Function Documentation

6.28.3.1 TPtrQueryNode COrQueryNode::getLeftNode () const [inline]

Returns the left node.

Definition at line 120 of file OrQueryNode.hpp.

References `m_pLeftNode`.

Referenced by `CTraverser::Visit()`.

6.28.3.2 TPtrQueryNode COrQueryNode::getRightNode () const [inline]

Returns the right subnode.

Definition at line 126 of file OrQueryNode.hpp.

References `m_pRightNode`.

Referenced by CTraverser::Visit().

6.28.3.3 COrQueryNode& COrQueryNode::operator= (const COrQueryNode &) [private]

No assignment defined.

6.28.4 Member Data Documentation

6.28.4.1 TPtrQueryNode COrQueryNode::m_pLeftNode [private]

Left hand query node.

Definition at line 85 of file OrQueryNode.hpp.

Referenced by getLeftNode().

6.28.4.2 TPtrQueryNode COrQueryNode::m_pRightNode [private]

Right hand query node.

Definition at line 88 of file OrQueryNode.hpp.

Referenced by getRightNode().

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

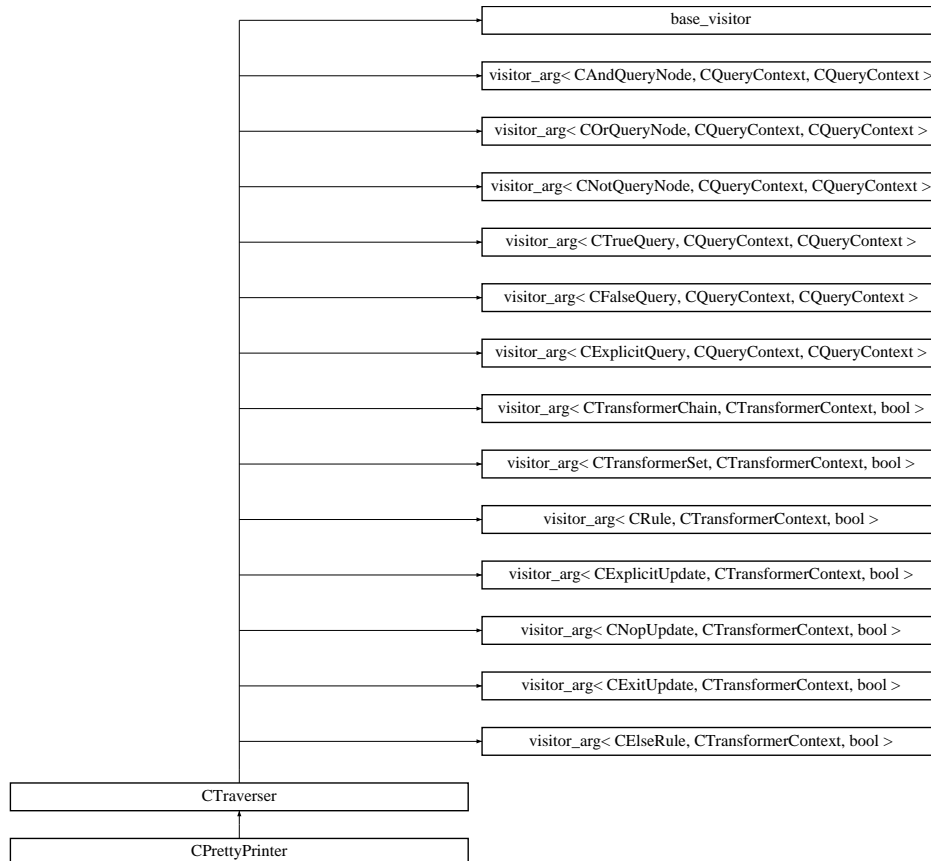
- **OrQueryNode.hpp**

6.29 CPrettyPrinter Class Reference

Jazzyk program pretty printing service class.

```
#include <PrettyPrinter.hpp>
```

Inheritance diagram for CPrettyPrinter::



Public Member Functions

- **CPrettyPrinter** (TPtrProgram pProgram, std::ostream &str=std::cout)
Constructor with the program to print.
- virtual **~CPrettyPrinter** ()
Virtual destructor.
- virtual void **dump** ()
Prints the program to the stream.

Protected Member Functions

- virtual **CQueryContext & PreAction** (CAndQueryNode &, CQueryContext &)

*Inherited virtuals for handling various types of the **CExpression** (p. 62) hierarchy.*

- virtual **CQueryContext & PreAction** (**COrQueryNode &**, **CQueryContext &**)
- virtual **CQueryContext & PreAction** (**CNotQueryNode &**, **CQueryContext &**)
- virtual **CQueryContext & PreAction** (**CTrueQuery &**, **CQueryContext &**)
- virtual **CQueryContext & PreAction** (**CFalseQuery &**, **CQueryContext &**)
- virtual **CQueryContext & PreAction** (**CExplicitQuery &**, **CQueryContext &**)
- virtual void **PreAction** (**CTransformerChain &**, **CTransformerContext &**)
- virtual void **PreAction** (**CTransformerSet &**, **CTransformerContext &**)
- virtual void **PreAction** (**CRule &**, **CTransformerContext &**)
- virtual void **PreAction** (**CExplicitUpdate &**, **CTransformerContext &**)
- virtual void **PreAction** (**CNoUpdate &**, **CTransformerContext &**)
- virtual void **PreAction** (**CElseRule &**, **CTransformerContext &**)
- virtual void **DefaultPostAction** ()

Private Member Functions

- **std::string indent** ()
Prints the indentation at the beginning of the line.
- **CPrettyPrinter** (const **CPrettyPrinter &**)
No copying defined.
- **CPrettyPrinter & operator=** (const **CPrettyPrinter &**)
No assignment defined.

Private Attributes

- **TPtrProgram m_pProgram**
Program to print.
- **std::ostream & m_outStream**
Output stream to be used.
- **unsigned int m_level**
Indentation level of the structure.

6.29.1 Detailed Description

Jazzyk program pretty printig service class.

Pretty printer prints the parsed program in a structured manner with adjusted indentation for each level of the program nested structure.

The program is printed to the given output stream. If no output stream is provided, `std::cout` is used.

Definition at line 51 of file `PrettyPrinter.hpp`.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 CPrettyPrinter::CPrettyPrinter (TPtrProgram *pProgram*, std::ostream & *str* = std::cout)

Constructor with the program to print.

Definition at line 39 of file PrettyPrinter.cpp.

6.29.2.2 CPrettyPrinter::~~CPrettyPrinter () [virtual]

Virtual destructor.

Definition at line 47 of file PrettyPrinter.cpp.

6.29.2.3 CPrettyPrinter::CPrettyPrinter (const CPrettyPrinter &) [private]

No copying defined.

6.29.3 Member Function Documentation

6.29.3.1 void CPrettyPrinter::dump () [virtual]

Prints the program to the stream.

Definition at line 64 of file PrettyPrinter.cpp.

References `m_pProgram`, and `CTraverser::traverse()`.

6.29.3.2 CQueryContext & CPrettyPrinter::PreAction (CAndQueryNode &, CQueryContext & *ctx*) [protected, virtual]

Inherited virtuals for handling various types of the **CExpression** (p. 62) hierarchy.

Reimplemented from **CTraverser** (p. 190).

Definition at line 70 of file PrettyPrinter.cpp.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.3 CQueryContext & CPrettyPrinter::PreAction (COrQueryNode &, CQueryContext & *ctx*) [protected, virtual]

Reimplemented from **CTraverser** (p. 190).

Definition at line 79 of file PrettyPrinter.cpp.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.4 CQueryContext & CPrettyPrinter::PreAction (CNotQueryNode &, CQueryContext & *ctx*) [protected, virtual]

Reimplemented from **CTraverser** (p. 190).

Definition at line 88 of file PrettyPrinter.cpp.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.5 `CQueryContext & CPrettyPrinter::PreAction (CTrueQuery &, CQueryContext & ctx)` [protected, virtual]

Reimplemented from `CTraverser` (p. 191).

Definition at line 97 of file `PrettyPrinter.cpp`.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.6 `CQueryContext & CPrettyPrinter::PreAction (CFalseQuery &, CQueryContext & ctx)` [protected, virtual]

Reimplemented from `CTraverser` (p. 191).

Definition at line 106 of file `PrettyPrinter.cpp`.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.7 `CQueryContext & CPrettyPrinter::PreAction (CExplicitQuery & quest, CQueryContext & ctx)` [protected, virtual]

Reimplemented from `CTraverser` (p. 191).

Definition at line 115 of file `PrettyPrinter.cpp`.

References `CExplicitQuery::getFreeVariables()`, `indent()`, `m_level`, and `m_outStream`.

6.29.3.8 `void CPrettyPrinter::PreAction (CTransformerChain &, CTransformerContext &)` [protected, virtual]

Reimplemented from `CTraverser` (p. 191).

Definition at line 137 of file `PrettyPrinter.cpp`.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.9 `void CPrettyPrinter::PreAction (CTransformerSet &, CTransformerContext &)` [protected, virtual]

Reimplemented from `CTraverser` (p. 191).

Definition at line 144 of file `PrettyPrinter.cpp`.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.10 `void CPrettyPrinter::PreAction (CRule &, CTransformerContext &)` [protected, virtual]

Reimplemented from `CTraverser` (p. 191).

Definition at line 151 of file `PrettyPrinter.cpp`.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.11 `void CPrettyPrinter::PreAction (CExplicitUpdate & guest, CTransformerContext & ctx)` [protected, virtual]

Reimplemented from `CTraverser` (p.192).

Definition at line 158 of file `PrettyPrinter.cpp`.

References `CExplicitUpdate::getUsedVariables()`, `indent()`, `m_level`, and `m_outStream`.

6.29.3.12 `void CPrettyPrinter::PreAction (CNopUpdate &, CTransformerContext &)` [protected, virtual]

Reimplemented from `CTraverser` (p.192).

Definition at line 188 of file `PrettyPrinter.cpp`.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.13 `void CPrettyPrinter::PreAction (CElseRule &, CTransformerContext &)` [protected, virtual]

Reimplemented from `CTraverser` (p.192).

Definition at line 195 of file `PrettyPrinter.cpp`.

References `indent()`, `m_level`, and `m_outStream`.

6.29.3.14 `void CPrettyPrinter::DefaultPostAction ()` [protected, virtual]

Reimplemented from `CTraverser` (p.195).

Definition at line 202 of file `PrettyPrinter.cpp`.

References `m_level`.

6.29.3.15 `std::string CPrettyPrinter::indent ()` [private]

Prints the indentation at the beginning of the line.

Definition at line 52 of file `PrettyPrinter.cpp`.

References `m_level`.

Referenced by `PreAction()`.

6.29.3.16 `CPrettyPrinter& CPrettyPrinter::operator= (const CPrettyPrinter &)` [private]

No assignment defined.

6.29.4 Member Data Documentation

6.29.4.1 `TPtrProgram CPrettyPrinter::m_pProgram` [private]

Program to print.

Definition at line 98 of file PrettyPrinter.hpp.

Referenced by `dump()`.

6.29.4.2 `std::ostream& CPrettyPrinter::m_outStream` [private]

Output stream to be used.

Definition at line 101 of file PrettyPrinter.hpp.

Referenced by `PreAction()`.

6.29.4.3 `unsigned int CPrettyPrinter::m_level` [private]

Indentation level of the structure.

Definition at line 104 of file PrettyPrinter.hpp.

Referenced by `DefaultPostAction()`, `indent()`, and `PreAction()`.

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

- **PrettyPrinter.hpp**
- **PrettyPrinter.cpp**

6.30 CProgram Class Reference

The Jazzyk program backend structure.

```
#include <Program.hpp>
```

Public Member Functions

- **CProgram** ()
Default constructor.
- virtual **~CProgram** ()
Virtual destructor.
- void **add** (TPtrModuleDeclaration pDecl)
Adds a new module declaration to the program.
- void **add** (TPtrTransformer pTransformer)
Adds a new mental state transformer to the program.
- void **add** (const TPtrModuleNotification &pNotification)
Adds a notification to the module.
- bool **isModuleDeclared** (const TPtrIdentifier &pIdentifier) const
Returns true if there is an already declared module with the identifier.
- TPtrModuleDeclaration **getModuleDeclaration** (const TPtrIdentifier &id) const
Returns pointer to the declaration of the module 'id'.
- TPtrTransformer **getTransformer** () const
Returns the pointer to the top-level mental state transformer.
- void **loadModules** ()
Loads all the module declarations.
- void **unloadModules** ()
Unloads all the module declarations.
- void **cycle** ()
Notifies all the modules about the new deliberation cycle.
- void **retrieveInterfaces** ()
Retrieves the loaded module query/update interfaces.

Private Member Functions

- **CProgram** (const CProgram &)
No copying defined.

- **CProgram & operator= (const CProgram &)**

No assignment defined.

Private Attributes

- **TModuleDeclCollection m_moduleDeclarations**

Collection of module declarations.

- **TPtrTransformerSet m_pTransformerCollection**

Top level mental state transformer.

6.30.1 Detailed Description

The Jazzyk program backend structure.

The Jazzyk program consists of a set of a collection of module declarations and a collection of mental state transformers in the form of a set of transformers (semicolon separated statements).

The collection of mental state transformers is held in a top-level mental state transformer and build a tree structure of sub-transformers.

The program structure is generated during the parsing stage of the compiler run. Later it is checked for validity and finally the interpreter interprets it.

On destruction, the program unloads all the modules it loaded.

Definition at line 56 of file Program.hpp.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 CProgram::CProgram ()

Default constructor.

Definition at line 46 of file Program.cpp.

6.30.2.2 CProgram::~~CProgram () [virtual]

Virtual destructor.

Definition at line 53 of file Program.cpp.

References unloadModules().

6.30.2.3 CProgram::CProgram (const CProgram &) [private]

No copying defined.

6.30.3 Member Function Documentation

6.30.3.1 void CProgram::add (TPtrModuleDeclaration *pDecl*)

Adds a new module declaration to the program.

Definition at line 59 of file Program.cpp.

References CError::JZCOMPILER_VALIDITY_MODULE_REDEFINED, and m_ - moduleDeclarations.

6.30.3.2 void CProgram::add (TPtrTransformer *pTransformer*)

Adds a new mental state transformer to the program.

Definition at line 76 of file Program.cpp.

References m_pTransformerCollection.

6.30.3.3 void CProgram::add (const TPtrModuleNotification & *pNotification*)

Adds a notification to the module.

If the module was not previously declared, exception is thrown.

Definition at line 89 of file Program.cpp.

References getModuleDeclaration(), isModuleDeclared(), and CError::JZPARSER_PARSE_ERROR.

6.30.3.4 bool CProgram::isModuleDeclared (const TPtrIdentifier & *pIdentifier*) const

Returns true if there is an already declared module with the identifier.

Definition at line 82 of file Program.cpp.

References m_ moduleDeclarations.

Referenced by add().

6.30.3.5 TPtrModuleDeclaration CProgram::getModuleDeclaration (const TPtrIdentifier & *id*) const

Returns pointer to the declaration of the module 'id'.

Definition at line 110 of file Program.cpp.

References m_ moduleDeclarations.

Referenced by add().

6.30.3.6 TPtrTransformer CProgram::getTransformer () const

Returns the pointer to the top-level mental state transformer.

Definition at line 116 of file Program.cpp.

References `m_pTransformerCollection`.

6.30.3.7 void CProgram::loadModules ()

Loads all the module declarations.

Definition at line 122 of file `Program.cpp`.

References `CModuleManager::Instance()`, `CModuleManager::load()`, and `m_moduleDeclarations`.

6.30.3.8 void CProgram::unloadModules ()

Unloads all the module declarations.

Definition at line 133 of file `Program.cpp`.

References `CModuleManager::Instance()`, `m_moduleDeclarations`, and `CModuleManager::unload()`.

Referenced by `~CProgram()`.

6.30.3.9 void CProgram::cycle ()

Notifies all the modules about the new deliberation cycle.

Definition at line 144 of file `Program.cpp`.

References `CModuleManager::cycle()`, `CModuleManager::Instance()`, and `m_moduleDeclarations`.

6.30.3.10 void CProgram::retrieveInterfaces ()

Retrieves the loaded module query/update interfaces.

The retrieved interfaces from the KR modules are stroed in the corresponding module declaration instances.

6.30.3.11 CProgram& CProgram::operator= (const CProgram &) [private]

No assignment defined.

6.30.4 Member Data Documentation

6.30.4.1 TModuleDeclCollection CProgram::m_moduleDeclarations [private]

Collection of module declarations.

Definition at line 117 of file `Program.hpp`.

Referenced by `add()`, `cycle()`, `getModuleDeclaration()`, `isModuleDeclared()`, `loadModules()`, and `unloadModules()`.

6.30.4.2 TPtrTransformerSet CProgram::m_pTransformerCollection [private]

Top level mental state transformer.

Definition at line 120 of file Program.hpp.

Referenced by add(), and getTransformer().

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

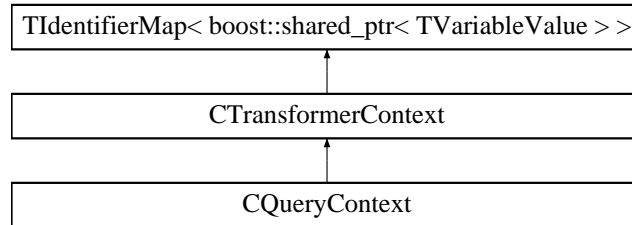
- **Program.hpp**
- **Program.cpp**

6.31 CQueryContext Class Reference

Interpreter context used while traversing the query subtree.

```
#include <Context.hpp>
```

Inheritance diagram for CQueryContext::



Public Types

- enum **EQueryReturnValue** { **QFALSE**, **QTRUE**, **QUNDEFINED** }
Query return values.

Public Member Functions

- **CQueryContext** (const **CTransformerContext** &ctx)
Basic constructor.
- virtual **~CQueryContext** ()
Context is copy-constructible.
- **CQueryContext** & **operator=** (const **CQueryContext** &)
Assignment operator.
- **EQueryReturnValue** **getResult** () const
Returns the query evaluation result.
- void **setResult** (**EQueryReturnValue** val)
Sets the truth value of the context.
- void **resetResult** ()
Resets result to UNDEFINED turth value.
- bool **isTrue** () const
Returns true when the result is strictly TRUE.
- bool **isTrueW** () const
Returns true when the query result is TRUE, or UNDEFINED (weak true).
- bool **isFalse** () const
Returns true when the query result is strictly FALSE.

- `bool isFalseW () const`
Returns true when the query result is FALSE, or UNDEFINED (weak false).
- `virtual CQueryContext & operator &= (const EQueryReturnValue &rhs)`
Updates the context result in a 3-valued AND manner.
- `virtual CQueryContext & operator |= (const EQueryReturnValue &rhs)`
Updates the context result in a 3-valued OR manner.
- `virtual CQueryContext & negate ()`
Updates the context result in a 3-valued NOT manner.

Private Member Functions

- `CQueryContext ()`
Default constructor.

Private Attributes

- `EQueryReturnValue m_result`
Query context return value.

6.31.1 Detailed Description

Interpreter context used while traversing the query subtree.

When created it derives its internal structures from the currently valid `CTransformer` (p. 172) context.

The query context is also used as a return value of Accept/Visit dispatch. It holds mainly the result of the query evaluation.

Finally its held variable substitution is supposed to be unified with the variable substitution held by the parent transformer context (e.g. in the interpretation phase).

Note: The instances are copied extensively during the program tree traversal, but that allows passing the substitution between the separate parts of the query expression finally resulting in a valid variable substitution.

Definition at line 147 of file Context.hpp.

6.31.2 Member Enumeration Documentation

6.31.2.1 `enum CQueryContext::EQueryReturnValue`

Query return values.

Enumerator:

QFALSE

QTRUE

QUNDEFINED

Definition at line 177 of file Context.hpp.

6.31.3 Constructor & Destructor Documentation

6.31.3.1 CQueryContext::CQueryContext (const CTransformerContext & *ctx*)

Basic constructor.

Initializes the shared data for traversing the whole query subtree. It is a specialized copy constructor of the **CTransformer** (p. 172) parent class.

Should be used only at the very top node of a query subtree.

Definition at line 98 of file Context.cpp.

6.31.3.2 CQueryContext::~~CQueryContext () [virtual]

Context is copy-constructible.

Let the compiler create a default copy constructor. Virtual destructor

Definition at line 105 of file Context.cpp.

6.31.3.3 CQueryContext::CQueryContext () [private]

Default constructor.

The method should be never called and **visitor** (p. 218) should throw a `runtime_error` exception in the case it is needed.

Note: It is never defined, although it might break some compilers.

6.31.4 Member Function Documentation

6.31.4.1 CQueryContext & CQueryContext::operator= (const CQueryContext & *ctx*)

Assignment operator.

Only the return result and the variable substitution are copied, not the context transformer reference.

That's because only the first top-level query context is created using a constructor from context and all the others are copied from parent query contexts. I.e. all the query contexts of one large query expression share the reference to one single transformer context object.

Definition at line 171 of file Context.cpp.

References `m_result`.

6.31.4.2 CQueryContext::EQueryReturnValue CQueryContext::getResult () const [inline]

Returns the query evaluation result.

Definition at line 281 of file Context.hpp.

References m_result.

Referenced by CTraverser::Visit().

6.31.4.3 void CQueryContext::setResult (EQueryReturnValue val) [inline]

Sets the truth value of the context.

Definition at line 287 of file Context.hpp.

References m_result.

Referenced by CInterpreter::PostAction(), resetResult(), and CTraverser::Visit().

6.31.4.4 void CQueryContext::resetResult () [inline]

Resets result to UNDEFINED turth value.

Definition at line 293 of file Context.hpp.

References QUNDEFINED, and setResult().

Referenced by CTraverser::Visit().

6.31.4.5 bool CQueryContext::isTrue () const [inline]

Returns true when the result is strictly TRUE.

Definition at line 299 of file Context.hpp.

References m_result, and QTRUE.

6.31.4.6 bool CQueryContext::isTrueW () const [inline]

Returns true when the query result is TRUE, or UNDEFINED (weak true).

Definition at line 305 of file Context.hpp.

References m_result, QTRUE, and QUNDEFINED.

Referenced by CTraverser::Visit().

6.31.4.7 bool CQueryContext::isFalse () const [inline]

Returns true when the query result is strictly FALSE.

Definition at line 311 of file Context.hpp.

References m_result, and QFALSE.

6.31.4.8 `bool CQueryContext::isFalseW () const [inline]`

Returns true when the query result is FALSE, or UNDEFINED (weak false).

Definition at line 317 of file Context.hpp.

References `m_result`, `QFALSE`, and `QUNDEFINED`.

Referenced by `CTraverser::Visit()`.

6.31.4.9 `CQueryContext & CQueryContext::operator &= (const EQueryReturnValue & rhs) [virtual]`

Updates the context result in a 3-valued AND manner.

Truth values: (T,T)=T, (T,F)=F, (F,T)=F, (F,F)=F (U,*)=U, (*,U)=U

Note: Does not affect the contained variable substitution!

Definition at line 109 of file Context.cpp.

References `m_result`, `QFALSE`, `QTRUE`, and `QUNDEFINED`.

6.31.4.10 `CQueryContext & CQueryContext::operator |= (const EQueryReturnValue & rhs) [virtual]`

Updates the context result in a 3-valued OR manner.

Truth values: (T,T)=T, (T,F)=T, (F,T)=T, (F,F)=T (U,*)=U, (*,U)=U

Note: Does not affect the contained variable substitution!

Definition at line 129 of file Context.cpp.

References `m_result`, `QFALSE`, `QTRUE`, and `QUNDEFINED`.

6.31.4.11 `CQueryContext & CQueryContext::negate () [virtual]`

Updates the context result in a 3-valued NOT manner.

Truth values: (T)=F, (F)=T, (U)=U

Note: Clears the contained variable substitution!

Definition at line 149 of file Context.cpp.

References `m_result`, `QFALSE`, `QTRUE`, and `QUNDEFINED`.

Referenced by `CTraverser::Visit()`.

6.31.5 Member Data Documentation**6.31.5.1** `EQueryReturnValue CQueryContext::m_result [private]`

Query context return value.

Definition at line 259 of file Context.hpp.

Referenced by `getResult()`, `isFalse()`, `isFalseW()`, `isTrue()`, `isTrueW()`, `negate()`, `operator &=()`, `operator=()`, `operator|=()`, and `setResult()`.

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

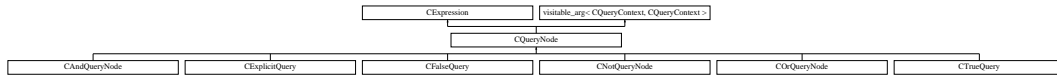
- **Context.hpp**
- **Context.cpp**

6.32 CQueryNode Class Reference

Query node abstract base class.

```
#include <QueryNode.hpp>
```

Inheritance diagram for CQueryNode::



Public Member Functions

- **CQueryNode** ()
Default constructor.
- virtual **~CQueryNode** ()=0
Pure virtual destructor.

Private Member Functions

- **CQueryNode** (const **CQueryNode** &)
No copying defined.
- **CQueryNode** & **operator=** (const **CQueryNode** &)
No assignment defined.

6.32.1 Detailed Description

Query node abstract base class.

Query nodes are specialized expressions, which compose a tree structure of queries which can be hierarchically evaluated according to 1st order logic rules.

CQueryNode (p.136) hierarchy is **visitable** (p.214) with **CQueryContext** (p.130) instance as an argument asd well as a return value.

Definition at line 49 of file QueryNode.hpp.

6.32.2 Constructor & Destructor Documentation

6.32.2.1 CQueryNode::CQueryNode () [inline]

Default constructor.

Definition at line 87 of file QueryNode.hpp.

6.32.2.2 CQueryNode::~~CQueryNode () [inline, pure virtual]

Pure virtual destructor.

Definition at line 92 of file QueryNode.hpp.

6.32.2.3 CQueryNode::CQueryNode (const CQueryNode &) [private]

No copying defined.

6.32.3 Member Function Documentation**6.32.3.1 CQueryNode& CQueryNode::operator= (const CQueryNode &) [private]**

No assignment defined.

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

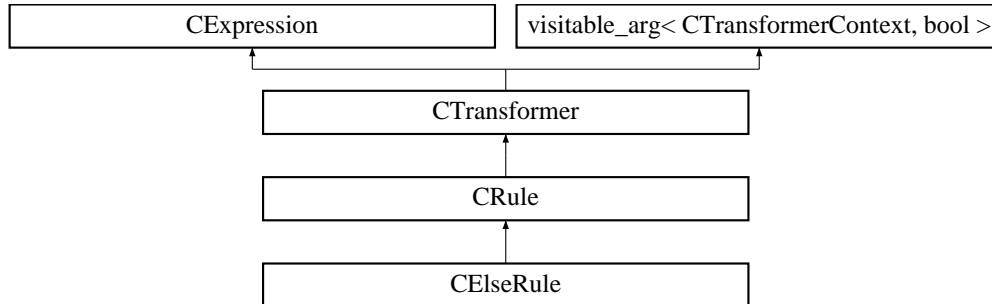
- **QueryNode.hpp**

6.33 CRule Class Reference

If-then rule mental state transformer.

```
#include <Rule.hpp>
```

Inheritance diagram for CRule::



Public Member Functions

- **CRule** (**TPtrQueryNode** pLHS, **TPtrTransformer** pRHS)
Default constructor.
- virtual **~CRule** ()
Virtual destructor.
- **TPtrQueryNode** **getQuery** () const
Returns pointer to the left hand side query.
- **TPtrTransformer** **getThenTransformer** () const
Returns pointer to the right hand side transformer.

Protected Attributes

- **TPtrQueryNode** **m_pQueryList**
Left hand side query.
- **TPtrTransformer** **m_pThenTransformer**
Right hand side mental state transformer.

Private Member Functions

- **CRule** (const **CRule** &)
No copying defined.
- **CRule** & **operator=** (const **CRule** &)
No assignment defined.

6.33.1 Detailed Description

If-then rule mental state transformer.

The rule mental state transformer stands for the full rule with both left hand as well as right hand side. The left hand side is a query, while the right hand side is a mental state transformer.

Definition at line 45 of file Rule.hpp.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 CRule::CRule (TPtrQueryNode *pLHS*, TPtrTransformer *pRHS*) [inline]

Default constructor.

Definition at line 99 of file Rule.hpp.

6.33.2.2 CRule::~CRule () [inline, virtual]

Virtual destructor.

Definition at line 106 of file Rule.hpp.

6.33.2.3 CRule::CRule (const CRule &) [private]

No copying defined.

6.33.3 Member Function Documentation

6.33.3.1 TPtrQueryNode CRule::getQuery () const [inline]

Returns pointer to the left hand side query.

Definition at line 111 of file Rule.hpp.

References m_pQueryList.

Referenced by CTraverser::Visit().

6.33.3.2 TPtrTransformer CRule::getThenTransformer () const [inline]

Returns pointer to the right hand side transformer.

Definition at line 117 of file Rule.hpp.

References m_pThenTransformer.

Referenced by CTraverser::Visit().

6.33.3.3 CRule& CRule::operator= (const CRule &) [private]

No assignment defined.

6.33.4 Member Data Documentation

6.33.4.1 TPtrQueryNode CRule::m_pQueryList [protected]

Left hand side query.

Definition at line 77 of file Rule.hpp.

Referenced by `getQuery()`.

6.33.4.2 TPtrTransformer CRule::m_pThenTransformer [protected]

Right hand side mental state transformer.

Definition at line 80 of file Rule.hpp.

Referenced by `getThenTransformer()`.

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

- **Rule.hpp**

6.34 CShmClientMgr Class Reference

Client side interprocess communication manager.

```
#include <ShmClientMgr.hpp>
```

Public Member Functions

- void **initialize** (const std::string &szModuleID)
Initializes the client side shared memory manager.
- void **finalize** ()
Closes the shared memory and releases the handles to named conditions, mutexes etc.
- void **notifyReady** ()
Notifies the server (main) process that the child (client) is ready.
- void **notifyResponse** ()
Notifies the server (main) process that the response is present in the shared memory and should be processed.
- void **waitForRequest** (ShmemLock &lock)
Waits for a request sync signal from the main process.
- SModuleRequest * **retrieveRequest** ()
Retrieves the request data structure from the shared memory.
- SModuleResponse * **createResponse** ()
Creates a response data structure in the shared memory.

Static Public Member Functions

- static CShmClientMgr & **Instance** ()
Returns the singleton instance.

Private Member Functions

- CShmClientMgr ()
Default constructor.
- virtual ~CShmClientMgr ()
Virtual destructor.
- CShmClientMgr (const CShmClientMgr &)
No copying defined.
- CShmClientMgr & **operator=** (const CShmClientMgr &)
No assignment defined.

Private Attributes

- **CShmemManager::TPtrSModuleSync m_pModuleSync**
Synchronisation conditions.

6.34.1 Detailed Description

Client side interprocess communication manager.

A singleton interface to the **CShmemManager** (p. 145) to be used in the subprocesses.

It stores a set of synchronisation conditions for the subprocess.

Definition at line 44 of file ShmClientMgr.hpp.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 CShmClientMgr::CShmClientMgr () [private]

Default constructor.

Definition at line 38 of file ShmClientMgr.cpp.

6.34.2.2 CShmClientMgr::~~CShmClientMgr () [private, virtual]

Virtual destructor.

Definition at line 44 of file ShmClientMgr.cpp.

6.34.2.3 CShmClientMgr::CShmClientMgr (const CShmClientMgr &) [private]

No copying defined.

6.34.3 Member Function Documentation

6.34.3.1 CShmClientMgr & CShmClientMgr::Instance () [static]

Returns the singleton instance.

Definition at line 49 of file ShmClientMgr.cpp.

Referenced by **CSingleModule::extractRequest()**, **CSingleModule::handleModuleError()**, **CSingleModule::run()**, and **CSingleModule::writeResponse()**.

6.34.3.2 void CShmClientMgr::initialize (const std::string & szModuleID)

Initializes the client side shared memory manager.

Opens the shared memory objects. To map the named synchronisation conditions, it uses the module ID string provided as an argument.

Definition at line 57 of file ShmClientMgr.cpp.

References CShmemManager::initializeChildProcess(), CShmemManager::Instance(), and m_pModuleSync.

Referenced by CSingleModule::run().

6.34.3.3 void CShmClientMgr::finalize ()

Closes the shared memory and releases the handles to named conditions, mutexes etc.

Definition at line 69 of file ShmClientMgr.cpp.

References CShmemManager::finalizeChildProcess(), CShmemManager::Instance(), and m_pModuleSync.

Referenced by CSingleModule::run().

6.34.3.4 void CShmClientMgr::notifyReady ()

Notifies the server (main) process that the child (client) is ready.

Notification is complementary to CShmServerMgr::waitForReady() (p. 162) method.

Definition at line 79 of file ShmClientMgr.cpp.

References m_pModuleSync.

Referenced by CSingleModule::run().

6.34.3.5 void CShmClientMgr::notifyResponse ()

Notifies the server (main) process that the response is present in the shared memory and should be processed.

Notification is complementary to CShmServerMgr::waitForResponse() (p. 162) method.

Definition at line 85 of file ShmClientMgr.cpp.

References m_pModuleSync.

Referenced by CSingleModule::handleModuleError(), and CSingleModule::run().

6.34.3.6 void CShmClientMgr::waitForRequest (ShmemLock & lock)

Waits for a request sync signal from the main process.

Takes a lock object of the scope the caller is in.

This wait is complementary to CShmServerMgr::notifyRequest() (p. 162) method.

Definition at line 91 of file ShmClientMgr.cpp.

References m_pModuleSync.

Referenced by CSingleModule::run().

6.34.3.7 SModuleRequest * CShmClientMgr::retrieveRequest ()

Retrieves the request data structure from the shared memory.

Returns a plain pointer to the extracted request data structure. The user should not destroy the shared object. The server is responsible for its correct destruction.

Definition at line 97 of file ShmClientMgr.cpp.

References CShmemManager::Instance(), and CShmemManager::retrieveRequest().

Referenced by CSingleModule::extractRequest().

6.34.3.8 SModuleResponse * CShmClientMgr::createResponse ()

Creates a response data structure in the shared memory.

Returns a plain pointer to the allocated data structure. The client shouldn't destroy the object after it communicates as it is left for the server side to destroy it correctly.

Definition at line 103 of file ShmClientMgr.cpp.

References CShmemManager::allocateResponse(), and CShmemManager::Instance().

Referenced by CSingleModule::writeResponse().

6.34.3.9 CShmClientMgr& CShmClientMgr::operator= (const CShmClientMgr &) [private]

No assignment defined.

6.34.4 Member Data Documentation

6.34.4.1 CShmemManager::TPtrSModuleSync CShmClientMgr::m_pModuleSync [private]

Synchronisation conditions.

Definition at line 114 of file ShmClientMgr.hpp.

Referenced by finalize(), initialize(), notifyReady(), notifyResponse(), and waitForRequest().

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

- ShmClientMgr.hpp
- ShmClientMgr.cpp

6.35 CShmemManager Class Reference

Service managing the shared memory between the main process and plug-in processes.

```
#include <ShmemManager.hpp>
```

Protected Types

- typedef boost::shared_ptr< **SModuleSync** > **TPtrSModuleSync**
*Type definition for a smart pointer to the **SModuleSync** (p. 155) struct.*

Protected Member Functions

- const std::string **getMemoryID** ()
Name resolution helper methods.
- const std::string **getMutexID** ()
- const **TPtrIpcMutex** & **getMutex** ()
Returns reference to the mutex.
- void **initializeMainProcess** (unsigned int nMemSize)
Compound method to be called in the main process (server side) to initialize the shared memory of the server.
- void **initializeChildProcess** ()
Compound method to be called in the subprocess (client side) to initialize the shared memory of the client.
- void **finalizeMainProcess** ()
Compound method to be called when the shared memory of the main process is no longer to be used.
- void **finalizeChildProcess** ()
Compound method to be called when the shared memory of the child process is no longer to be used.
- **SModuleRequest** * **allocateRequest** ()
Allocates a new request object in the shared memory.
- **SModuleResponse** * **allocateResponse** ()
Allocates a new response object in the shared memory.
- **SModuleRequest** * **retrieveRequest** ()
Retrieves a handle to the request data structure from the shared memory.
- **SModuleResponse** * **retrieveResponse** ()
Retrieves a handle to the response data structure from the shared memory.
- const **TPtrSModuleRequestDeleter** & **getRequestDeleter** ()

Returns the instance of the request deleter.

- `const TPtrSModuleResponseDeleter & getResponseDeleter ()`
Returns the instance of the response deleter.

Static Protected Member Functions

- `static void configure (const std::string &szPrefix)`
Configures the prefix of the shared memory objects.
- `static CShmemManager & Instance ()`
Returns singleton instance.

Private Member Functions

- `void createShmem (unsigned int nMemSize)`
Create the shared memory objects.
- `void deleteShmem ()`
Destroys the shared memory.
- `void openShmem ()`
Open the shared memory objects.
- `void closeShmem ()`
Closes the shared memory.
- `void initializeHelpers ()`
Initialize the shared memory helper functors.
- `const std::string getRequestID ()`
Constructs a request/response instance names.
- `const std::string getResponseID ()`
- `CShmemManager ()`
Default constructor.
- `virtual ~CShmemManager ()`
Virtual destructor.
- `CShmemManager (const CShmemManager &)`
No copying defined.
- `CShmemManager & operator= (const CShmemManager &)`
No assignment defined.

Private Attributes

- **TPtrIpcManagedSharedMemory m_pShMemory**
Shared memory object.
- **TPtrIpcMutex m_pShMutex**
Shared mutex for access to the whole shared memory.
- **TPtrIpcCharAllocator m_pCharAllocator**
Character allocator instance.
- **TPtrIpcSubstAllocator m_pSubstAllocator**
Variable substitution allocator instance.
- **TPtrSModuleRequestDeleter m_pRequestDeleter**
Module request smart pointer deleter.
- **TPtrSModuleResponseDeleter m_pResponseDeleter**
Module response smart pointer deleter.

Static Private Attributes

- static std::string **m_szPrefix**
Application specific prefix for shared memory objects.

Friends

- struct **ShmemLock**
- class **CShmClientMgr**
- class **CShmServerMgr**

Classes

- struct **SModuleSync**
Synchronisation conditions structure.

6.35.1 Detailed Description

Service managing the shared memory between the main process and plug-in processes.

The service holds all the memory allocators, pointer deleters and other helper objects.

It is not to be used itself, therefore it has empty public interface. Instead a specialised interface subclasses should be used by the final user code.

All subprocesses share the same memory, same mutex for it, but for each of them, the manager holds a separate set of signaling named condition.

All identifiers are composed of a constant application prefix, main process ID and an additional data structure suffix. The prefix is provided by the application when the shared memory manager is created. The constants are defined in the corresponding cpp file.

Important note: Each process (main + subprocesses) should use only one single instance of the shared memory manager. However, the main process has to **configure** and subsequently **create** the shared memory and, the mutexes and condition variables *_before_* subprocesses are forked, while these **open** them after they are launched. Do not re-use the server side (main process) instance in the child process after the fork invocation!

Important note: The class does not handle the exceptions possibly thrown by the Boost.Interprocess library. The user code has to handle these!!!

Each failure in Boost.Interprocess results in `boost::interprocess::interprocess_exception` throw.

Definition at line 82 of file `ShmemManager.hpp`.

6.35.2 Member Typedef Documentation

6.35.2.1 `typedef boost::shared_ptr<SModuleSync> CShmemManager::TPtrSModuleSync` [protected]

Type definition for a smart pointer to the `SModuleSync` (p. 155) struct.

Definition at line 235 of file `ShmemManager.hpp`.

6.35.3 Constructor & Destructor Documentation

6.35.3.1 `CShmemManager::CShmemManager ()` [private]

Default constructor.

Definition at line 62 of file `ShmemManager.cpp`.

6.35.3.2 `CShmemManager::~~CShmemManager ()` [private, virtual]

Virtual destructor.

Definition at line 67 of file `ShmemManager.cpp`.

6.35.3.3 `CShmemManager::CShmemManager (const CShmemManager &)` [private]

No copying defined.

6.35.4 Member Function Documentation

6.35.4.1 `void CShmemManager::configure (const std::string & szPrefix)` [inline, static, protected]

Configures the prefix of the shared memory objects.

Definition at line 349 of file `ShmemManager.hpp`.

References `m_szPrefix`.

Referenced by CShmServerMgr::initialize().

6.35.4.2 `const std::string CShmemManager::getMemoryID ()` [protected]

Name resolution helper methods.

These methods construct the canonical name of the corresponding shared objects.

Definition at line 154 of file ShmemManager.cpp.

References `g_szMemorySuffix()`, and `m_szPrefix`.

Referenced by `createShmem()`, `deleteShmem()`, and `openShmem()`.

6.35.4.3 `const std::string CShmemManager::getMutexID ()` [protected]

Definition at line 160 of file ShmemManager.cpp.

References `g_szMutexSuffix()`, and `m_szPrefix`.

Referenced by `createShmem()`, `deleteShmem()`, and `openShmem()`.

6.35.4.4 `const TPtrIpcMutex & CShmemManager::getMutex ()` [inline, protected]

Returns reference to the mutex.

Used by the friend class `ShmemLock` (p. 203).

Definition at line 355 of file ShmemManager.hpp.

References `m_pShMutex`.

6.35.4.5 `CShmemManager & CShmemManager::Instance ()` [static, protected]

Returns singleton instance.

Definition at line 72 of file ShmemManager.cpp.

Referenced by `CShmServerMgr::createRequest()`, `CShmClientMgr::createResponse()`, `CShmServerMgr::finalize()`, `CShmClientMgr::finalize()`, `CShmServerMgr::initialize()`, `CShmClientMgr::initialize()`, `CShmClientMgr::retrieveRequest()`, and `CShmServerMgr::retrieveResponse()`.

6.35.4.6 `void CShmemManager::initializeMainProcess (unsigned int nMemSize)` [protected]

Compound method to be called in the main process (server side) to initialize the shared memory of the server.

Before the main process forks the subprocesses it should invoke this method in order to create the shared memory segment and initialize the shared memory manager for the future child oprocesses.

The argument is the size of the requested shared memory to be allocated.

Definition at line 180 of file ShmemManager.cpp.

References `createShmem()`, and `initializeHelpers()`.

Referenced by `CShmServerMgr::initialize()`.

6.35.4.7 void CShmemManager::initializeChildProcess () [protected]

Compound method to be called in the subprocess (client side) to initialize the shared memory of the client.

When a sub-process is launched, it should call this method in order to reset some already in the main process initialized data members (e.g. pointers to the shared memory which was already created in the main process).

Definition at line 166 of file ShmemManager.cpp.

References initializeHelpers(), m_pCharAllocator, m_pRequestDeleter, m_pResponseDeleter, m_pShMemory, m_pShMutex, m_pSubstAllocator, and openShmem().

Referenced by CShmClientMgr::initialize().

6.35.4.8 void CShmemManager::finalizeMainProcess () [protected]

Compound method to be called when the shared memory of the main process is no longer to be used.

Should be called in the main process only. It closes the references to the shared memory objects and destroys the shared memory object from the operating system resources.

Definition at line 187 of file ShmemManager.cpp.

References deleteShmem().

Referenced by CShmServerMgr::finalize().

6.35.4.9 void CShmemManager::finalizeChildProcess () [protected]

Compound method to be called when the shared memory of the child process is no longer to be used.

Should be called only in the child processes. It closes the references to the shared objects.

Definition at line 193 of file ShmemManager.cpp.

References closeShmem().

Referenced by CShmClientMgr::finalize().

6.35.4.10 SModuleRequest * CShmemManager::allocateRequest () [protected]

Allocates a new request object in the shared memory.

Returns a shared pointer to the constructed object. The shared pointer is initialized with the appropriate object deleter, so when the object is being deleted, it is deleted using this deleter.

The user is responsible for the correct object destruction. The best is to let it be deleted automatically when the last shared pointer reference to it is destroyed.

Definition at line 199 of file ShmemManager.cpp.

References getRequestID(), m_pCharAllocator, m_pShMemory, and m_pSubstAllocator.

6.35.4.11 SModuleResponse * CShmemManager::allocateResponse () [protected]

Allocates a new response object in the shared memory.

See comments by `allocateRequest()` (p. 150) above.

Definition at line 205 of file `ShmemManager.cpp`.

References `getResponseID()`, `m_pCharAllocator`, `m_pShMemory`, and `m_pSubstAllocator`.

Referenced by `CShmClientMgr::createResponse()`.

6.35.4.12 `SModuleRequest * CShmemManager::retrieveRequest ()` [protected]

Retrieves a handle to the request data structure from the shared memory.

Definition at line 224 of file `ShmemManager.cpp`.

References `getRequestID()`, and `m_pShMemory`.

Referenced by `CShmClientMgr::retrieveRequest()`.

6.35.4.13 `SModuleResponse * CShmemManager::retrieveResponse ()` [protected]

Retrieves a handle to the response data structure from the shared memory.

Definition at line 232 of file `ShmemManager.cpp`.

References `getResponseID()`, and `m_pShMemory`.

6.35.4.14 `const TPtrSModuleRequestDeleter & CShmemManager::getRequestDeleter ()` [inline, protected]

Returns the instance of the request deleter.

Definition at line 361 of file `ShmemManager.hpp`.

References `m_pRequestDeleter`.

6.35.4.15 `const TPtrSModuleResponseDeleter & CShmemManager::getResponseDeleter ()` [inline, protected]

Returns the instance of the response deleter.

Definition at line 367 of file `ShmemManager.hpp`.

References `m_pResponseDeleter`.

6.35.4.16 `void CShmemManager::createShmem (unsigned int nMemSize)` [private]

Create the shared memory objects.

Should be called only in the main process.

Definition at line 80 of file `ShmemManager.cpp`.

References `getMemoryID()`, `getMutexID()`, `m_pShMemory`, and `m_pShMutex`.

Referenced by `initializeMainProcess()`.

6.35.4.17 void CShmemManager::deleteShmem () [private]

Destroys the shared memory.

Should be called only in the main process.

Definition at line 104 of file ShmemManager.cpp.

References `closeShmem()`, `getMemoryID()`, and `getMutexID()`.

Referenced by `finalizeMainProcess()`.

6.35.4.18 void CShmemManager::openShmem () [private]

Open the shared memory objects.

Should be called only in the subprocesses.

Definition at line 114 of file ShmemManager.cpp.

References `getMemoryID()`, `getMutexID()`, `m_pShMemory`, and `m_pShMutex`.

Referenced by `initializeChildProcess()`.

6.35.4.19 void CShmemManager::closeShmem () [private]

Closes the shared memory.

Should be called only in the subprocesses.

Definition at line 132 of file ShmemManager.cpp.

References `m_pCharAllocator`, `m_pRequestDeleter`, `m_pResponseDeleter`, `m_pShMemory`, `m_pShMutex`, and `m_pSubstAllocator`.

Referenced by `deleteShmem()`, and `finalizeChildProcess()`.

6.35.4.20 void CShmemManager::initializeHelpers () [private]

Initialize the shared memory helper functors.

Initializes the character/pair allocators and `scoped_ptr` corresponding deleters.

This method can be called only after either `createShmem()` (p. 151), or `openShmem()` (p. 152) was called.

Definition at line 144 of file ShmemManager.cpp.

References `m_pCharAllocator`, `m_pRequestDeleter`, `m_pResponseDeleter`, `m_pShMemory`, and `m_pSubstAllocator`.

Referenced by `initializeChildProcess()`, and `initializeMainProcess()`.

6.35.4.21 const std::string CShmemManager::getRequestID () [private]

Constructs a request/response instance names.

Definition at line 211 of file ShmemManager.cpp.

References `g_szRequestSuffix()`, and `m_szPrefix`.

Referenced by `allocateRequest()`, and `retrieveRequest()`.

6.35.4.22 `const std::string CShmemManager::getResponseID ()` [private]

Definition at line 217 of file ShmemManager.cpp.

References `g_szResponseSuffix()`, and `m_szPrefix`.

Referenced by `allocateResponse()`, and `retrieveResponse()`.

6.35.4.23 `CShmemManager& CShmemManager::operator= (const CShmemManager &)` [private]

No assignment defined.

6.35.5 Friends And Related Function Documentation**6.35.5.1** `friend struct ShmemLock` [friend]

Definition at line 86 of file ShmemManager.hpp.

6.35.5.2 `friend class CShmClientMgr` [friend]

Definition at line 87 of file ShmemManager.hpp.

6.35.5.3 `friend class CShmServerMgr` [friend]

Definition at line 88 of file ShmemManager.hpp.

6.35.6 Member Data Documentation**6.35.6.1** `TPtrIpcManagedSharedMemory CShmemManager::m_pShMemory` [private]

Shared memory object.

Definition at line 283 of file ShmemManager.hpp.

Referenced by `allocateRequest()`, `allocateResponse()`, `closeShmem()`, `createShmem()`, `initializeChildProcess()`, `initializeHelpers()`, `openShmem()`, `retrieveRequest()`, and `retrieveResponse()`.

6.35.6.2 `TPtrIpcMutex CShmemManager::m_pShMutex` [private]

Shared mutex for access to the whole shared memory.

Definition at line 286 of file ShmemManager.hpp.

Referenced by `closeShmem()`, `createShmem()`, `getMutex()`, `initializeChildProcess()`, and `openShmem()`.

6.35.6.3 `TPtrIpcCharAllocator CShmemManager::m_pCharAllocator` [private]

Character allocator instance.

Definition at line 289 of file ShmemManager.hpp.

Referenced by `allocateRequest()`, `allocateResponse()`, `closeShmem()`, `initializeChildProcess()`, and `initializeHelpers()`.

6.35.6.4 `TPtrIpcSubstAllocator` `CShmemManager::m_pSubstAllocator` [private]

Variable substitution allocator instance.

Definition at line 292 of file ShmemManager.hpp.

Referenced by `allocateRequest()`, `allocateResponse()`, `closeShmem()`, `initializeChildProcess()`, and `initializeHelpers()`.

6.35.6.5 `TPtrSModuleRequestDeleter` `CShmemManager::m_pRequestDeleter` [private]

Module request smart pointer deleter.

Definition at line 295 of file ShmemManager.hpp.

Referenced by `closeShmem()`, `getRequestDeleter()`, `initializeChildProcess()`, and `initializeHelpers()`.

6.35.6.6 `TPtrSModuleResponseDeleter` `CShmemManager::m_pResponseDeleter` [private]

Module response smart pointer deleter.

Definition at line 298 of file ShmemManager.hpp.

Referenced by `closeShmem()`, `getResponseDeleter()`, `initializeChildProcess()`, and `initializeHelpers()`.

6.35.6.7 `std::string` `CShmemManager::m_szPrefix` [static, private]

Application specific prefix for shared memory objects.

Definition at line 301 of file ShmemManager.hpp.

Referenced by `configure()`, `getMemoryID()`, `getMutexID()`, `CShmemManager::SModuleSync::getReadySemaphoreID()`, `CShmemManager::SModuleSync::getRequestConditionID()`, `getRequestID()`, `CShmemManager::SModuleSync::getResponseConditionID()`, and `getResponseID()`.

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

- **ShmemManager.hpp**
- **ShmemManager.cpp**

6.36 CShmemManager::SModuleSync Struct Reference

Synchronisation conditions structure.

```
#include <ShmemManager.hpp>
```

Public Member Functions

- **SModuleSync** (const std::string &szModuleID)
Constructor with the module ID.
- void **create** ()
Creates the shared named conditions.
- void **open** ()
Opens the shared named conditions.
- void **close** ()
Closes the handles to the shared conditions.
- void **destroy** ()
Destroys the shared named conditions.
- const std::string **getRequestConditionID** ()
Name resolution helper methods.
- const std::string **getResponseConditionID** ()
- const std::string **getReadySemaphoreID** ()

Public Attributes

- std::string **m_szModuleID**
Application specific prefix for shared named conditions.
- **TPtrIpcCondition m_pShReqCondition**
Shared condition for signaling that request is present in the memory.
- **TPtrIpcCondition m_pShRespCondition**
Shared condition for signaling that response is present in the memory.
- **TPtrIpcSemaphore m_pShReadySemaphore**
Shared condition for signaling readiness for an request-response exchange.

6.36.1 Detailed Description

Synchronisation conditions structure.

The structure for synchronising the a single module (client) with the main (server) process. It is instantiated on both Ipc sides. On server side, the conditions are created, while on the client side they are just opened.

The instance is copyable and copyconstructible.

Definition at line 195 of file ShmemManager.hpp.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 CShmemManager::SModuleSync::SModuleSync (const std::string & *szModuleID*)

Constructor with the module ID.

Definition at line 244 of file ShmemManager.cpp.

6.36.3 Member Function Documentation

6.36.3.1 void CShmemManager::SModuleSync::create ()

Creates the shared named conditions.

Definition at line 250 of file ShmemManager.cpp.

References `getReadySemaphoreID()`, `getRequestConditionID()`, `getResponseConditionID()`, `m_pShReadySemaphore`, `m_pShReqCondition`, and `m_pShRespCondition`.

6.36.3.2 void CShmemManager::SModuleSync::open ()

Opens the shared named conditions.

Definition at line 260 of file ShmemManager.cpp.

References `getReadySemaphoreID()`, `getRequestConditionID()`, `getResponseConditionID()`, `m_pShReadySemaphore`, `m_pShReqCondition`, and `m_pShRespCondition`.

6.36.3.3 void CShmemManager::SModuleSync::close ()

Closes the handles to the shared conditions.

Definition at line 300 of file ShmemManager.cpp.

References `m_pShReadySemaphore`, `m_pShReqCondition`, and `m_pShRespCondition`.

Referenced by `destroy()`.

6.36.3.4 void CShmemManager::SModuleSync::destroy ()

Destroys the shared named conditions.

Definition at line 288 of file ShmemManager.cpp.

References `close()`, `getReadySemaphoreID()`, `getRequestConditionID()`, and `getResponseConditionID()`.

6.36.3.5 `const std::string CShmemManager::SModuleSync::getRequestConditionID()`

Name resolution helper methods.

These methods construct the canonical name of the corresponding shared objects.

Definition at line 270 of file ShmemManager.cpp.

References `g_szRequestConditionSuffix()`, `m_szModuleID`, and `CShmemManager::m_szPrefix`.

Referenced by `create()`, `destroy()`, and `open()`.

6.36.3.6 `const std::string CShmemManager::SModuleSync::getResponseConditionID()`

Definition at line 276 of file ShmemManager.cpp.

References `g_szResponseConditionSuffix()`, `m_szModuleID`, and `CShmemManager::m_szPrefix`.

Referenced by `create()`, `destroy()`, and `open()`.

6.36.3.7 `const std::string CShmemManager::SModuleSync::getReadySemaphoreID()`

Definition at line 282 of file ShmemManager.cpp.

References `g_szReadySemaphoreSuffix()`, `m_szModuleID`, and `CShmemManager::m_szPrefix`.

Referenced by `create()`, `destroy()`, and `open()`.

6.36.4 Member Data Documentation

6.36.4.1 `std::string CShmemManager::SModuleSync::m_szModuleID`

Application specific prefix for shared named conditions.

Definition at line 201 of file ShmemManager.hpp.

Referenced by `getReadySemaphoreID()`, `getRequestConditionID()`, and `getResponseConditionID()`.

6.36.4.2 `TPtrIpcCondition CShmemManager::SModuleSync::m_pShReqCondition`

Shared condition for signaling that request is present in the memory.

Definition at line 204 of file ShmemManager.hpp.

Referenced by `close()`, `create()`, and `open()`.

6.36.4.3 `TPtrIpcCondition CShmemManager::SModuleSync::m_pShRespCondition`

Shared condition for signaling that response is present in the memory.

Definition at line 207 of file ShmemManager.hpp.

Referenced by `close()`, `create()`, and `open()`.

6.36.4.4 TPtrIpcSemaphore CShmemManager::SModuleSync::m_ - pShReadySemaphore

Shared condition for signaling readiness for an request-response exchange.

Definition at line 210 of file ShmemManager.hpp.

Referenced by close(), create(), and open().

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

- **ShmemManager.hpp**
- **ShmemManager.cpp**

6.37 CShmServerMgr Class Reference

Server side interprocess communication manager.

```
#include <ShmServerMgr.hpp>
```

Public Member Functions

- void **initialize** (unsigned int nMemSize)
Initialized the server side shared memory manager.
- void **finalize** ()
Destroys the shared memory objects.
- void **addModuleSync** (const TPtrIdentifier &identifier)
Adds a new conditions set for a given module.
- void **waitForReady** (const TPtrIdentifier &identifier)
Waits for the ready sync signal from the child process.
- void **waitForResponse** (const TPtrIdentifier &identifier, **ShmemLock** &lock)
Waits for the response sync signal from the child process.
- void **waitForResponse** (const TPtrIdentifier &identifier, const unsigned long &nMilliseconds, **ShmemLock** &lock)
Waits a limited time for the response sync signal from the child process.
- void **notifyRequest** (const TPtrIdentifier &identifier)
Notifies the subprocess about the request data presence in the shared memory.
- **TPtrSModuleRequest createRequest** ()
Creates a request structure in the shared memory.
- **TPtrSModuleResponse retrieveResponse** ()
Retrieves a response from the shared memory.

Static Public Member Functions

- static **CShmServerMgr & Instance** ()
Returns the singleton instance.

Private Types

- typedef **TIdentifierMap< CShmemManager::TPtrSModuleSync > TModuleSyncMap**
Type definition for map of module synchronisation conditions.

Private Member Functions

- void **freeSyncs** ()
Destroys the shared condition resources for the shared memory.
- **CShmServerMgr** ()
Default constructor.
- virtual **~CShmServerMgr** ()
Virtual destructor.
- **CShmServerMgr** (const **CShmServerMgr** &)
No copying defined.
- **CShmServerMgr** & **operator=** (const **CShmServerMgr** &)
No assignment defined.

Private Attributes

- **TModuleSyncMap** **m_mapModuleSync**
Association map for sync. conditions specified per each subprocess (identified by an identifier).

6.37.1 Detailed Description

Server side interprocess communication manager.

An interface to the server side **CShmemManager** (p.145) class. The singleton is only a bare interface to the shared memory manager singleton instance in the process.

Stores a map of module synchronisation conditions used for signaling between the main process (server) and a corresponding subprocess (client).

It is also used to configure the shared memory manager with the main proces specific prefix. It's form is "jazzyk_<pid>_".

Important note: Since the shared memory prefix has to be propagated to all the subprocesses which should subsequently open the shared memory according to this identifier, the singleton **must** be first time used to create the shared memory before forking the subprocesses.

Definition at line 59 of file ShmServerMgr.hpp.

6.37.2 Member Typedef Documentation

6.37.2.1 typedef TIdentifierMap<CShmemManager::TPtrSModuleSync> CShmServerMgr::TModuleSyncMap [private]

Type definition for map of module synchronisation conditions.

Basically, it is good only to save typing in the source code.

Definition at line 167 of file ShmServerMgr.hpp.

6.37.3 Constructor & Destructor Documentation

6.37.3.1 CShmServerMgr::CShmServerMgr () [private]

Default constructor.

Definition at line 52 of file ShmServerMgr.cpp.

6.37.3.2 CShmServerMgr::~CShmServerMgr () [private, virtual]

Virtual destructor.

Definition at line 58 of file ShmServerMgr.cpp.

6.37.3.3 CShmServerMgr::CShmServerMgr (const CShmServerMgr &) [private]

No copying defined.

6.37.4 Member Function Documentation

6.37.4.1 CShmServerMgr & CShmServerMgr::Instance () [static]

Returns the singleton instance.

Definition at line 63 of file ShmServerMgr.cpp.

Referenced by CModuleManager::communicate(), CModuleManager::extractResponse(), CModuleManager::finalize(), CModuleManager::initialize(), CModuleManager::load(), and CModuleManager::writeRequest().

6.37.4.2 void CShmServerMgr::initialize (unsigned int *nMemSize*)

Initialized the server side shared memory manager.

Configures the **CShmemManager** (p. 145) class and creates the shared memory objects. It takes an argument which is the size of the shared memory segment to allocate.

To be called before forking the subprocesses!

Definition at line 71 of file ShmServerMgr.cpp.

References CShmemManager::configure(), g_szSharedMemoryPrefix(), CShmemManager::initializeMainProcess(), and CShmemManager::Instance().

Referenced by CModuleManager::initialize().

6.37.4.3 void CShmServerMgr::finalize ()

Destroys the shared memory objects.

It is to be used when the shared memory is no longer to be used.

Definition at line 84 of file ShmServerMgr.cpp.

References CShmemManager::finalizeMainProcess(), freeSyncs(), and CShmemManager::Instance().

Referenced by `CModuleManager::finalize()`.

6.37.4.4 `void CShmServerMgr::addModuleSync (const TPtrIdentifier & identifier)`

Adds a new conditions set for a given module.

Definition at line 94 of file `ShmServerMgr.cpp`.

References `m_mapModuleSync`.

Referenced by `CModuleManager::load()`.

6.37.4.5 `void CShmServerMgr::waitForReady (const TPtrIdentifier & identifier)`

Waits for the ready sync signal from the child process.

Waits for the ready condition signal from the subprocess corresponding to the identifier in the method's first argument.

The second argument is the scoped lock object for the scope we are in and which is currently being locked.

For more detail, check the Boost.Interprocess documentation on `boost::interprocess::named_condition::wait(L& lock)`.

Definition at line 119 of file `ShmServerMgr.cpp`.

References `m_mapModuleSync`.

Referenced by `CModuleManager::communicate()`.

6.37.4.6 `void CShmServerMgr::waitForResponse (const TPtrIdentifier & identifier, ShmemLock & lock)`

Waits for the response sync signal from the child process.

For more info, check the `waitForReady(.)` method documentation above.

Definition at line 125 of file `ShmServerMgr.cpp`.

References `m_mapModuleSync`.

Referenced by `CModuleManager::communicate()`.

6.37.4.7 `void CShmServerMgr::waitForResponse (const TPtrIdentifier & identifier, const unsigned long & nMilliseconds, ShmemLock & lock)`

Waits a limited time for the response sync signal from the child process.

Definition at line 131 of file `ShmServerMgr.cpp`.

References `m_mapModuleSync`.

6.37.4.8 `void CShmServerMgr::notifyRequest (const TPtrIdentifier & identifier)`

Notifies the subprocess about the request data presence in the shared memory.

Notifies the corresponding subprocess that the request data structure is present in the memory and should be processed.

On more info, read the documentation of `boost::interprocess::named_condition::notify_one(.)`.

Definition at line 141 of file `ShmServerMgr.cpp`.

References `m_mapModuleSync`.

Referenced by `CModuleManager::communicate()`.

6.37.4.9 TPtrSModuleRequest CShmServerMgr::createRequest ()

Creates a request structure in the shared memory.

Returns a smart pointer to the allocated object in the shared memory. The user is responsible for the object destruction! I.e., let the smart pointer die and the shared memory object dies with it.

Remark: The server side is responsible for the shared memory object destruction.

Definition at line 147 of file `ShmServerMgr.cpp`.

References `CShmemManager::Instance()`.

6.37.4.10 TPtrSModuleResponse CShmServerMgr::retrieveResponse ()

Retrieves a response from the shared memory.

The variable substitution is erased and its content is replaced with the returned one.

Note: Regarding object's destruction, see the comment in `createRequest(...)` method.

Remark: The server side is responsible for the shared memory object destruction.

Definition at line 156 of file `ShmServerMgr.cpp`.

References `CShmemManager::Instance()`.

6.37.4.11 void CShmServerMgr::freeSyncs () [private]

Destroys the shared condition resources for the shared memory.

Definition at line 106 of file `ShmServerMgr.cpp`.

References `m_mapModuleSync`.

Referenced by `finalize()`.

6.37.4.12 CShmServerMgr& CShmServerMgr::operator= (const CShmServerMgr &) [private]

No assignment defined.

6.37.5 Member Data Documentation

6.37.5.1 TModuleSyncMap CShmServerMgr::m_mapModuleSync [private]

Association map for sync. conditions specified per each subprocess (identified by an identifier).

However, we have to be careful to destroy the shared named conditions properly. On server side `destroy()` has to be called. On client side, no special handling is needed.

Definition at line 175 of file ShmServerMgr.hpp.

Referenced by `addModuleSync()`, `freeSyncs()`, `notifyRequest()`, `waitForReady()`, and `waitForResponse()`.

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

- **ShmServerMgr.hpp**
- **ShmServerMgr.cpp**

6.38 CSingleModule Class Reference

Wrapper for a KR module.

```
#include <SingleModule.hpp>
```

Public Member Functions

- void **run** (const std::string &szType, const std::string &pModuleID, const **TVecStrings** &vecSearchPaths)
Implements the subprocess main routine.

Static Public Member Functions

- static **CSingleModule** & **Instance** ()
Returns the singleton instance.

Private Types

- typedef boost::shared_ptr< **jazyk::KRModuleInterface** > **TPtrModule**
Type definition for a pointer to the abstract KR module interface instance.

Private Member Functions

- **SModuleRequest** * **extractRequest** ()
Retrieves a request from the shared memory.
- void **writeResponse** (const **EModuleErrCode** eError, const char *parrszSubstName[]=0, const char *parrszSubstValue[]=0, const unsigned int *pnSubstSize=0, const bool *pbQueryResult=0)
Writes the response into the shared memory.
- bool **processRequest** ()
Handles the request on the subprocess side.
- void **loadDll** ()
Loads the DLL corresponding to the KR module.
- void **unloadDll** ()
Unloads the DLL corresponding to the KR module.
- void **extract_names_values** (const **TIPCSubst** &substitution, char **&arrszName, char **&arrszValue, unsigned int &nSize)
Converts a variable substitution from the shared memory format into the raw format.
- void **convert_operations** (char **&arrszID1, char **&arrszID2, char **&arrszName, char **&arrszValue, const unsigned int &nSize1, const unsigned int &nSize2)

Fills the map structure with module operation API entries according to their type.

- void **free_names_values** (char **arrszName, char **arrszValue, const unsigned int &nSize)
De-allocates the converted raw variable substitution.
- void **handleModuleError** (const std::string &szReason, bool bForce=false)
Reports the libtool error.
- **CSingleModule** ()
Default constructor.
- virtual **~CSingleModule** ()
Virtual destructor.
- **CSingleModule** (const **CSingleModule** &)
No copying defined.
- **CSingleModule** & **operator=** (const **CSingleModule** &)
No assignment defined.

Private Attributes

- std::string **m_szModuleType**
KR module type.
- std::string **m_szModuleID**
KR module identifier.
- TVecStrings **m_vecPaths**
Shared library paths.
- lt_dlhandle **m_hLibrary**
Handle to the DLL.
- boost::shared_ptr< **jazyk::KRModuleInterface** > **m_pModule**
Pointer to the abstract KR module interface instance.

6.38.1 Detailed Description

Wrapper for a KR module.

Singleton responsible to the run the KR module subprocess. Communicates with the server (ModuleManager) via shared memory and on the other hand controls the shared library containing the corresponding KR module.

Definition at line 52 of file SingleModule.hpp.

6.38.2 Member Typedef Documentation

6.38.2.1 `typedef boost::shared_ptr<jazzyk::KRModuleInterface> CSingleModule::TPtrModule [private]`

Type definition for a pointer to the abstract KR module interface instance.

Definition at line 198 of file SingleModule.hpp.

6.38.3 Constructor & Destructor Documentation

6.38.3.1 `CSingleModule::CSingleModule () [private]`

Default constructor.

Definition at line 61 of file SingleModule.cpp.

6.38.3.2 `CSingleModule::~~CSingleModule () [private, virtual]`

Virtual destructor.

Definition at line 70 of file SingleModule.cpp.

6.38.3.3 `CSingleModule::CSingleModule (const CSingleModule &) [private]`

No copying defined.

6.38.4 Member Function Documentation

6.38.4.1 `CSingleModule & CSingleModule::Instance () [static]`

Returns the singleton instance.

Definition at line 75 of file SingleModule.cpp.

Referenced by CModuleManager::load().

6.38.4.2 `void CSingleModule::run (const std::string & szType, const std::string & pModuleID, const TVecStrings & vecSearchPaths)`

Implements the subprocess main routine.

The subprocess enters the communication cycle: 1.- signal readiness to the main process 2.- wait for a request in the shared memory 3.- handle the request 4.- put the response to the shared memory 5.- signal to the main process that response is present in the shared memory 6.- goto 1.

The communication cycle is synchronised using a shared named mutex and semaphore with the main process.

The arguments designate the plug-in type and the identifier of the module.

IMPORTANT: The first message communicated to the subprocess has to be INITIALIZE, which causes the subprocess to load the KR module shared library. Only then the other messages can be processed correctly.

Definition at line 82 of file SingleModule.cpp.

References CShmClientMgr::finalize(), CShmClientMgr::initialize(), CShmClientMgr::Instance(), LOG, m_szModuleID, m_szModuleType, m_vecPaths, CShmClientMgr::notifyReady(), CShmClientMgr::notifyResponse(), processRequest(), and CShmClientMgr::waitForRequest().

Referenced by CModuleManager::load().

6.38.4.3 SModuleRequest * CSingleModule::extractRequest () [private]

Retrieves a request from the shared memory.

Returns a plain pointer to the request. No attempt to destroy the object should be made.

Definition at line 141 of file SingleModule.cpp.

References CShmClientMgr::Instance(), and CShmClientMgr::retrieveRequest().

Referenced by processRequest().

6.38.4.4 void CSingleModule::writeResponse (const EModuleErrCode *eError*, const char * *parrszSubstName*[] = 0, const char * *parrszSubstValue*[] = 0, const unsigned int * *pnSubstSize* = 0, const bool * *pbQueryResult* = 0) [private]

Writes the response into the shared memory.

Fills the non-null arguments into the response which is then written to the shared memory.

Does not return any handle. The lifetime of the object is controlled by the server side.

Definition at line 147 of file SingleModule.cpp.

References SModuleResponse::addVariableSubst(), CShmClientMgr::createResponse(), handleModuleError(), CShmClientMgr::Instance(), LOG, SModuleResponse::m_bQueryResult, and SModuleResponse::m_eError.

Referenced by handleModuleError(), and processRequest().

6.38.4.5 bool CSingleModule::processRequest () [private]

Handles the request on the subprocess side.

1.- Reads the shared memory, 2.- Retrieves the request to the subprocess, 3.- Forwards the request to the corresponding KR module, and finally 4.- Writes response to the shared memory

Returns true if the child process should exit for whatever reason, otherwise returns false.

It is a counterpart of communicate(...) on the server side.

Definition at line 186 of file SingleModule.cpp.

References convert_operations(), CYCLE, extract_names_values(), extractRequest(), FAIL, FINALIZE, free_names_values(), GETINTERFACE, handleModuleError(), INITIALIZE, jazzyk::INVALID_OPERATION, loadDll(), LOG, SModuleRequest::m_eType, SModuleRequest::m_mapFreeVariables, SModuleRequest::m_mapSubstitution, m_pModule, SModuleRequest::m_szCodeBlock, SModuleRequest::m_szOperation, OK, jazzyk::OK, QUERY, UPDATE, and writeResponse().

Referenced by run().

6.38.4.6 void CSingleModule::loadDll () [private]

Loads the DLL corresponding to the KR module.

Initializes the libtool DLL loading mechanism and loads the shared library.

Definition at line 425 of file SingleModule.cpp.

References `cszLibNamePrefix()`, `cszLibNameSuffix()`, `handleModuleError()`, `JAZZYK_API_VERSION`, `JZMODULE_API_REGISTER_ROUTINE`, `JZMODULE_API_VERSION_ROUTINE`, `LOG`, `m_hLibrary`, `m_pModule`, `m_szModuleType`, and `m_vecPaths`.

Referenced by `processRequest()`.

6.38.4.7 void CSingleModule::unloadDll () [private]

Unloads the DLL corresponding to the KR module.

Unloads the shared library and de-initializes the libtool loading mechanism.

Definition at line 495 of file SingleModule.cpp.

References `handleModuleError()`, and `m_hLibrary`.

6.38.4.8 void CSingleModule::extract_names_values (const TIPCSubst & substitution, char **& arrszName, char **& arrszValue, unsigned int & nSize) [private]

Converts a variable substitution from the shared memory format into the raw format.

Definition at line 543 of file SingleModule.cpp.

Referenced by `processRequest()`.

6.38.4.9 void CSingleModule::convert_operations (char ** arrszID1, char ** arrszID2, char **& arrszName, char **& arrszValue, const unsigned int & nSize1, const unsigned int & nSize2) [private]

Fills the map structure with module operation API entries according to their type.

- first argument is a list of query operations
- second argument is a list of update operations
- third/fourth arguments are the output map
- fifth/sixth arguments are sizes of first/second arguments respectively

Definition at line 564 of file SingleModule.cpp.

References `g_JZAPI_QUERY`, and `g_JZAPI_UPDATE`.

Referenced by `processRequest()`.

6.38.4.10 void CSingleModule::free_names_values (char ** arrszName, char ** arrszValue, const unsigned int & nSize) [private]

De-allocates the converted raw variable substitution.

Definition at line 595 of file SingleModule.cpp.

Referenced by processRequest().

6.38.4.11 void CSingleModule::handleModuleError (const std::string & *szReason*, bool *bForce* = false) [private]

Reports the libtool error.

Writes a response about failing request processing into the shared memory, notifies the main process about this issue.

However, it does not exit the subprocess, but rather waits for the main process to finish it by the FINALIZE message.

If the optional argument bForce is set to true, nothing will be written to the shared memory and the process will simply throw. This should be used in situations when the shared memory is broken and there's no point in attempting a communication again.

IMPORTANT: Can be called only if the communication channel is opened and working. Otherwise, the subprocess should simply throw.

Definition at line 513 of file SingleModule.cpp.

References FAIL, CShmClientMgr::Instance(), CError::JZMODULE_ERROR, LOG, m_ szModuleID, CShmClientMgr::notifyResponse(), and writeResponse().

Referenced by loadDll(), processRequest(), unloadDll(), and writeResponse().

6.38.4.12 CSingleModule& CSingleModule::operator= (const CSingleModule &) [private]

No assignment defined.

6.38.5 Member Data Documentation

6.38.5.1 std::string CSingleModule::m_ szModuleType [private]

KR module type.

Definition at line 186 of file SingleModule.hpp.

Referenced by loadDll(), and run().

6.38.5.2 std::string CSingleModule::m_ szModuleID [private]

KR module identifier.

Definition at line 189 of file SingleModule.hpp.

Referenced by handleModuleError(), and run().

6.38.5.3 TVecStrings CSingleModule::m_ vecPaths [private]

Shared library paths.

Definition at line 192 of file SingleModule.hpp.

Referenced by `loadDll()`, and `run()`.

6.38.5.4 `It_dlhandle CSingleModule::m_hLibrary` [private]

Handle to the DLL.

Definition at line 195 of file `SingleModule.hpp`.

Referenced by `loadDll()`, and `unloadDll()`.

6.38.5.5 `boost::shared_ptr<jazzyk::KRModuleInterface> CSingleModule::m_pModule` [private]

Pointer to the abstract KR module interface instance.

Definition at line 201 of file `SingleModule.hpp`.

Referenced by `loadDll()`, and `processRequest()`.

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

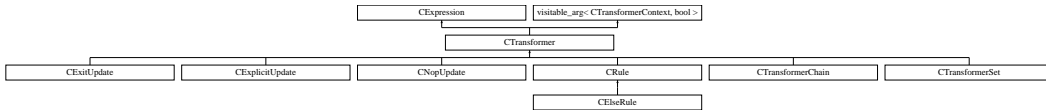
- `SingleModule.hpp`
- `SingleModule.cpp`

6.39 CTransformer Class Reference

Abstract class for all the various types of mental state transformers.

```
#include <Transformer.hpp>
```

Inheritance diagram for CTransformer::



Public Member Functions

- **CTransformer** ()
Default constructor.
- **virtual ~CTransformer** ()=0
Virtual destructor.

Private Member Functions

- **CTransformer** (const **CTransformer** &)
No copying defined.
- **CTransformer & operator=** (const **CTransformer** &)
No assignment defined.

6.39.1 Detailed Description

Abstract class for all the various types of mental state transformers.

A general mental state transformer is considered to be a compound mental state transformer. Only its special subclass stands for a single transformer.

CTransformer (p. 172) hierarchy is **visitable** (p. 214) with an argument **CTransformerContext** (p. 177) instance and void return value.

Definition at line 51 of file Transformer.hpp.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 CTransformer::CTransformer () [inline]

Default constructor.

Definition at line 93 of file Transformer.hpp.

6.39.2.2 CTransformer::~~CTransformer () [inline, pure virtual]

Virtual destructor.

Definition at line 98 of file Transformer.hpp.

6.39.2.3 CTransformer::CTransformer (const CTransformer &) [private]

No copying defined.

6.39.3 Member Function Documentation**6.39.3.1** CTransformer& CTransformer::operator= (const CTransformer &)
[private]

No assignment defined.

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

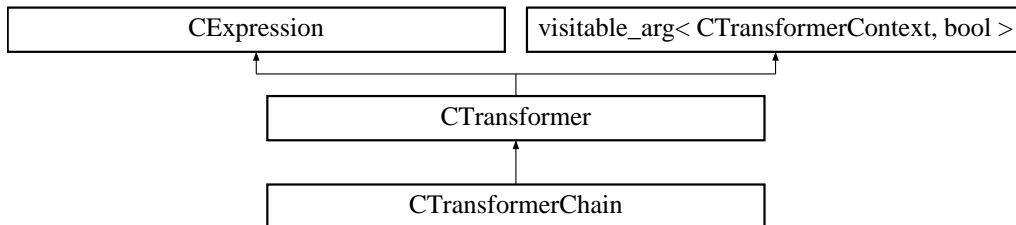
- **Transformer.hpp**

6.40 CTransformerChain Class Reference

Chain of mental state transformers.

```
#include <TransformerChain.hpp>
```

Inheritance diagram for CTransformerChain::



Public Member Functions

- **CTransformerChain** ()
Default constructor.
- **CTransformerChain** (TPtrTransformer pTransformer1, TPtrTransformer pTransformer2)
Constructor from two transformers.
- virtual **~CTransformerChain** ()
Virtual destructor.
- virtual void **push_front** (TPtrTransformer pTransformer)
Adds a new member of the chain to the front.
- virtual void **push_back** (TPtrTransformer pTransformer)
Adds a new member of the chain to the back.
- **TPtrTransformers getTransformers** () const
Returns pointer to the collection of transformers.

Private Member Functions

- **CTransformerChain** (const CTransformerChain &)
No copying defined.
- **CTransformerChain & operator=** (const CTransformerChain &)
No assignment defined.

Private Attributes

- **TPtrTransformers m_pTransformers**

Chain of contained mental state transformers.

6.40.1 Detailed Description

Chain of mental state transformers.

Chain of mental state transformers is executed one after another until the end.

Definition at line 44 of file TransformerChain.hpp.

6.40.2 Constructor & Destructor Documentation

6.40.2.1 CTransformerChain::CTransformerChain ()

Default constructor.

Initializes to an empty chain transformer

Definition at line 38 of file TransformerChain.cpp.

6.40.2.2 CTransformerChain::CTransformerChain (TPtrTransformer pTransformer1, TPtrTransformer pTransformer2)

Constructor from two transformers.

Definition at line 44 of file TransformerChain.cpp.

References m_pTransformers.

6.40.2.3 CTransformerChain::~~CTransformerChain () [virtual]

Virtual destructor.

Definition at line 53 of file TransformerChain.cpp.

6.40.2.4 CTransformerChain::CTransformerChain (const CTransformerChain &) [private]

No copying defined.

6.40.3 Member Function Documentation

6.40.3.1 void CTransformerChain::push_front (TPtrTransformer pTransformer) [virtual]

Adds a new member of the chain to the front.

Definition at line 58 of file TransformerChain.cpp.

References m_pTransformers.

6.40.3.2 `void CTransformerChain::push_back (TPtrTransformer pTransformer)` [virtual]

Adds a new member of the chain to the back.

Definition at line 64 of file TransformerChain.cpp.

References `m_pTransformers`.

6.40.3.3 `TPtrTransformers CTransformerChain::getTransformers () const` [inline]

Returns pointer to the collection of transformers.

Definition at line 108 of file TransformerChain.hpp.

References `m_pTransformers`.

Referenced by `CTraverser::Visit()`.

6.40.3.4 `CTransformerChain& CTransformerChain::operator= (const CTransformerChain &)` [private]

No assignment defined.

6.40.4 Member Data Documentation

6.40.4.1 `TPtrTransformers CTransformerChain::m_pTransformers` [private]

Chain of contained mental state transformers.

Definition at line 89 of file TransformerChain.hpp.

Referenced by `CTransformerChain()`, `getTransformers()`, `push_back()`, and `push_front()`.

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

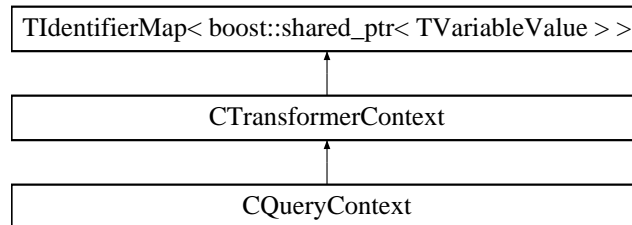
- `TransformerChain.hpp`
- `TransformerChain.cpp`

6.41 CTransformerContext Class Reference

Interpreter context used while traversing the transformer tree structure.

```
#include <Context.hpp>
```

Inheritance diagram for CTransformerContext::



Public Member Functions

- **CTransformerContext** (TPtrProgram pProgram)
Basic constructor.
- virtual **~CTransformerContext** ()
Context is copy-constructible.
- bool **isModuleDeclared** (const TPtrIdentifier &id) const
Verifies whether the module ID is declared in the corresponding transformer context.
- **TPtrModuleDeclaration** **getModuleDeclaration** (const TPtrIdentifier &id) const
Returns the associated module link.
- virtual bool **unify** (TVariableSubstitution &subst, TPtrIdentifier &pVariable)
Adds new variable substitution to the context.
- virtual bool **unify** (TVariableSubstitution &subst)
Simply adds a new variable substitution to the context.

Private Member Functions

- **CTransformerContext** & **operator=** (const CTransformerContext &)
No assignment defined.

Private Attributes

- **TPtrProgram** **m_pProgram**
Pointer to the program so that the root is always accessible.

Friends

- class `CQueryContext`

6.41.1 Detailed Description

Interpreter context used while traversing the transformer tree structure.

While the interpreter traverses the mental state transformer program sub-structure, it holds the context data in `CTransformerContext` (p.177). `CTransformerContext` (p.177) is also used in compiler for validating the program.

The main task of the context is to hold the currently valid free variable substitution. Variable substitutions used in the children nodes are not shared by the higher level nodes, therefore in each subsequent recursive step, the context is being copied and on the return it is destroyed.

Definition at line 53 of file `Context.hpp`.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 `CTransformerContext::CTransformerContext (TPtrProgram pProgram)`

Basic constructor.

The context is always initialized with the program parse trees to be traversed.

Definition at line 39 of file `Context.cpp`.

6.41.2.2 `CTransformerContext::~CTransformerContext ()` [virtual]

Context is copy-constructible.

Context has to be passed to the **visitor** (p.218) pattern machinery with argument. During the recursive traversal new contexts in each node are created.

No declaration/implementation provided - let the compiler generate an implicit copy constructor.
Virtual destructor

Definition at line 45 of file `Context.cpp`.

6.41.3 Member Function Documentation

6.41.3.1 `bool CTransformerContext::isModuleDeclared (const TPtrIdentifier & id) const`

Verifies whether the module ID is declared in the corresponding transformer context.

Definition at line 50 of file `Context.cpp`.

References `m_pProgram`.

Referenced by `CCompiler::PreAction()`.

6.41.3.2 TPtrModuleDeclaration CTransformerContext::getModuleDeclaration (const TPtrIdentifier & *id*) const

Returns the associated module link.

Definition at line 56 of file Context.cpp.

References m_pProgram.

Referenced by CCompiler::PreAction().

6.41.3.3 bool CTransformerContext::unify (TVariableSubstitution & *subst*, TPtrIdentifier & *pVariable*) [virtual]

Adds new variable substitution to the context.

Additionally, performs a consistency check on the new substitution. In the case the new variable substitution is about to rewrite a currently held variable substitution with a new value for a variable returns false and variable contains a pointer to the variable which first caused the inconsistency.

Otherwise the return value is true and variable is not touched.

Definition at line 62 of file Context.cpp.

Referenced by unify(), and CTraverser::Visit().

6.41.3.4 bool CTransformerContext::unify (TVariableSubstitution & *subst*) [virtual]

Simply adds a new variable substitution to the context.

Performs no consistency check.

Definition at line 88 of file Context.cpp.

References unify().

6.41.3.5 CTransformerContext& CTransformerContext::operator= (const CTransformerContext &) [private]

No assignment defined.

6.41.4 Friends And Related Function Documentation

6.41.4.1 friend class CQueryContext [friend]

Definition at line 56 of file Context.hpp.

6.41.5 Member Data Documentation

6.41.5.1 TPtrProgram CTransformerContext::m_pProgram [private]

Pointer to the program so that the root is always accessible.

Definition at line 119 of file Context.hpp.

Referenced by `getModuleDeclaration()`, and `isModuleDeclared()`.

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

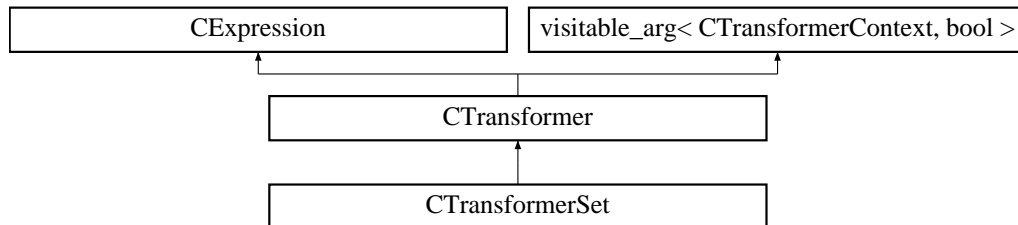
- **Context.hpp**
- **Context.cpp**

6.42 CTransformerSet Class Reference

Set of mental state transformers.

```
#include <TransformerSet.hpp>
```

Inheritance diagram for CTransformerSet::



Public Member Functions

- **CTransformerSet** ()
Default constructor.
- **CTransformerSet** (TPtrTransformer pTransformer1, TPtrTransformer pTransformer2)
Constructor from two transformers.
- virtual **~CTransformerSet** ()
Virtual destructor.
- virtual void **push_front** (TPtrTransformer pTransformer)
Adds new transformer to the set.
- virtual void **push_back** (TPtrTransformer pTransformer)
Adds new transformer to the set.
- **TPtrTransformers getTransformers** () const
Returns the pointer to the transformers.

Private Member Functions

- **CTransformerSet** (const CTransformerSet &)
No copying defined.
- **CTransformerSet & operator=** (const CTransformerSet &)
No assignment defined.

Private Attributes

- **TPtrTransformers m_pTransformers**
Set of mental state transformers.

6.42.1 Detailed Description

Set of mental state transformers.

Set of mental state transformers is executed by picking one transformer which is executed.

Definition at line 44 of file TransformerSet.hpp.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 CTransformerSet::CTransformerSet ()

Default constructor.

Initializes to an empty set transformer.

Definition at line 38 of file TransformerSet.cpp.

6.42.2.2 CTransformerSet::CTransformerSet (TPtrTransformer *pTransformer1*, TPtrTransformer *pTransformer2*)

Constructor from two transformers.

Definition at line 44 of file TransformerSet.cpp.

References m_pTransformers.

6.42.2.3 CTransformerSet::~~CTransformerSet () [virtual]

Virtual destructor.

Definition at line 52 of file TransformerSet.cpp.

6.42.2.4 CTransformerSet::CTransformerSet (const CTransformerSet &) [private]

No copying defined.

6.42.3 Member Function Documentation

6.42.3.1 void CTransformerSet::push_front (TPtrTransformer *pTransformer*) [virtual]

Adds new transformer to the set.

Adds the new member to the front of the list.

Also set is implemented as a list, and we want to keep track of order as well.

Definition at line 57 of file TransformerSet.cpp.

References m_pTransformers.

6.42.3.2 void CTransformerSet::push_back (TPtrTransformer *pTransformer*) [virtual]

Adds new transformer to the set.

Adds the new member to the back of the list

Also set is implemented as a list, and we want to keep track of order as well.

Definition at line 63 of file TransformerSet.cpp.

References m_pTransformers.

6.42.3.3 TPtrTransformers CTransformerSet::getTransformers () const [inline]

Returns the pointer to the transformers.

Definition at line 119 of file TransformerSet.hpp.

References m_pTransformers.

Referenced by CTraverser::Visit(), and CInterpreter::Visit().

6.42.3.4 CTransformerSet& CTransformerSet::operator= (const CTransformerSet &) [private]

No assignment defined.

6.42.4 Member Data Documentation

6.42.4.1 TPtrTransformers CTransformerSet::m_pTransformers [private]

Set of mental state transformers.

Definition at line 100 of file TransformerSet.hpp.

Referenced by CTransformerSet(), getTransformers(), push_back(), and push_front().

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

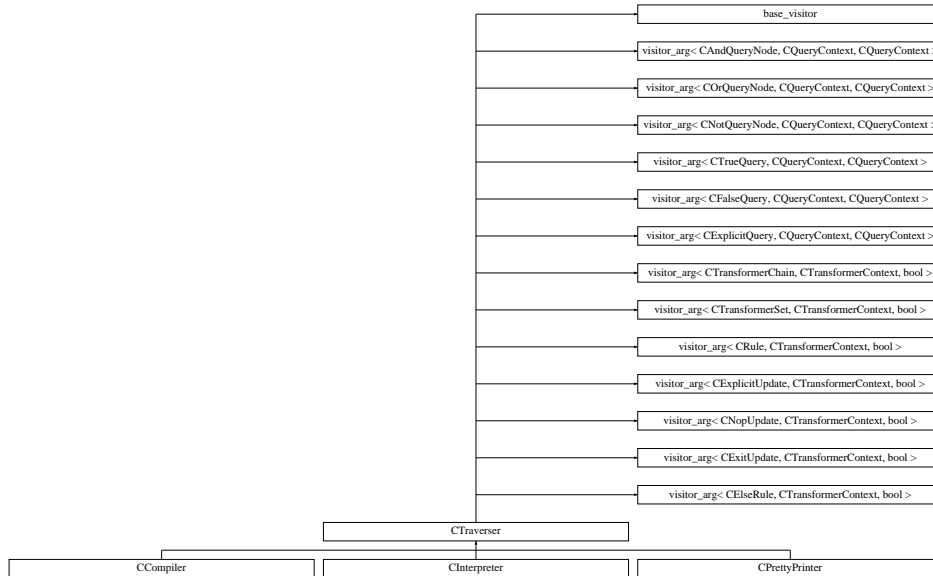
- TransformerSet.hpp
- TransformerSet.cpp

6.43 CTraverser Class Reference

Program transformer tree traversal abstract class.

```
#include <Traverser.hpp>
```

Inheritance diagram for CTraverser::



Public Member Functions

- **CTraverser ()**
Default constructor.
- **virtual ~CTraverser ()=0**
Virtual destructor.
- **virtual CQueryContext Visit (CAndQueryNode &, CQueryContext)**
Visitor methods for the CExpression (p. 62) hierarchy.
- **virtual CQueryContext Visit (COrQueryNode &, CQueryContext)**
Abstract callback from the guest.
- **virtual CQueryContext Visit (CNotQueryNode &, CQueryContext)**
Abstract callback from the guest.
- **virtual CQueryContext Visit (CTrueQuery &, CQueryContext)**
Abstract callback from the guest.
- **virtual CQueryContext Visit (CFalseQuery &, CQueryContext)**
Abstract callback from the guest.
- **virtual CQueryContext Visit (CExplicitQuery &, CQueryContext)**

Abstract callback from the guest.

- virtual bool **Visit** (CTransformerChain &, CTransformerContext)
Abstract callback from the guest.
- virtual bool **Visit** (CTransformerSet &, CTransformerContext)
Abstract callback from the guest.
- virtual bool **Visit** (CRule &, CTransformerContext)
Abstract callback from the guest.
- virtual bool **Visit** (CExplicitUpdate &, CTransformerContext)
Abstract callback from the guest.
- virtual bool **Visit** (CNoUpdate &, CTransformerContext)
Abstract callback from the guest.
- virtual bool **Visit** (CExitUpdate &, CTransformerContext)
Abstract callback from the guest.
- virtual bool **Visit** (CElseRule &, CTransformerContext)
Abstract callback from the guest.

Protected Member Functions

- virtual CQueryContext & PreAction (CAndQueryNode &, CQueryContext &)
Action invoked before the node is being visited.
- virtual CQueryContext & PreAction (COrQueryNode &, CQueryContext &)
- virtual CQueryContext & PreAction (CNotQueryNode &, CQueryContext &)
- virtual CQueryContext & PreAction (CTrueQuery &, CQueryContext &)
- virtual CQueryContext & PreAction (CFalseQuery &, CQueryContext &)
- virtual CQueryContext & PreAction (CExplicitQuery &, CQueryContext &)
- virtual void PreAction (CTransformerChain &, CTransformerContext &)
- virtual void PreAction (CTransformerSet &, CTransformerContext &)
- virtual void PreAction (CRule &, CTransformerContext &)
- virtual void PreAction (CExplicitUpdate &, CTransformerContext &)
- virtual void PreAction (CNoUpdate &, CTransformerContext &)
- virtual void PreAction (CExitUpdate &, CTransformerContext &)
- virtual void PreAction (CElseRule &, CTransformerContext &)
- virtual CQueryContext & PostAction (CAndQueryNode &, CQueryContext &)
Action invoked after the node is being visited.
- virtual CQueryContext & PostAction (COrQueryNode &, CQueryContext &)
- virtual CQueryContext & PostAction (CNotQueryNode &, CQueryContext &)
- virtual CQueryContext & PostAction (CTrueQuery &, CQueryContext &)
- virtual CQueryContext & PostAction (CFalseQuery &, CQueryContext &)
- virtual CQueryContext & PostAction (CExplicitQuery &, CQueryContext &)
- virtual void PostAction (CTransformerChain &, CTransformerContext &)

- virtual void **PostAction** (**CTransformerSet** &, **CTransformerContext** &)
- virtual void **PostAction** (**CRule** &, **CTransformerContext** &)
- virtual void **PostAction** (**CExplicitUpdate** &, **CTransformerContext** &)
- virtual void **PostAction** (**CNopUpdate** &, **CTransformerContext** &)
- virtual void **PostAction** (**CExitUpdate** &, **CTransformerContext** &)
- virtual void **PostAction** (**CElseRule** &, **CTransformerContext** &)
- virtual void **RuleMidAction** (**CRule** &, **CQueryContext** &, **CTransformerContext** &)

Action invoked after the rule query is traversed but before the transformer is visited.

- virtual void **RuleMidAction** (**CElseRule** &, **CQueryContext** &, **CTransformerContext** &)
- virtual **CQueryContext** & **DefaultPreAction** (**CExpression** &, **CQueryContext** &)

Default Pre-/Post- and MidRule- action implementations.

- virtual void **DefaultPreAction** (**CExpression** &, **CTransformerContext** &)
- virtual **CQueryContext** & **DefaultPostAction** (**CExpression** &, **CQueryContext** &)
- virtual void **DefaultPostAction** (**CExpression** &, **CTransformerContext** &)
- virtual void **DefaultRuleMidAction** (**CExpression** &, **CQueryContext** &, **CTransformerContext** &)
- virtual void **DefaultPreAction** ()
- virtual void **DefaultPostAction** ()
- virtual void **DefaultRuleMidAction** ()
- virtual void **traverse** (**TPtrProgram** pProgram)

Start the tree structure traversal.

Private Member Functions

- **CTraverser** (const **CTraverser** &)
- No copying defined.*
- **CTraverser** & **operator=** (const **CTraverser** &)
- No assignment defined.*

6.43.1 Detailed Description

Program transformer tree traversal abstract class.

CTraverser (p. 184) is a **visitor** (p. 218) of the whole **CExpression** (p. 62) hierarchy and implements the transformer tree depth-first recursive traversal.

It serves as a base class of classes which need to traverse the program tree. Program mental transformer tree is searched with the context object passed through the calls.

Subclasses are to implement the action methods for particular nodes of the program parse tree: they do not have to implement the traversal algorithm itself. Each node visit results in a calls to **PreAction**() (p. 190) and **PostAction**() (p. 192) called before and after the node visit respectively.

The traversal of **CTransformer** (p. 172) subhierarchy is guided by the **CTransformerContext** (p. 177) context objects, while **CQuery** subhierarchy is traversed using **CQueryContext** (p. 130) context objects.

The context handling is as follows: In the mental state transformer subtree, the contexts are passed in a top-down manner so data from higher level nodes are not shared by the children nodes, but rather plainly copied. Contrary to that, in the query node subtree, the contexts are "built" bottom up therefore the data are shared between the parent and children nodes.

Visits of transformer subhierarchy result in a bool value which indicates whether the leaf mst within the considered transformer was executed or not.

NOTE: The substitution resulting from a visit to a negated query (**CNotQueryNode** (p. 114)) is simply thrown away as the substitution is useless later. The same holds for the resulting substitution resulting from a visit to **COrQueryNode** (p. 116).

Definition at line 77 of file Traverser.hpp.

6.43.2 Constructor & Destructor Documentation

6.43.2.1 CTraverser::CTraverser ()

Default constructor.

Definition at line 41 of file Traverser.cpp.

6.43.2.2 CTraverser::~~CTraverser () [pure virtual]

Virtual destructor.

This is a pure abstract class, however still implementing default (empty) behaviour.

Definition at line 46 of file Traverser.cpp.

6.43.2.3 CTraverser::CTraverser (const CTraverser &) [private]

No copying defined.

6.43.3 Member Function Documentation

6.43.3.1 CQueryContext CTraverser::Visit (CAndQueryNode & *guest*, CQueryContext *ctx*) [virtual]

Visitor methods for the **CExpression** (p. 62) hierarchy.

Visit() (p. 187) always invokes virtual methods **PreAction()** (p. 190) and **PostAction()** (p. 192) with the corresponding context passed to the subclasses.

Implements **visitor_arg< CAndQueryNode, CQueryContext, CQueryContext >** (p. 221).

Definition at line 59 of file Traverser.cpp.

References **CAndQueryNode::getLeftNode()**, **CQueryContext::getResult()**, **CAndQueryNode::getRightNode()**, **CQueryContext::isTrueW()**, **PostAction()**, **PreAction()**, **CQueryContext::resetResult()**, and **CQueryContext::setResult()**.

6.43.3.2 CQueryContext CTraverser::Visit (COrQueryNode &, CQueryContext) [virtual]

Abstract callback from the guest.

Implements `visitor_arg< COrQueryNode, CQueryContext, CQueryContext >` (p. 221).

Definition at line 101 of file Traverser.cpp.

References `COrQueryNode::getLeftNode()`, `CQueryContext::getResult()`, `COrQueryNode::getRightNode()`, `CQueryContext::isFalseW()`, `CQueryContext::isTrueW()`, `PostAction()`, `PreAction()`, `CQueryContext::resetResult()`, and `CQueryContext::setResult()`.

6.43.3.3 CQueryContext CTraverser::Visit (CNotQueryNode &, CQueryContext) [virtual]

Abstract callback from the guest.

Implements `visitor_arg< CNotQueryNode, CQueryContext, CQueryContext >` (p. 221).

Definition at line 146 of file Traverser.cpp.

References `CNotQueryNode::getNode()`, `CQueryContext::getResult()`, `CQueryContext::negate()`, `PostAction()`, `PreAction()`, and `CQueryContext::setResult()`.

6.43.3.4 CQueryContext CTraverser::Visit (CTrueQuery &, CQueryContext) [virtual]

Abstract callback from the guest.

Implements `visitor_arg< CTrueQuery, CQueryContext, CQueryContext >` (p. 221).

Definition at line 164 of file Traverser.cpp.

References `PostAction()`, and `PreAction()`.

6.43.3.5 CQueryContext CTraverser::Visit (CFalseQuery &, CQueryContext) [virtual]

Abstract callback from the guest.

Implements `visitor_arg< CFalseQuery, CQueryContext, CQueryContext >` (p. 221).

Definition at line 174 of file Traverser.cpp.

References `PostAction()`, and `PreAction()`.

6.43.3.6 CQueryContext CTraverser::Visit (CExplicitQuery &, CQueryContext) [virtual]

Abstract callback from the guest.

Implements `visitor_arg< CExplicitQuery, CQueryContext, CQueryContext >` (p. 221).

Definition at line 184 of file Traverser.cpp.

References `CExplicitQuery::getFreeVariables()`, `PostAction()`, and `PreAction()`.

6.43.3.7 bool CTraverser::Visit (CTransformerChain &, CTransformerContext) [virtual]

Abstract callback from the guest.

Implements **visitor_arg**< CTransformerChain, CTransformerContext, bool > (p. 221).

Definition at line 208 of file Traverser.cpp.

References CTransformerChain::getTransformers(), PostAction(), and PreAction().

6.43.3.8 bool CTraverser::Visit (CTransformerSet &, CTransformerContext) [virtual]

Abstract callback from the guest.

Implements **visitor_arg**< CTransformerSet, CTransformerContext, bool > (p. 221).

Reimplemented in **CInterpreter** (p. 73).

Definition at line 227 of file Traverser.cpp.

References CTransformerSet::getTransformers(), PostAction(), and PreAction().

6.43.3.9 bool CTraverser::Visit (CRule &, CTransformerContext) [virtual]

Abstract callback from the guest.

Implements **visitor_arg**< CRule, CTransformerContext, bool > (p. 221).

Definition at line 252 of file Traverser.cpp.

References CRule::getQuery(), CRule::getThenTransformer(), CQueryContext::isTrueW(), PostAction(), PreAction(), RuleMidAction(), and CTransformerContext::unify().

6.43.3.10 bool CTraverser::Visit (CExplicitUpdate &, CTransformerContext) [virtual]

Abstract callback from the guest.

Implements **visitor_arg**< CExplicitUpdate, CTransformerContext, bool > (p. 221).

Definition at line 283 of file Traverser.cpp.

References PostAction(), and PreAction().

6.43.3.11 bool CTraverser::Visit (CNoUpdate &, CTransformerContext) [virtual]

Abstract callback from the guest.

Implements **visitor_arg**< CNoUpdate, CTransformerContext, bool > (p. 221).

Definition at line 296 of file Traverser.cpp.

References PostAction(), and PreAction().

6.43.3.12 `bool CTraverser::Visit (CExitUpdate &, CTransformerContext)` [virtual]

Abstract callback from the guest.

Implements `visitor_arg< CExitUpdate, CTransformerContext, bool >` (p. 221).

Definition at line 309 of file `Traverser.cpp`.

References `PostAction()`, and `PreAction()`.

6.43.3.13 `bool CTraverser::Visit (CElseRule &, CTransformerContext)` [virtual]

Abstract callback from the guest.

Implements `visitor_arg< CElseRule, CTransformerContext, bool >` (p. 221).

Definition at line 323 of file `Traverser.cpp`.

References `CElseRule::getElseTransformer()`, `CRule::getQuery()`, `CRule::getThenTransformer()`, `CQueryContext::isFalseW()`, `CQueryContext::isTrueW()`, `PostAction()`, `PreAction()`, `RuleMidAction()`, and `CTransformerContext::unify()`.

6.43.3.14 `CQueryContext & CTraverser::PreAction (CAndQueryNode & guest, CQueryContext & qCtx)` [protected, virtual]

Action invoked before the node is being visited.

Allows additional context tweaking.

By default executes corresponding `DefaultPreAction()` (p. 195) method.

Note: Method is always implemented as empty in the base class so that subclasses do not have to implement all the variants corresponding to the whole `CExpression` (p. 62) hierarchy, but only needed ones. In the default case the call lands in the empty call of the base class.

Reimplemented in `CPrettyPrinter` (p. 121).

Definition at line 370 of file `Traverser.cpp`.

References `DefaultPreAction()`.

Referenced by `Visit()`, and `CInterpreter::Visit()`.

6.43.3.15 `CQueryContext & CTraverser::PreAction (CORQueryNode & guest, CQueryContext & qCtx)` [protected, virtual]

Reimplemented in `CPrettyPrinter` (p. 121).

Definition at line 376 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.16 `CQueryContext & CTraverser::PreAction (CNotQueryNode & guest, CQueryContext & qCtx)` [protected, virtual]

Reimplemented in `CPrettyPrinter` (p. 121).

Definition at line 382 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.17 `CQueryContext & CTraverser::PreAction (CTrueQuery & quest, CQueryContext & qCtx)` [protected, virtual]

Reimplemented in `CPrettyPrinter` (p.122).

Definition at line 388 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.18 `CQueryContext & CTraverser::PreAction (CFalseQuery & quest, CQueryContext & qCtx)` [protected, virtual]

Reimplemented in `CPrettyPrinter` (p.122).

Definition at line 394 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.19 `CQueryContext & CTraverser::PreAction (CExplicitQuery & quest, CQueryContext & qCtx)` [protected, virtual]

Reimplemented in `CCompiler` (p.37), and `CPrettyPrinter` (p.122).

Definition at line 400 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.20 `void CTraverser::PreAction (CTransformerChain & quest, CTransformerContext & tCtx)` [protected, virtual]

Reimplemented in `CPrettyPrinter` (p.122).

Definition at line 406 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.21 `void CTraverser::PreAction (CTransformerSet & quest, CTransformerContext & tCtx)` [protected, virtual]

Reimplemented in `CPrettyPrinter` (p.122).

Definition at line 412 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.22 `void CTraverser::PreAction (CRule & quest, CTransformerContext & tCtx)` [protected, virtual]

Reimplemented in `CPrettyPrinter` (p.122).

Definition at line 418 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.23 `void CTraverser::PreAction (CExplicitUpdate & guest, CTransformerContext & tCtx)` [protected, virtual]

Reimplemented in `CCompiler` (p. 38), and `CPrettyPrinter` (p. 123).

Definition at line 424 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.24 `void CTraverser::PreAction (CNoUpdate & guest, CTransformerContext & tCtx)` [protected, virtual]

Reimplemented in `CPrettyPrinter` (p. 123).

Definition at line 430 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.25 `void CTraverser::PreAction (CExitUpdate & guest, CTransformerContext & tCtx)` [protected, virtual]

Definition at line 436 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.26 `void CTraverser::PreAction (CElseRule & guest, CTransformerContext & tCtx)` [protected, virtual]

Reimplemented in `CPrettyPrinter` (p. 123).

Definition at line 442 of file `Traverser.cpp`.

References `DefaultPreAction()`.

6.43.3.27 `CQueryContext & CTraverser::PostAction (CAndQueryNode & guest, CQueryContext & qCtx)` [protected, virtual]

Action invoked after the node is being visited.

Allows additional context tweaking.

By default executes corresponding `DefaultPostAction()` (p. 195) method.

Note: See note by `PreAction`.

Definition at line 448 of file `Traverser.cpp`.

References `DefaultPostAction()`.

Referenced by `Visit()`.

6.43.3.28 `CQueryContext & CTraverser::PostAction (COrQueryNode & guest, CQueryContext & qCtx)` [protected, virtual]

Definition at line 454 of file `Traverser.cpp`.

References `DefaultPostAction()`.

6.43.3.29 CQueryContext & CTraverser::PostAction (CNotQueryNode & *guest*, CQueryContext & *qCtx*) [protected, virtual]

Definition at line 460 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.30 CQueryContext & CTraverser::PostAction (CTrueQuery & *guest*, CQueryContext & *qCtx*) [protected, virtual]

Reimplemented in **CInterpreter** (p. 73).

Definition at line 466 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.31 CQueryContext & CTraverser::PostAction (CFalseQuery & *guest*, CQueryContext & *qCtx*) [protected, virtual]

Reimplemented in **CInterpreter** (p. 73).

Definition at line 472 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.32 CQueryContext & CTraverser::PostAction (CExplicitQuery & *guest*, CQueryContext & *qCtx*) [protected, virtual]

Reimplemented in **CInterpreter** (p. 73).

Definition at line 478 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.33 void CTraverser::PostAction (CTransformerChain & *guest*, CTransformerContext & *tCtx*) [protected, virtual]

Definition at line 484 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.34 void CTraverser::PostAction (CTransformerSet & *guest*, CTransformerContext & *tCtx*) [protected, virtual]

Definition at line 490 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.35 void CTraverser::PostAction (CRule & *guest*, CTransformerContext & *tCtx*) [protected, virtual]

Definition at line 496 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.36 void CTraverser::PostAction (CExplicitUpdate & *guest*, CTransformerContext & *tCtx*) [protected, virtual]

Reimplemented in CInterpreter (p. 74).

Definition at line 502 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.37 void CTraverser::PostAction (CNopUpdate & *guest*, CTransformerContext & *tCtx*) [protected, virtual]

Definition at line 508 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.38 void CTraverser::PostAction (CExitUpdate & *guest*, CTransformerContext & *tCtx*) [protected, virtual]

Reimplemented in CInterpreter (p. 74).

Definition at line 514 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.39 void CTraverser::PostAction (CElseRule & *guest*, CTransformerContext & *tCtx*) [protected, virtual]

Definition at line 520 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.40 void CTraverser::RuleMidAction (CRule & *guest*, CQueryContext & *qCtx*, CTransformerContext & *tCtx*) [protected, virtual]

Action invoked after the rule query is traversed but before the transformer is visited.

Allows additional context tweaking before the -then branch of rules is executed.

In the case the -else branch of CElseRule (p. 44) is about to be executed no context tweaking is allowed because context is not changed by a negative query evaluation.

By default executes DefaultRuleMidAction() (p. 196) method.

Note: See note by PreAction.

Definition at line 526 of file Traverser.cpp.

References DefaultRuleMidAction().

Referenced by Visit().

6.43.3.41 void CTraverser::RuleMidAction (CElseRule & *guest*, CQueryContext & *qCtx*, CTransformerContext & *tCtx*) [protected, virtual]

Definition at line 532 of file Traverser.cpp.

References DefaultRuleMidAction().

6.43.3.42 CQueryContext & CTraverser::DefaultPreAction (CExpression &, CQueryContext & *qCtx*) [protected, virtual]

Default Pre-/Post- and MidRule- action implementations.

By default all the generic implementations provided in the **CTraverser** (p.184) protected interface, simply link to the default actions. In the case Pre-/Post-, or MidRule- actions are implemented by the same routine in the whole Traverser specialisation, simply overload these methods.

Finally, all Default*Action methods simply link to the Default*Action() methods.

Definition at line 538 of file Traverser.cpp.

References DefaultPreAction().

6.43.3.43 void CTraverser::DefaultPreAction (CExpression &, CTransformerContext &) [protected, virtual]

Definition at line 546 of file Traverser.cpp.

References DefaultPreAction().

6.43.3.44 CQueryContext & CTraverser::DefaultPostAction (CExpression &, CQueryContext & *qCtx*) [protected, virtual]

Definition at line 552 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.45 void CTraverser::DefaultPostAction (CExpression &, CTransformerContext &) [protected, virtual]

Definition at line 560 of file Traverser.cpp.

References DefaultPostAction().

6.43.3.46 void CTraverser::DefaultRuleMidAction (CExpression &, CQueryContext &, CTransformerContext &) [protected, virtual]

Definition at line 566 of file Traverser.cpp.

References DefaultRuleMidAction().

6.43.3.47 void CTraverser::DefaultPreAction () [protected, virtual]

Definition at line 572 of file Traverser.cpp.

Referenced by DefaultPreAction(), and PreAction().

6.43.3.48 void CTraverser::DefaultPostAction () [protected, virtual]

Reimplemented in **CPrettyPrinter** (p.123).

Definition at line 578 of file Traverser.cpp.

Referenced by DefaultPostAction(), and PostAction().

6.43.3.49 void CTraverser::DefaultRuleMidAction () [protected, virtual]

Definition at line 584 of file Traverser.cpp.

Referenced by DefaultRuleMidAction(), and RuleMidAction().

6.43.3.50 void CTraverser::traverse (TPtrProgram *pProgram*) [protected, virtual]

Start the tree structure traversal.

To be used only in the subclasses. The traversal start is always wrapped in some service interface.

Definition at line 51 of file Traverser.cpp.

Referenced by CCompiler::compile(), CInterpreter::doCycle(), and CPrettyPrinter::dump().

6.43.3.51 CTraverser& CTraverser::operator= (const CTraverser &) [private]

No assignment defined.

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

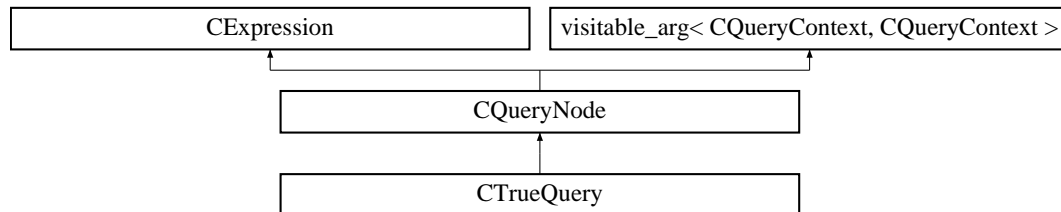
- Traverser.hpp
- Traverser.cpp

6.44 CTrueQuery Class Reference

Trivial True query.

```
#include <TrivialQuery.hpp>
```

Inheritance diagram for CTrueQuery::



Public Member Functions

- **CTrueQuery** ()
Default constructor.
- virtual **~CTrueQuery** ()
Virtual destructor.

Private Member Functions

- **CTrueQuery** (const **CTrueQuery** &)
No copying defined.
- **CTrueQuery** & **operator=** (const **CTrueQuery** &)
No assignment defined.

6.44.1 Detailed Description

Trivial True query.

Query execution always returns true.

Definition at line 42 of file TrivialQuery.hpp.

6.44.2 Constructor & Destructor Documentation

6.44.2.1 CTrueQuery::CTrueQuery () [inline]

Default constructor.

Definition at line 104 of file TrivialQuery.hpp.

6.44.2.2 CTrueQuery::~~CTrueQuery () [inline, virtual]

Virtual destructor.

Definition at line 109 of file TrivialQuery.hpp.

6.44.2.3 CTrueQuery::CTrueQuery (const CTrueQuery &) [private]

No copying defined.

6.44.3 Member Function Documentation**6.44.3.1 CTrueQuery& CTrueQuery::operator= (const CTrueQuery &) [private]**

No assignment defined.

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

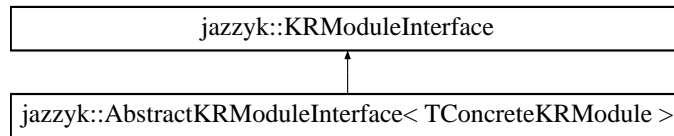
- **TrivialQuery.hpp**

6.45 jazzyk::KRModuleInterface Class Reference

Abstract base class for defining knowledge representation modules.

```
#include <jazzyk_common.hpp>
```

Inheritance diagram for jazzyk::KRModuleInterface::



Public Member Functions

- virtual **EKRModuleError** __JZDLL_CALL **initialize** (const char *szCodeBlock)=0
KR module initialization.
- virtual **EKRModuleError** __JZDLL_CALL **finalize** (const char *szCodeBlock)=0
Invokes the finalize function.
- virtual **EKRModuleError** __JZDLL_CALL **cycle** (const char *szCodeBlock)=0
Invokes the cycle function.
- virtual **EKRModuleError** __JZDLL_CALL **getQueryInterface** (char **&arrQueryOperations, unsigned int &nSize)=0
Fills an array of available query operations into the argument empty pointer.
- virtual **EKRModuleError** __JZDLL_CALL **getUpdateInterface** (char **&arrUpdateOperations, unsigned int &nSize)=0
Fills an array of available update operations into the argument empty pointer.
- virtual **EKRModuleError** __JZDLL_CALL **query** (const char *szOperation, const char *szQueryCode, const char **arrInVarID, const char **arrInVarValue, const unsigned int nInArrSize, char **&arrOutVarID, char **&arrOutVarValue, unsigned int &nOutArrSize, bool &bResult)=0
Query to the KR module.
- virtual **EKRModuleError** __JZDLL_CALL **update** (const char *szOperation, const char *szQueryCode, const char **arrVarID, const char **arrVarValue, const unsigned int nArrSize)=0
Update the KR module.
- virtual void __JZDLL_CALL **free_resource** (char **&array, unsigned int nSize)=0
Helper method for correct variable substitution deallocation.
- void **operator delete** (void *ptr)
Specialized delete that calls destroy instead of the destructor.

Protected Member Functions

- virtual void `__JZDLL_CALL` **destroy** ()=0

Proprietary destruction of the module interface.

6.45.1 Detailed Description

Abstract base class for defining knowledge representation modules.

The class precisely defines the interface of a module, which all the plugins have to comply to. It is used only on the server side (main process) as a handle to the module DLL interface.

It correctly defines the API instance destruction according to the article <http://aegisknight.org/cppinterface.html> by Chad Austin.

Definition at line 91 of file `jazzyk_common.hpp`.

6.45.2 Member Function Documentation

- 6.45.2.1** virtual `EKRModuleError` `__JZDLL_CALL`
jazzyk::KRModuleInterface::initialize (`const char *`
`szCodeBlock`) [pure virtual]

KR module initialization.

To be called at the very beginning of the module usage right after the module is opened. It is supposed to prepare the module for the execution of the application.

Implemented in `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >` (p. 20).

- 6.45.2.2** virtual `EKRModuleError` `__JZDLL_CALL`
jazzyk::KRModuleInterface::finalize (`const char *`
`szCodeBlock`) [pure virtual]

Invokes the finalize function.

To be called when the module is being closed and de-initialized. It is supposed to provide an opportunity to clean up all the held resources.

Implemented in `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >` (p. 20).

- 6.45.2.3** virtual `EKRModuleError` `__JZDLL_CALL`
jazzyk::KRModuleInterface::cycle (`const char *`
`szCodeBlock`) [pure virtual]

Invokes the cycle function.

It is invoked at the end of each deliberation cycle of the main interpreter. It is supposed to clean up all the held resources which were acquired during the deliberation cycle execution (possible caching).

Implemented in `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >` (p. 21).

6.45.2.4 virtual EKRModuleError __JZDLL_CALL
jazzyk::KRModuleInterface::getQueryInterface (char **&
arrQueryOperations, unsigned int & *nSize*) [pure virtual]

Fills an array of available query operations into the argument empty pointer.

The method *replaces* the argument pointer, so the memory gets lost! Use empty pointers only and then do not forget to free the allocated memory using `free_resource(...)`.

The second argument receives the size of the returned array.

Implemented in **jazzyk::AbstractKRModuleInterface**< **TConcreteKRModule** > (p. 22).

6.45.2.5 virtual EKRModuleError __JZDLL_CALL
jazzyk::KRModuleInterface::getUpdateInterface (char **&
arrUpdateOperations, unsigned int & *nSize*) [pure virtual]

Fills an array of available update operations into the argument empty pointer.

The method *replaces* the argument pointer, so the memory gets lost! Use empty pointers only and then do not forget to free the allocated memory using `free_resource(...)`.

The second argument receives the size of the returned array.

Implemented in **jazzyk::AbstractKRModuleInterface**< **TConcreteKRModule** > (p. 22).

6.45.2.6 virtual EKRModuleError __JZDLL_CALL
jazzyk::KRModuleInterface::query (const char * *szOperation*,
const char * *szQueryCode*, const char ** *arrInVarID*, const char **
arrInVarValue, const unsigned int *nInArrSize*, char **& *arrOutVarID*,
char **& *arrOutVarValue*, unsigned int & *nOutArrSize*, bool & *bResult*)
[pure virtual]

Query to the KR module.

Takes a query formula (code), an input variable substitution for this formula, and a query operation to perform. Returns a truth value of the query operation and a new variable substitution (result of the query).

IMPORTANT: After successful processing of the output substitution, the output substitution has to be correctly de-allocated using `free_resource(.)` method below.

Implemented in **jazzyk::AbstractKRModuleInterface**< **TConcreteKRModule** > (p. 21).

6.45.2.7 virtual EKRModuleError __JZDLL_CALL
jazzyk::KRModuleInterface::update (const char * *szOperation*,
const char * *szQueryCode*, const char ** *arrVarID*, const char **
arrVarValue, const unsigned int *nArrSize*) [pure virtual]

Update the KR module.

Takes an update formula, a variable substitution for this formula and an update operation to perform.

Implemented in `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >` (p. 21).

6.45.2.8 `virtual void __JZDLL_CALL jazzyk::KRModuleInterface::free_resource (char **& array, unsigned int nSize) [pure virtual]`

Helper method for correct variable substitution deallocation.

Implemented in `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >` (p. 22).

6.45.2.9 `virtual void __JZDLL_CALL jazzyk::KRModuleInterface::destroy () [protected, pure virtual]`

Proprietary destruction of the module interface.

Handles the destruction of this class. This is essentially a destructor that works across DLL boundaries.

Implemented in `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >` (p. 23).

Referenced by operator `delete()`.

6.45.2.10 `void jazzyk::KRModuleInterface::operator delete (void * ptr) [inline]`

Specialized delete that calls `destroy` instead of the destructor.

A kind of virtual destructor invoked via DLL boundary. For more documentation, see <http://aegisknight.org/cppinterface.html>. Designed and inspired by this article.

Reimplemented in `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >` (p. 23).

Definition at line 187 of file `jazzyk_common.hpp`.

References `destroy()`.

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

- `jazzyk_common.hpp`

6.46 ShmemLock Struct Reference

Specialised shared memory lock.

```
#include <ShmemManager.hpp>
```

Public Member Functions

- **ShmemLock** ()
Constructor.

6.46.1 Detailed Description

Specialised shared memory lock.

The lock is specific for the memory managed by the **CShmemManager** (p. 145) singleton.

Definition at line 327 of file ShmemManager.hpp.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 ShmemLock::ShmemLock ()

Constructor.

Creates a lock specific to the shared memory managed by the **CShmemManager** (p. 145) singleton. For that, it needs to retrieve the mutex from the singleton. Hence the struct is a friend class of **CShmemManager** (p. 145).

Definition at line 308 of file ShmemManager.cpp.

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

- **ShmemManager.hpp**
- **ShmemManager.cpp**

6.47 SIdentifierCompare Struct Reference

Comparison operator.

```
#include <IdentifierMap.hpp>
```

Public Member Functions

- **bool operator()** (const boost::shared_ptr< **CIdentifier** > &s1, const boost::shared_ptr< **CIdentifier** > &s2) const

6.47.1 Detailed Description

Comparison operator.

Definition at line 71 of file IdentifierMap.hpp.

6.47.2 Member Function Documentation

- 6.47.2.1** **bool SIdentifierCompare::operator()** (const boost::shared_ptr< **CIdentifier** > & s1, const boost::shared_ptr< **CIdentifier** > & s2) const
[inline]

Definition at line 73 of file IdentifierMap.hpp.

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

- **IdentifierMap.hpp**

6.48 SIdentifierEqual Struct Reference

Equality operator.

```
#include <IdentifierMap.hpp>
```

Public Member Functions

- `bool operator() (const boost::shared_ptr< CIdentifier > &s1, const boost::shared_ptr< CIdentifier > &s2) const`

6.48.1 Detailed Description

Equality operator.

The program can compile with either `std::map`, or `__gnu_cxx::hash_map` used for `map`, where `__gnu_cxx` is preferred.

TIdentifierMap (p. 213) is an alias for an appropriate map with `TPtrIdentifier` as the key.

STL containers are not polymorphic, so do not store, or copy **TIdentifierMap** (p. 213) to their parent classes!

Definition at line 62 of file `IdentifierMap.hpp`.

6.48.2 Member Function Documentation

- 6.48.2.1** `bool SIdentifierEqual::operator() (const boost::shared_ptr< CIdentifier > & s1, const boost::shared_ptr< CIdentifier > & s2) const [inline]`

Definition at line 64 of file `IdentifierMap.hpp`.

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

- `IdentifierMap.hpp`

6.49 SIpcStringCompare Struct Reference

String comparison operator.

```
#include <ModuleCommData.hpp>
```

Public Member Functions

- `bool operator()` (`const boost::interprocess::string s1`, `const boost::interprocess::string s2`)
const

6.49.1 Detailed Description

String comparison operator.

Helper structure for the variable substitution map.

Definition at line 63 of file `ModuleCommData.hpp`.

6.49.2 Member Function Documentation

- ##### 6.49.2.1 `bool SIpcStringCompare::operator()` (`const boost::interprocess::string s1`, `const boost::interprocess::string s2`) const [inline]

Definition at line 65 of file `ModuleCommData.hpp`.

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

- `ModuleCommData.hpp`

6.50 SModuleRequest Struct Reference

Plug-in request structure.

```
#include <ModuleCommData.hpp>
```

Public Member Functions

- **SModuleRequest** (const **TIPCCharAllocator** &charalloc, const **TIPCSubstAllocator** &strpairalloc)
Default structure constructor.
- void **addVariableSubst** (const char *szVarName, const char *szVarValue)
Adds a new variable substitution.
- void **addFreeVariable** (const char *szVarName)
Adds a new free variable.

Public Attributes

- **EModuleRequestType m_eType**
Request type.
- **TIPCString m_szOperation**
Interface query/update operator name.
- **TIPCString m_szCodeBlock**
Code block associated with the plug-in operation.
- **TIPCSubst m_mapSubstitution**
Variable substitution passed to Query/Update operations.
- **TIPCSubst m_mapFreeVariables**
Initial free variables substitution.

Private Attributes

- const **TIPCCharAllocator** & **m_charAllocator**
String allocator for adding new map pairs.

6.50.1 Detailed Description

Plug-in request structure.

Definition at line 155 of file ModuleCommData.hpp.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 SModuleRequest::SModuleRequest (const TIPCCharAllocator & *charalloc*, const TIPCSubstAllocator & *strpairalloc*) [inline]

Default structure constructor.

Definition at line 324 of file ModuleCommData.hpp.

6.50.3 Member Function Documentation

6.50.3.1 void SModuleRequest::addVariableSubst (const char * *szVarName*, const char * *szVarValue*) [inline]

Adds a new variable substitution.

Definition at line 334 of file ModuleCommData.hpp.

References `m_charAllocator`, and `m_mapSubstitution`.

6.50.3.2 void SModuleRequest::addFreeVariable (const char * *szVarName*) [inline]

Adds a new free variable.

Definition at line 342 of file ModuleCommData.hpp.

References `m_charAllocator`, and `m_mapFreeVariables`.

6.50.4 Member Data Documentation

6.50.4.1 EModuleRequestType SModuleRequest::m_eType

Request type.

Before manipulating the communicated data further, the type has to be examined and then only the appropriate data members should be manipulated. For details see the documentation for each member of the structure.

Type cannot be NOP after the data structure is received. If so, an error should be raised.

Definition at line 167 of file ModuleCommData.hpp.

Referenced by `CSingleModule::processRequest()`.

6.50.4.2 TIPCString SModuleRequest::m_szOperation

Interface query/update operator name.

In the case `m_eType` is QUERY/UPDATE, the member contains the name of the plug-in API operator to be invoked on the communicated variable substitution. Otherwise the content is undefined and shouldn't be touched.

Definition at line 176 of file ModuleCommData.hpp.

Referenced by `CSingleModule::processRequest()`.

6.50.4.3 TIpCString SModuleRequest::m_szCodeBlock

Code block associated with the plug-in operation.

In the case m_eType is QUERY/UPDATE/INITIALIZE/FINALIZE. The member contains a code block used in the operation invocation. Otherwise its content is undefined and shouldn't be touched.

Definition at line 184 of file ModuleCommData.hpp.

Referenced by CSingleModule::processRequest().

6.50.4.4 TIpSubst SModuleRequest::m_mapSubstitution

Variable substitution passed to Query/Update operations.

Query/Update operations receive a variable substitution to process. It is a map of string pairs. On other than QUERY/UPDATE request types, the member shouldn't be touched. Its content is undefined.

Definition at line 192 of file ModuleCommData.hpp.

Referenced by addVariableSubst(), and CSingleModule::processRequest().

6.50.4.5 TIpSubst SModuleRequest::m_mapFreeVariables

Initial free variables substitution.

Map of free variables with initialized empty values. The map should be filled by the module and then sent back in the response.

Effectively that means that a module has no powers over the already substituted variables and can change only free ones.

NOTE: This data structure is also used as a store for update interface in the GETINTERFACE message (the key indicates the operation identifier and the value the type of the operation ("query"/"update").

Of course this is not clear implementation, however it saves memory and does not harm.

Definition at line 211 of file ModuleCommData.hpp.

Referenced by addFreeVariable(), and CSingleModule::processRequest().

6.50.4.6 const TIpCharAllocator& SModuleRequest::m_charAllocator [private]

String allocator for adding new map pairs.

Definition at line 229 of file ModuleCommData.hpp.

Referenced by addFreeVariable(), and addVariableSubst().

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

- **ModuleCommData.hpp**

6.51 SModuleResponse Struct Reference

Plug-in response structure.

```
#include <ModuleCommData.hpp>
```

Public Member Functions

- **SModuleResponse** (const **TIPCCharAllocator** &charalloc, const **TIPCSubstAllocator** &strpairalloc)
Default structure constructor.
- void **addVariableSubst** (const char *szVarName, const char *szVarValue)
Adds a new variable substitution.

Public Attributes

- **EModuleErrCode** **m_eError**
Response error code.
- **TIPCSubst** **m_mapSubstitution**
Response variable substitution.
- bool **m_bQueryResult**
Query result.

Private Attributes

- const **TIPCCharAllocator** & **m_charAllocator**
String allocator for adding new map pairs.

6.51.1 Detailed Description

Plug-in response structure.

Definition at line 234 of file ModuleCommData.hpp.

6.51.2 Constructor & Destructor Documentation

6.51.2.1 SModuleResponse::SModuleResponse (const TIPCCharAllocator & charalloc, const TIPCSubstAllocator & strpairalloc) [inline]

Default structure constructor.

Definition at line 353 of file ModuleCommData.hpp.

6.51.3 Member Function Documentation

6.51.3.1 void SModuleResponse::addVariableSubst (const char * *szVarName*, const char * *szVarValue*) [inline]

Adds a new variable substitution.

Definition at line 361 of file ModuleCommData.hpp.

References `m_charAllocator`, and `m_mapSubstitution`.

Referenced by `CSingleModule::writeResponse()`.

6.51.4 Member Data Documentation

6.51.4.1 EModuleErrCode SModuleResponse::m_eError

Response error code.

OK indicates that technically everything is alright. Otherwise an error code is reported.

Definition at line 241 of file ModuleCommData.hpp.

Referenced by `CSingleModule::writeResponse()`.

6.51.4.2 TIPCSubst SModuleResponse::m_mapSubstitution

Response variable substitution.

After processing QUERY/UPDATE request type, a resulting variable substitution is returned. It is defined only for QUERY/UPDATE response types. Otherwise shouldn't be manipulated.

Definition at line 249 of file ModuleCommData.hpp.

Referenced by `addVariableSubst()`.

6.51.4.3 bool SModuleResponse::m_bQueryResult

Query result.

Queries return a truth value in the response. The member is defined only for QUERY response type and shouldn't be manipulated otherwise.

Definition at line 256 of file ModuleCommData.hpp.

Referenced by `CSingleModule::writeResponse()`.

6.51.4.4 const TIPCCharAllocator& SModuleResponse::m_charAllocator [private]

String allocator for adding new map pairs.

Definition at line 271 of file ModuleCommData.hpp.

Referenced by `addVariableSubst()`.

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

- **ModuleCommData.hpp**

6.52 jazzyk_private::StringCompare Struct Reference

String comparison operator.

```
#include <jazzyk.hpp>
```

Public Member Functions

- **bool operator()** (const std::string &s1, const std::string s2) const

6.52.1 Detailed Description

String comparison operator.

Definition at line 54 of file jazzyk.hpp.

6.52.2 Member Function Documentation

- 6.52.2.1** **bool jazzyk_private::StringCompare::operator()** (const std::string & *s1*, const std::string *s2*) const [inline]

Definition at line 56 of file jazzyk.hpp.

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

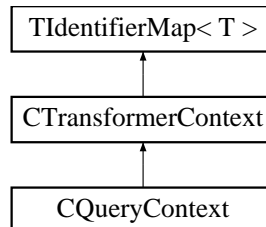
- **jazzyk.hpp**

6.53 TIdentifierMap< T > Class Template Reference

CIdentifier (p. 67) map.

```
#include <IdentifierMap.hpp>
```

Inheritance diagram for TIdentifierMap< T >::



6.53.1 Detailed Description

```
template<class T> class TIdentifierMap< T >
```

CIdentifier (p. 67) map.

The specialized map is able to return value of the map based on the TPtrIdentifier argument.

CIdentifier (p. 67) has a built in conversion to 'const char *'.

Definition at line 130 of file IdentifierMap.hpp.

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

- IdentifierMap.hpp

6.54 visitable< R > Class Template Reference

Base class for **visitable** (p. 214) classes.

```
#include <visitor.hpp>
```

Public Types

- typedef R **TReturnType**
*Return type of **Accept()** (p. 215).*

Public Member Functions

- virtual **~visitable** ()
Virtual destructor.
- virtual **TReturnType Accept** (**base_visitor** &)=0
Abstract accepting method.

Static Protected Member Functions

- template<class T>
static **TReturnType AcceptImpl** (T &host, **base_visitor** &guest)
*Bounce back to allow the macro **DEFINE_VISITABLE** call **Accept()** (p. 215).*

6.54.1 Detailed Description

```
template<typename R = void> class visitable< R >
```

Base class for **visitable** (p. 214) classes.

Visitable class just bounces back the Visit callback. The top parent of the **visitable** (p. 214) hierarchy has to inherit from **visitable** (p. 214) and each **visitable** (p. 214) subclass has to define method **Accept**. This is automatized by macro **DECLARE_VISITABLE**

Definition at line 117 of file visitor.hpp.

6.54.2 Member Typedef Documentation

6.54.2.1 template<typename R = void> typedef R visitable< R >::**TReturnType**

Return type of **Accept()** (p. 215).

Definition at line 121 of file visitor.hpp.

6.54.3 Constructor & Destructor Documentation

6.54.3.1 `template<typename R = void> virtual visitable< R >::~~visitable ()`
[inline, virtual]

Virtual destructor.

Definition at line 124 of file visitor.hpp.

6.54.4 Member Function Documentation

6.54.4.1 `template<typename R = void> virtual TReturnType visitable< R`
`>::Accept (base_visitor &) [pure virtual]`

Abstract accepting method.

To be overwritten in the subclasses. It is an obligatory interface to allow callbacks.

Macro DECLARE_VISITABLE generates the corresponding invocation in subclasses.

6.54.4.2 `template<typename R = void> template<class T> static TReturnType`
`visitable< R >::AcceptImpl (T & host, base_visitor & guest) [inline,`
`static, protected]`

Bounce back to allow the macro DEFINE_VISITABLE call `Accept()` (p. 215).

Invocation generated by DEFINE_VISITABLE.

Definition at line 143 of file visitor.hpp.

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

- `visitor.hpp`

6.55 visitable_arg< A, R > Class Template Reference

Version of `visitable` (p. 214) class with argument.

```
#include <visitor.hpp>
```

Public Types

- typedef R **TReturnType**
Return type of `Accept()` (p. 217).
- typedef A **TArgType**
`Accept()` (p. 217) argument type.

Public Member Functions

- virtual `~visitable_arg ()`
Virtual destructor.
- virtual **TReturnType** `Accept (base_visitor &, TArgType)=0`
Abstract accepting method.

Static Protected Member Functions

- `template<class T>`
`static TReturnType AcceptImpl (T &host, base_visitor &guest, TArgType argu-`
`ment)`
Bounce back to allow the macro `DEFINE_VISITABLE` call `Accept()` (p. 217).

6.55.1 Detailed Description

```
template<typename A, typename R = void> class visitable_arg< A, R >
```

Version of `visitable` (p. 214) class with argument.

Allows passing an argument to `Accept()` (p. 217) method. The argument is passed by value so the type A has to be copy-constructible.

Definition at line 162 of file `visitor.hpp`.

6.55.2 Member Typedef Documentation

6.55.2.1 `template<typename A, typename R = void> typedef R visitable_arg< A, R >::TReturnType`

Return type of `Accept()` (p. 217).

Definition at line 166 of file `visitor.hpp`.

6.55.2.2 `template<typename A, typename R = void> typedef A visitable_arg< A, R >::TArgType`

`Accept()` (p. 217) argument type.

Definition at line 169 of file `visitor.hpp`.

6.55.3 Constructor & Destructor Documentation

6.55.3.1 `template<typename A, typename R = void> virtual visitable_arg< A, R >::~~visitable_arg () [inline, virtual]`

Virtual destructor.

Definition at line 172 of file `visitor.hpp`.

6.55.4 Member Function Documentation

6.55.4.1 `template<typename A, typename R = void> virtual TReturnType visitable_arg< A, R >::Accept (base_visitor &, TArgType) [pure virtual]`

Abstract accepting method.

To be overwritten in the subclasses. It is an obligatory interface to allow callbacks.

Macro `DECLARE_VISITABLE` generates the corresponding invocation in subclasses.

6.55.4.2 `template<typename A, typename R = void> template<class T> static TReturnType visitable_arg< A, R >::AcceptImpl (T & host, base_visitor & guest, TArgType argument) [inline, static, protected]`

Bounce back to allow the macro `DEFINE_VISITABLE` call `Accept()` (p. 217).

Invocation generated by `DEFINE_VISITABLE`.

Definition at line 191 of file `visitor.hpp`.

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

- `visitor.hpp`

6.56 visitor< T, R > Class Template Reference

Visitor class template.

```
#include <visitor.hpp>
```

Public Types

- typedef R **TReturnType**
Return type of `Visit()` (p. 219).

Public Member Functions

- virtual **TReturnType Visit** (T &)=0
Abstract callback from the guest.
- virtual **~visitor** ()
Virtual destructor.

6.56.1 Detailed Description

```
template<class T, typename R = void> class visitor< T, R >
```

Visitor class template.

T is the visited class and R is the return type of the `Visitor()` method.

Definition at line 64 of file `visitor.hpp`.

6.56.2 Member Typedef Documentation

6.56.2.1 `template<class T, typename R = void> typedef R visitor< T, R >::TReturnType`

Return type of `Visit()` (p. 219).

Definition at line 68 of file `visitor.hpp`.

6.56.3 Constructor & Destructor Documentation

6.56.3.1 `template<class T, typename R = void> virtual visitor< T, R >::~~visitor () [inline, virtual]`

Virtual destructor.

Unnecessary. However gcc -Wall complains about it when missing.

Definition at line 77 of file `visitor.hpp`.

6.56.4 Member Function Documentation

6.56.4.1 `template<class T, typename R = void> virtual TReturnType visitor< T, R >::Visit (T &) [pure virtual]`

Abstract callback from the guest.

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

- `visitor.hpp`

6.57 visitor_arg< T, A, R > Class Template Reference

Visitor passing an argument class template.

```
#include <visitor.hpp>
```

Public Types

- typedef R **TReturnType**
*Return type of **Visit()** (p. 221).*
- typedef A **TArgType**
Visit argument type.

Public Member Functions

- virtual R **Visit** (T &, **TArgType**)=0
Abstract callback from the guest.
- virtual ~**visitor_arg** ()
Virtual destructor.

6.57.1 Detailed Description

```
template<class T, typename A, typename R = void> class visitor_arg< T, A, R >
```

Visitor passing an argument class template.

Specialized version of **visitor** (p. 218), which is able to pass an argument to the **Visit()** (p. 221) method.

T is the visited class and R is the return type of the **Visitor()** method. A is a **Visit()** (p. 221) method argument type.

Note: Argument to **Visit()** (p. 221) is passed by value, so A has to be copy constructible type. The price for the use of this type of **visitor** (p. 218) is that the argument object is copied 3 times until it reaches the **visitor** (p. 218) object. Therefore use only efficiently copy-able object as arguments.

Definition at line 94 of file visitor.hpp.

6.57.2 Member Typedef Documentation

6.57.2.1 template<class T, typename A, typename R = void> typedef R
visitor_arg< T, A, R >::TReturnType

Return type of **Visit()** (p. 221).

Definition at line 98 of file visitor.hpp.

6.57.2.2 `template<class T, typename A, typename R = void> typedef A
visitor_arg< T, A, R >::TArgType`

Visit argument type.

Definition at line 101 of file visitor.hpp.

6.57.3 Constructor & Destructor Documentation

6.57.3.1 `template<class T, typename A, typename R = void> virtual visitor_arg<
T, A, R >::~~visitor_arg () [inline, virtual]`

Virtual destructor.

Definition at line 107 of file visitor.hpp.

6.57.4 Member Function Documentation

6.57.4.1 `template<class T, typename A, typename R = void> virtual R
visitor_arg< T, A, R >::Visit (T &, TArgType) [pure virtual]`

Abstract callback from the guest.

Implemented in `CInterpreter` (p. 73), `CTraverser` (p. 187), `CTraverser` (p. 188), `CTraverser` (p. 188), `CTraverser` (p. 188), `CTraverser` (p. 188), `CTraverser` (p. 188), `CTraverser` (p. 189), `CTraverser` (p. 189), `CTraverser` (p. 189), `CTraverser` (p. 189), `CTraverser` (p. 189), `CTraverser` (p. 189), `CTraverser` (p. 189), `CTraverser` (p. 190), and `CTraverser` (p. 190).

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

- `visitor.hpp`

Chapter 7

File Documentation

7.1 AndQueryNode.hpp File Reference

```
#include "QueryNode.hpp"
```

Classes

- class **CAndQueryNode**
Query node for AND logical connector of queries.

7.2 CodeBlock.cpp File Reference

```
#include "CodeBlock.hpp"
```

7.3 CodeBlock.hpp File Reference

```
#include <string>
```

Classes

- class **CCodeBlock**
Code of block holder.

7.4 Compiler.cpp File Reference

```
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <signal.h>
#include "Error.hpp"
#include "Compiler.hpp"
#include "Context.hpp"
#include "Parser.h++"
#include "ParserAux.hpp"
#include <string>
#include <sstream>
#include <vector>
#include <cerrno>
```

7.5 Compiler.hpp File Reference

```
#include "ExpressionsFwd.hpp"
#include "Traverser.hpp"
#include "visitor.hpp"
#include <string>
#include <sstream>
#include <vector>
#include <iostream>
#include <unistd.h>
#include <sys/types.h>
```

Classes

- class **CCompiler**
Jazzyk programming language compiler.
- struct **CCompiler::SMPInvocationSettings**
Configuration structure for the macro preprocessor invocation.

7.6 Context.cpp File Reference

```
#include "Context.hpp"  
#include "Program.hpp"
```

7.7 Context.hpp File Reference

```
#include "ExpressionsFwd.hpp"
```

```
#include "IdentifierMap.hpp"
```

Classes

- class **CTransformerContext**
Interpreter context used while traversing the transformer tree structure.
- class **CQueryContext**
Interpreter context used while traversing the query subtree.

7.8 ElseRule.hpp File Reference

```
#include "ExpressionsFwd.hpp"  
#include "Rule.hpp"
```

Classes

- class **CElseRule**
If-then-else type rule.

7.9 Error.cpp File Reference

```
#include "Error.hpp"
#include <iostream>
#include <sstream>
```

Functions

- `std::ostream & operator<< (std::ostream &ostr, const CError &err)`
Dumps the error to the output.

7.9.1 Function Documentation

7.9.1.1 `std::ostream& operator<< (std::ostream & ostr, const CError & err)`

Dumps the error to the output.

Definition at line 120 of file Error.cpp.

References CError::m_szErrText.

7.10 Error.hpp File Reference

```
#include <string>
#include <map>
#include <exception>
#include <ostream>
#include "location.hh"
```

Classes

- class **CError**
Error handler for the application.

Functions

- `std::ostream & operator<< (std::ostream &ostr, const CError &err)`
Dumps the error to the output.

7.10.1 Function Documentation

7.10.1.1 `std::ostream& operator<< (std::ostream & ostr, const CError & err)`

Dumps the error to the output.

Definition at line 120 of file Error.cpp.

References `CError::m_szErrText`.

7.11 ExplicitQuery.hpp File Reference

```
#include "ExpressionsFwd.hpp"  
#include "QueryNode.hpp"
```

Classes

- class **CExplicitQuery**
Explicit query class.

7.12 ExplicitUpdate.hpp File Reference

```
#include "ExpressionsFwd.hpp"  
#include "Transformer.hpp"
```

Classes

- class **CExplicitUpdate**
Explicit update class.

7.13 Expression.cpp File Reference

```
#include "Expression.hpp"  
#include "LocationTracker.hpp"
```

7.14 Expression.hpp File Reference

```
#include "location.hh"
```

Classes

- class **CExpression**
Base abstract class for all the query and update expressions.

Typedefs

- typedef yy::location **TSrcLocation**
Type definition of the location.

7.14.1 Typedef Documentation

7.14.1.1 typedef yy::location TSrcLocation

Type definition of the location.

Definition at line 38 of file Expression.hpp.

7.15 ExpressionsFwd.hpp File Reference

```
#include <boost/shared_ptr.hpp>
#include <list>
#include <vector>
#include <set>
#include "IdentifierMap.hpp"
```

Namespaces

- namespace yy

Typedefs

- typedef yy::location **TSrcLocation**
- typedef boost::shared_ptr< CTransformer > **TPtrTransformer**
- typedef std::list< TPtrTransformer > **TTransformers**
- typedef std::vector< TPtrTransformer > **TVecTransformers**
- typedef boost::shared_ptr< CQueryNode > **TPtrQueryNode**
- typedef boost::shared_ptr< CNopUpdate > **TPtrNopUpdate**
- typedef boost::shared_ptr< CExitUpdate > **TPtrExitUpdate**
- typedef boost::shared_ptr< TTransformers > **TPtrTransformers**
- typedef boost::shared_ptr< CTransformerChain > **TPtrTransformerChain**
- typedef boost::shared_ptr< CTransformerSet > **TPtrTransformerSet**
- typedef boost::shared_ptr< CIdentifier > **TPtrIdentifier**
- typedef std::vector< TPtrIdentifier > **TIdentifiers**
- typedef std::set< TPtrIdentifier, SIdentifierCompare > **TIdentifierSet**
- typedef boost::shared_ptr< TIdentifiers > **TPtrIdentifiers**
- typedef boost::shared_ptr< CCodeBlock > **TPtrCodeBlock**
- typedef boost::shared_ptr< CModuleDeclaration > **TPtrModuleDeclaration**
- typedef boost::shared_ptr< CModuleNotification > **TPtrModuleNotification**
- typedef **TIdentifierMap**< TPtrModuleDeclaration > **TModuleDeclCollection**
- typedef boost::shared_ptr< TModuleDeclCollection > **TPtrModuleDeclCollection**
- typedef boost::shared_ptr< CProgram > **TPtrProgram**
- typedef boost::shared_ptr< CTransformerContext > **TPtrTransformerContext**
- typedef std::string **TVariableValue**
- typedef boost::shared_ptr< TVariableValue > **TPtrVariableValue**
- typedef **TIdentifierMap**< TPtrVariableValue > **TVariableSubstitution**
- typedef boost::shared_ptr< TVariableSubstitution > **TPtrVariableSubstitution**
- typedef boost::shared_ptr< CPlugin > **TPtrPlugin**
- typedef std::vector< std::string > **TVecStrings**

7.15.1 Typedef Documentation

7.15.1.1 typedef std::vector<TPtrIdentifier> TIdentifiers

Definition at line 92 of file ExpressionsFwd.hpp.

7.15.1.2 `typedef std::set<TPtrIdentifier, SIdentifierCompare> TIdentifierSet`

Definition at line 93 of file ExpressionsFwd.hpp.

7.15.1.3 `typedef TIdentifierMap<TPtrModuleDeclaration>
TModuleDeclCollection`

Definition at line 98 of file ExpressionsFwd.hpp.

7.15.1.4 `update TPtrCodeBlock`

Definition at line 95 of file ExpressionsFwd.hpp.

7.15.1.5 `typedef boost::shared_ptr<CExitUpdate> TPtrExitUpdate`

Definition at line 87 of file ExpressionsFwd.hpp.

7.15.1.6 `update TPtrIdentifier`

Definition at line 91 of file ExpressionsFwd.hpp.

7.15.1.7 `update TPtrIdentifiers`

Definition at line 94 of file ExpressionsFwd.hpp.

7.15.1.8 `typedef boost::shared_ptr<CModuleDeclaration>
TPtrModuleDeclaration`

Definition at line 96 of file ExpressionsFwd.hpp.

7.15.1.9 `typedef boost::shared_ptr<TModuleDeclCollection>
TPtrModuleDeclCollection`

Definition at line 99 of file ExpressionsFwd.hpp.

7.15.1.10 `typedef boost::shared_ptr<CModuleNotification>
TPtrModuleNotification`

Definition at line 97 of file ExpressionsFwd.hpp.

7.15.1.11 `typedef boost::shared_ptr<CNopUpdate> TPtrNopUpdate`

Definition at line 86 of file ExpressionsFwd.hpp.

7.15.1.12 `typedef boost::shared_ptr<CPlugin> TPtrPlugin`

Definition at line 107 of file ExpressionsFwd.hpp.

7.15.1.13 `typedef boost::shared_ptr<CProgram> TPtrProgram`

Definition at line 100 of file ExpressionsFwd.hpp.

7.15.1.14 `typedef boost::shared_ptr<CQueryNode> TPtrQueryNode`

Definition at line 85 of file ExpressionsFwd.hpp.

7.15.1.15 `rule TPtrTransformer`

Definition at line 79 of file ExpressionsFwd.hpp.

7.15.1.16 `typedef boost::shared_ptr<CTransformerChain> TPtrTransformerChain`

Definition at line 89 of file ExpressionsFwd.hpp.

7.15.1.17 `typedef boost::shared_ptr<CTransformerContext>
TPtrTransformerContext`

Definition at line 101 of file ExpressionsFwd.hpp.

7.15.1.18 `typedef boost::shared_ptr<TTransformers> TPtrTransformers`

Definition at line 88 of file ExpressionsFwd.hpp.

7.15.1.19 `typedef boost::shared_ptr<CTransformerSet> TPtrTransformerSet`

Definition at line 90 of file ExpressionsFwd.hpp.

7.15.1.20 `typedef boost::shared_ptr<TVariableSubstitution>
TPtrVariableSubstitution`

Definition at line 105 of file ExpressionsFwd.hpp.

7.15.1.21 `typedef boost::shared_ptr<TVariableValue> TPtrVariableValue`

Definition at line 103 of file ExpressionsFwd.hpp.

7.15.1.22 `typedef yy::location TSrcLocation`

Definition at line 52 of file ExpressionsFwd.hpp.

7.15.1.23 `typedef std::list<TPtrTransformer> TTransformers`

Definition at line 83 of file ExpressionsFwd.hpp.

7.15.1.24 `typedef TIdentifierMap<TPtrVariableValue> TVariableSubstitution`

Definition at line 104 of file ExpressionsFwd.hpp.

7.15.1.25 `typedef std::string TVariableValue`

Definition at line 102 of file ExpressionsFwd.hpp.

7.15.1.26 `typedef std::vector<std::string> TVecStrings`

Definition at line 108 of file ExpressionsFwd.hpp.

7.15.1.27 `typedef std::vector<TPtrTransformer> TVecTransformers`

Definition at line 84 of file ExpressionsFwd.hpp.

7.16 Identifier.hpp File Reference

```
#include <string>
```

Classes

- class **CIdentifier**
Identifier holder.

7.17 IdentifierMap.hpp File Reference

```
#include "config.h"
#include "Identifier.hpp"
#include <string>
#include <boost/shared_ptr.hpp>
#include <map>
```

Classes

- struct **SIdentifierEqual**
Equality operator.
- struct **SIdentifierCompare**
Comparison operator.
- class **TIdentifierMap< T >**
***CIdentifier** (p. 67) map.*

7.18 Interpreter.cpp File Reference

```
#include "Interpreter.hpp"
#include "Program.hpp"
#include "TransformerSet.hpp"
#include "TrivialUpdate.hpp"
#include "ExplicitQuery.hpp"
#include "ExplicitUpdate.hpp"
#include "Error.hpp"
#include "ModuleManager.hpp"
#include "ModuleDeclaration.hpp"
#include <algorithm>
#include <functional>
#include <sstream>
```

Functions

- static int **basic_random** (int n)
Basic random numbers generator.

7.18.1 Function Documentation

7.18.1.1 static int basic_random (int n) [static]

Basic random numbers generator.

The stock pseudo-random generator of `random_shuffle` is no good - it generates always the same sequences of permutations.

Definition at line 53 of file `Interpreter.cpp`.

7.19 Interpreter.hpp File Reference

```
#include "ExpressionsFwd.hpp"  
#include "Traverser.hpp"  
#include <functional>
```

Classes

- class **CInterpreter**
Interpreter engine of the Jazzyk language interpreter.
- struct **CInterpreter::SSettings**
Interpreter configuration.

7.20 jazzyk.hpp File Reference

```
#include <string>
#include <map>
#include <iostream>
#include <jazzyk/jazzyk_common.hpp>
```

Namespaces

- namespace **jazzyk_private**
- namespace **jazzyk**

Classes

- struct **jazzyk_private::StringCompare**
String comparison operator.
- class **jazzyk::AbstractKRModuleInterface**< **TConcreteKRModule** >
The abstract knowledge representation module interface adapter.
- class **jazzyk::AbstractKRModule**
Concrete knowledge representation module base class.

Defines

- #define **JZMODULE_MANIFEST_BEGIN**(ModuleClass)
KR module manifest support macros.
- #define **JZMODULE_MANIFEST_END**
Encloses the KR module manifest declaration.
- #define **REGISTER_QUERY**(szMethodID, ptrMethod) pModule → registerQuery(szMethodID, &_TJzModuleClass::ptrMethod);
Query/Update registration macros.
- #define **REGISTER_UPDATE**(szMethodID, ptrMethod) pModule → registerUpdate(szMethodID, &_TJzModuleClass::ptrMethod);
- #define **CALL_MEMBER_FN**(ptrToMember) (m_concreteKRModule.*(ptrToMember))

Helper macro definition for invoking a pointer-to-member function callback.

Typedefs

- typedef std::map< std::string, std::string, **jazzyk_private::StringCompare** > **jazzyk::TKRVarSubstitution**
Type definition for variable substitution map.

7.20.1 Define Documentation

7.20.1.1 `#define CALL_MEMBER_FN(ptrToMember) (m_ - concreteKRModule.*(ptrToMember))`

Helper macro definition for invoking a pointer-to-member function callback.

Definition at line 329 of file jazzyk.hpp.

Referenced by `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::query()`, and `jazzyk::AbstractKRModuleInterface< TConcreteKRModule >::update()`.

7.20.1.2 `#define JZMODULE_MANIFEST_BEGIN(ModuleClass)`

Value:

```
typedef ModuleClass _TJzModuleClass; \
extern "C" \
{ \
    unsigned int JZMODULE_API_VERSION_ROUTINE_ID() \
    { \
        static std::ios_base::Init initIostreams; \
        return JAZZYK_API_VERSION; \
    } \
    \
    jazzyk::KRModuleInterface* JZMODULE_API_REGISTER_ROUTINE_ID() \
    { \
        jazzyk::AbstractKRModuleInterface<ModuleClass>* pModule = new jazzyk::AbstractKRModuleInterface
```

KR module manifest support macros.

Each KR module has to provide a manifest of its query and update custom operation handlers. This is to be done using macros of the JZMODULE_MANIFEST family.

Example:

```
JZMODULE_MANIFEST_BEGIN(CMyModule) (p. 246) REGISTER_-
QUERY("queryOne", custom_query1) REGISTER_QUERY("queryTwo", custom_query2)
REGISTER_UPDATE("updateOne", custom_update1) REGISTER_UPDATE("updateTwo",
custom_update2) JZMODULE_MANIFEST_END Defines two primitive __JZDLL API
functions which should be invoked from the server side
```

```
unsigned int version();
```

- returns the __JZDLL API version. It must be invoked as the first routine after loading __JZDLL. Using two incompatible components (server+__JZDLL client) might lead to unexpected run-time errors!

```
jazzyk::KRModuleInterface (p. 199)* load_kr_module();
```

- returns an instance of the __JZDLL interface class. It implements the full functionality of the KR module.

Definition at line 297 of file jazzyk.hpp.

7.20.1.3 `#define JZMODULE_MANIFEST_END`

Value:


```
return pModule; \
    } \
}
```

Encloses the KR module manifest declaration.

Definition at line 312 of file jazzyk.hpp.

7.20.1.4 `#define REGISTER_QUERY(szMethodID, ptrMethod) pModule → registerQuery(szMethodID, &_TJzModuleClass::ptrMethod);`

Query/Update registration macros.

- first argument is the operation identifier.
- second argument is the member method identifier (implementation of the operation).

The same method might be registered with several operations.

Definition at line 325 of file jazzyk.hpp.

7.20.1.5 `#define REGISTER_UPDATE(szMethodID, ptrMethod) pModule → registerUpdate(szMethodID, &_TJzModuleClass::ptrMethod);`

Definition at line 326 of file jazzyk.hpp.

7.21 jazzyk_common.hpp File Reference

Namespaces

- namespace **jazzyk**

Classes

- class **jazzyk::KRModuleInterface**
Abstract base class for defining knowledge representation modules.

Defines

- #define **__JZDLL_EXPORT**
GNU/Linux/Win32 portability for dynamically linked shared libraries.
- #define **__JZDLL_CALL**
- #define **JZMODULE_API_VERSION_ROUTINE_ID** version
Identifier of the API routine returning the API version.
- #define **JZMODULE_API_VERSION_ROUTINE** "version"
- #define **JZMODULE_API_REGISTER_ROUTINE_ID** load_kr_module
Identifier of the KR module registration API routine.
- #define **JZMODULE_API_REGISTER_ROUTINE** "load_kr_module"

Typedefs

- typedef unsigned int(* **jazzyk::TJzModuleApiVersionRoutine**)()
Signature of the version API routine.
- typedef **jazzyk::KRModuleInterface** *(* **jazzyk::TJzModuleApiRegisterRoutine**)()
Signature of the KR module registration routine.

Enumerations

- enum **jazzyk::EKRModuleError** { **jazzyk::OK**, **jazzyk::FAIL**, **jazzyk::INVALID_OPERATION** }
Error codes returned from the KR module to the main process.

Variables

- const unsigned int **JAZZYK_API_VERSION** = 4
Knowledge representation module API version.

7.21.1 Define Documentation

7.21.1.1 #define __JZDLL_CALL

Definition at line 50 of file jazzyk_common.hpp.

7.21.1.2 #define __JZDLL_EXPORT

GNU/Linux/Win32 portability for dynamically linked shared libraries.

Definition at line 49 of file jazzyk_common.hpp.

7.21.1.3 #define JZMODULE_API_REGISTER_ROUTINE "load_kr_module"

Definition at line 60 of file jazzyk_common.hpp.

Referenced by CSingleModule::loadDll().

7.21.1.4 #define JZMODULE_API_REGISTER_ROUTINE_ID load_kr_module

Identifier of the KR module registration API routine.

Definition at line 59 of file jazzyk_common.hpp.

7.21.1.5 #define JZMODULE_API_VERSION_ROUTINE "version"

Definition at line 56 of file jazzyk_common.hpp.

Referenced by CSingleModule::loadDll().

7.21.1.6 #define JZMODULE_API_VERSION_ROUTINE_ID version

Identifier of the API routine returning the API version.

Definition at line 55 of file jazzyk_common.hpp.

7.21.2 Variable Documentation

7.21.2.1 const unsigned int JAZZYK_API_VERSION = 4

Knowledge representation module API version.

Used to check whether both sides of the interface use the same interface. Never use not-matching components!!!

Definition at line 67 of file jazzyk_common.hpp.

Referenced by CSingleModule::loadDll(), and CJazzykApp::version().

7.22 JazzykApp.cpp File Reference

```
#include "JazzykApp.hpp"
#include "jazzyk_common.hpp"
#include "Program.hpp"
#include "Compiler.hpp"
#include "Interpreter.hpp"
#include "PrettyPrinter.hpp"
#include "ModuleManager.hpp"
#include "Error.hpp"
#include <iostream>
#include <sstream>
#include <fstream>
#include <boost/program_options.hpp>
#include <boost/interprocess/exceptions.hpp>
```

Defines

- `#define MOTTO "* In varietate concordia *"`

Variables

- `int yydebug`

7.22.1 Define Documentation

7.22.1.1 `#define MOTTO "* In varietate concordia *"`

Definition at line 60 of file JazzykApp.cpp.

Referenced by `CJazzykApp::processCmdLine()`.

7.22.2 Variable Documentation

7.22.2.1 `int yydebug`

Definition at line 57 of file JazzykApp.cpp.

7.23 JazzykApp.hpp File Reference

```
#include <vector>
#include <string>
#include <sstream>
#include <istream>
```

Classes

- class **CJazzykApp**
Main Jazzyk application class.
- struct **CJazzykApp::SSettings**
Settings wrapper.

Typedefs

- typedef **CJazzykApp JazzykApp**
Jazzyk application singleton.

7.23.1 Typedef Documentation

7.23.1.1 typedef CJazzykApp JazzykApp

Jazzyk application singleton.

Definition at line 164 of file JazzykApp.hpp.

7.24 Lexer.l++ File Reference

```
#include "ParserAux.hpp"
#include "Parser.h++"
#include <string>
```

Defines

- #define **This** static_cast<CJazzykLexer*>(this)
- #define **ThisYylval** static_cast<yy::JazzykParser::semantic_type*>(static_cast<CJazzykLexer*>(this) → m_pyylval)
- #define **ThisYylloc** static_cast<yy::JazzykParser::location_type*>(static_cast<CJazzykLexer*>(this) → m_pyylloc)
- #define **YY_USER_ACTION**

Functions

- ThisYylloc step ()

7.24.1 Define Documentation

7.24.1.1 #define **This** static_cast<CJazzykLexer*>(this)

7.24.1.2 #define **ThisYylloc** static_cast<yy::JazzykParser::location_type*>(static_cast<CJazzykLexer*>(this) → m_pyylloc)

7.24.1.3 #define **ThisYylval** static_cast<yy::JazzykParser::semantic_type*>(static_cast<CJazzykLexer*>(this) → m_pyylval)

7.24.1.4 #define **YY_USER_ACTION**

Value:

```
{ \
    ThisYylloc->columns(yyleng); \
    CLocationTracker::Instance().set(*(ThisYylloc)); \
}
```

7.24.2 Function Documentation

7.24.2.1 ThisYylloc step ()

7.25 LocationTracker.hpp File Reference

```
#include "location.hh"
```

Classes

- class **CLocationTracker**

Small singleton for communication between the Bison location and CExpressions.

7.26 Logging.hpp File Reference

Defines

- `#define LOG(Msg)`

7.26.1 Define Documentation

7.26.1.1 `#define LOG(Msg)`

In this header we provide a very simple support for logging facility. In the case of a need, this should be enhanced in the future.

In order to switch logging facility on, compile with `-DLOGGING_ON`

Definition at line 45 of file `Logging.hpp`.

Referenced by `CModuleManager::communicate()`, `CModuleManager::extractResponse()`, `CSingleModule::handleModuleError()`, `CModuleManager::load()`, `CSingleModule::loadDll()`, `CSingleModule::processRequest()`, `CSingleModule::run()`, `CModuleManager::unload()`, and `CSingleModule::writeResponse()`.

7.27 main.cpp File Reference

```
#include "JazzykApp.hpp"
```

Functions

- int **main** (int argc, char *argv[])

7.27.1 Function Documentation

7.27.1.1 int main (int *argc*, char * *argv*[])

Definition at line 42 of file main.cpp.

References CJazzykApp::Instance(), and CJazzykApp::run().

7.28 ModuleCommData.hpp File Reference

```
#include <boost/shared_ptr.hpp>
#include <boost/interprocess/allocators/allocator.hpp>
#include <boost/interprocess/managed_shared_memory.hpp>
#include <boost/interprocess/sync/named_condition.hpp>
#include <boost/interprocess/sync/named_mutex.hpp>
#include <boost/interprocess/sync/named_semaphore.hpp>
#include <boost/interprocess/sync/scoped_lock.hpp>
#include <boost/interprocess/containers/string.hpp>
#include <boost/interprocess/containers/map.hpp>
#include <boost/interprocess/containers/vector.hpp>
#include <boost/interprocess/detail/utilities.hpp>
#include <functional>
```

Classes

- struct **SIPCStringCompare**
String comparison operator.
- struct **SModuleRequest**
Plug-in request structure.
- struct **SModuleResponse**
Plug-in response structure.
- class **CIPCPtrDeleter< T >**
A functor that destroys the shared memory object of an associated smart pointer.

Defines

- `#define g_JZAPI_QUERY "query"`
Constant indicating a query/update interface entry type in the GETINTERFACE request.
- `#define g_JZAPI_UPDATE "update"`

Typedefs

- `typedef boost::shared_ptr< boost::interprocess::managed_shared_memory > TPtrIpcManagedSharedMemory`
Type definitions for synchronisation mechanisms for the Ipc interaction.
- `typedef boost::shared_ptr< boost::interprocess::named_mutex > TPtrIpcMutex`
- `typedef boost::shared_ptr< boost::interprocess::named_condition > TPtrIpcCondition`

- `typedef boost::shared_ptr< boost::interprocess::named_semaphore > TPtrIpcSemaphore`
- `typedef boost::interprocess::allocator< char, boost::interprocess::managed_shared_memory::segment_manager > TIpcCharAllocator`
Type for character allocator for strings in the shared memory.
- `typedef boost::shared_ptr< TIpcCharAllocator > TPtrIpcCharAllocator`
Type definition for a smart pointer (in regular memory) to character allocator.
- `typedef boost::interprocess::basic_string< char, std::char_traits< char >, TIpcCharAllocator > TString`
Type for string allocated in the shared memory.
- `typedef std::pair< TString, TString > TIpSubstValueType`
Type for pair of strings allocated in the shared memory.
- `typedef boost::interprocess::allocator< TIpSubstValueType, boost::interprocess::managed_shared_memory::segment_manager > TIpSubstAllocator`
Type for allocator of pair of strings in the shared memory.
- `typedef boost::shared_ptr< TIpSubstAllocator > TPtrIpcSubstAllocator`
Type definition for smart pointer to string-pair allocator (in the regular memory).
- `typedef boost::interprocess::map< TString, TString, std::less< TString >, TIpSubstAllocator > TIpSubst`
Map of string pairs used in the interprocess communication.
- `typedef CIpcPtrDeleter< SModuleRequest > TSMModuleRequestDeleter`
Type definitions for Boost.Interprocess smart pointer deleters of SModuleRequest/SModuleResponse.
- `typedef CIpcPtrDeleter< SModuleResponse > TSMModuleResponseDeleter`
- `typedef boost::shared_ptr< TSMModuleRequestDeleter > TPtrSModuleRequestDeleter`
Type definitions for smart pointers (in regular memor) to request/response deleters.
- `typedef boost::shared_ptr< TSMModuleResponseDeleter > TPtrSModuleResponseDeleter`
- `typedef boost::shared_ptr< SModuleRequest > TPtrSModuleRequest`
Type definitions for smart pointer for SModuleRequest (p.207) and SModuleResponse (p.210).
- `typedef boost::shared_ptr< SModuleResponse > TPtrSModuleResponse`

Enumerations

- `enum EModuleRequestType {`
`NOP, QUERY, UPDATE, CYCLE,`
`INITIALIZE, FINALIZE, GETINTERFACE }`

Type of the request data to be communicated using the intrprocess shared memory.

- enum **EModuleErrCode** { **OK**, **FAIL** }

Module operation response error code.

7.28.1 Define Documentation

7.28.1.1 #define **g_JZAPI_QUERY** "query"

Constant indicating a query/update interface entry type in the GETINTERFACE request.

Definition at line 150 of file ModuleCommData.hpp.

Referenced by `CSingleModule::convert_operations()`, and `CModuleManager::load()`.

7.28.1.2 #define **g_JZAPI_UPDATE** "update"

Definition at line 151 of file ModuleCommData.hpp.

Referenced by `CSingleModule::convert_operations()`, and `CModuleManager::load()`.

7.28.2 Typedef Documentation

7.28.2.1 typedef **boost::interprocess::allocator< char, boost::interprocess::managed_shared_memory::segment_manager > TIpCharAllocator**

Type for character allocator for strings in the shared memory.

Definition at line 75 of file ModuleCommData.hpp.

7.28.2.2 typedef **boost::interprocess::basic_string< char, std::char_traits<char>, TIpCharAllocator > TIpString**

Type for string allocated in the shared memory.

Definition at line 85 of file ModuleCommData.hpp.

7.28.2.3 typedef **boost::interprocess::map< TIpString, TIpString, std::less<TIpString>, TIpSubstAllocator > TIpSubst**

Map of string pairs used in the interprocess communication.

The map is to be stored in the shared memory, moreover, it should place all its members into the shared memory as well. Therefore we need to specialize its allocator.

Definition at line 113 of file ModuleCommData.hpp.

7.28.2.4 `typedef boost::interprocess::allocator< TIpSubstValueType,
boost::interprocess::managed_shared_memory::segment_manager >
TIpSubstAllocator`

Type for allocator of pair of strings in the shared memory.

Definition at line 97 of file ModuleCommData.hpp.

7.28.2.5 `typedef std::pair< TIpString, TIpString > TIpSubstValueType`

Type for pair of strings allocated in the shared memory.

To be used in the map allocated in the shared memory.

Definition at line 91 of file ModuleCommData.hpp.

7.28.2.6 `typedef boost::shared_ptr<TIpCharAllocator> TPtrIpCharAllocator`

Type definition for a smart pointer (in regular memory) to character allocator.

Definition at line 78 of file ModuleCommData.hpp.

7.28.2.7 `typedef boost::shared_ptr<boost::interprocess::named_condition>
TPtrIpCondition`

Definition at line 56 of file ModuleCommData.hpp.

7.28.2.8 `typedef boost::shared_ptr<boost::interprocess::managed_shared_
memory> TPtrIpManagedSharedMemory`

Type definitions for synchronisation mechanisms for the Ipc interaction.

Definition at line 54 of file ModuleCommData.hpp.

7.28.2.9 `typedef boost::shared_ptr<boost::interprocess::named_mutex>
TPtrIpMutex`

Definition at line 55 of file ModuleCommData.hpp.

7.28.2.10 `typedef boost::shared_ptr<boost::interprocess::named_semaphore>
TPtrIpSemaphore`

Definition at line 57 of file ModuleCommData.hpp.

7.28.2.11 `typedef boost::shared_ptr<TIpSubstAllocator> TPtrIpSubstAllocator`

Type definition for smart pointer to string-pair allocator (in the regular memory).

Definition at line 100 of file ModuleCommData.hpp.

7.28.2.12 `typedef boost::shared_ptr<SModuleRequest> TPtrSModuleRequest`

Type definitions for smart pointer for `SModuleRequest` (p.207) and `SModuleResponse` (p.210).

Definition at line 313 of file `ModuleCommData.hpp`.

7.28.2.13 `typedef boost::shared_ptr<TSMModuleRequestDeleter> TPtrSModuleRequestDeleter`

Type definitions for smart pointers (in regular memor) to request/response deleters.

Definition at line 309 of file `ModuleCommData.hpp`.

7.28.2.14 `typedef boost::shared_ptr<SModuleResponse> TPtrSModuleResponse`

Definition at line 314 of file `ModuleCommData.hpp`.

7.28.2.15 `typedef boost::shared_ptr<TSMModuleResponseDeleter> TPtrSModuleResponseDeleter`

Definition at line 310 of file `ModuleCommData.hpp`.

7.28.2.16 `typedef CIPCPtrDeleter<SModuleRequest> TSMModuleRequestDeleter`

Type definitions for Boost.Interprocess smart pointer deleters of `SModuleRequest`/`SModuleResponse`.

Definition at line 305 of file `ModuleCommData.hpp`.

7.28.2.17 `typedef CIPCPtrDeleter<SModuleResponse> TSMModuleResponseDeleter`

Definition at line 306 of file `ModuleCommData.hpp`.

7.28.3 Enumeration Type Documentation

7.28.3.1 `enum EModuleErrCode`

Module operation response error code.

Enumerator:

OK

FAIL

Definition at line 143 of file `ModuleCommData.hpp`.

7.28.3.2 `enum EModuleRequestType`

Type of the request data to be communicated using the intrprocess shared memory.

Ipc communication data structures. These structures are internally exchanged between the main process and the plug-in subprocesses.

The communicated data structures are allocated in the memory segment shared between the main process and each plug-in subprocess.

The request structure is placed in the shared memory when the main process is about to wake the corresponding plug-in process to process it.

After the request is processed by the plug-in process, it places a corresponding response data structure into the shared memory and indicates the main process that it finished the processing.

Enumerator:

NOP

QUERY

UPDATE

CYCLE

INITIALIZE

FINALIZE

GETINTERFACE

Definition at line 131 of file ModuleCommData.hpp.

7.29 ModuleDeclaration.cpp File Reference

```
#include "ModuleDeclaration.hpp"  
#include "LocationTracker.hpp"  
#include "Error.hpp"  
#include <sstream>
```


7.30 ModuleDeclaration.hpp File Reference

```
#include "ExpressionsFwd.hpp"
#include "ModuleNotification.hpp"
#include "location.hh"
```

Classes

- class **CModuleDeclaration**
Module declaration and link holder.

7.31 ModuleManager.cpp File Reference

```
#include "ModuleManager.hpp"
#include "Error.hpp"
#include "SingleModule.hpp"
#include "Identifier.hpp"
#include "CodeBlock.hpp"
#include "ModuleDeclaration.hpp"
#include "ShmServerMgr.hpp"
#include <string>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include "Logging.hpp"
```

7.32 ModuleManager.hpp File Reference

```
#include "IdentifierMap.hpp"  
#include "ExpressionsFwd.hpp"  
#include "ModuleCommData.hpp"  
#include <sys/types.h>  
#include <string>  
#include <vector>
```

Classes

- class **CModuleManager**
External plug-ins/modules manager.

7.33 ModuleNotification.hpp File Reference

```
#include "ExpressionsFwd.hpp"  
#include "Identifier.hpp"  
#include "CodeBlock.hpp"  
#include "LocationTracker.hpp"
```

Classes

- class **CModuleNotification**
Wrapper class for the module notification declaration.

7.34 NotQueryNode.hpp File Reference

```
#include "QueryNode.hpp"
```

Classes

- class **CNotQueryNode**
Query node for NOT logical modifier of a query expression.

7.35 OrQueryNode.hpp File Reference

```
#include "QueryNode.hpp"
```

Classes

- class **COrQueryNode**
Query node for OR logical connector of queries.

7.36 Parser.y++ File Reference

```
#include "ParserAux.hpp"
```

Defines

- `#define yylex pLexer → yylex`

Functions

- `module_declaration TPtrIdentifier ($5)`
- `module_notification TPtrCodeBlock ($5)`
- `ms_transformer_chain TPtrTransformer ($3)`
- `query TPtrIdentifiers ($3)`
- `push_back (TPtrIdentifier($1))`

Variables

- `statement __pad0__`
- `module_declaration __pad1__`
- `module_notification __pad2__`
- `ms_transformer __pad3__`
- `ms_transformer_chain __pad4__`
- `ms_transformer_set __pad5__`
- `ms_transformer_schain __pad6__`
- `ms_transformer_single __pad7__`
- `rule __pad8__`
- `query_node __pad9__`
- `query __pad10__`
- `update __pad11__`
- `free_variables_decl __pad12__`
- `variables __pad13__`

7.36.1 Define Documentation

7.36.1.1 `#define yylex pLexer → yylex`

7.36.2 Function Documentation

7.36.2.1 `push_back (TPtrIdentifier($1))`

7.36.2.2 `module_notification TPtrCodeBlock ($ 5)`

7.36.2.3 `module_declaration TPtrIdentifier ($ 5)`

Type Constraints

7.36.2.4 `query TPtrIdentifiers ($ 3)`

7.36.2.5 `ms_transformer_chain TPtrTransformer ($ 3)`

Type Constraints

7.36.3 Variable Documentation

7.36.3.1 `statement __pad0__`

Definition at line 8124 of file Parser.y++.

7.36.3.2 `query __pad10__`

Definition at line 8283 of file Parser.y++.

7.36.3.3 `update __pad11__`

Definition at line 8301 of file Parser.y++.

7.36.3.4 `free_variables_decl __pad12__`

Definition at line 8315 of file Parser.y++.

7.36.3.5 `variables __pad13__`

Definition at line 8329 of file Parser.y++.

7.36.3.6 `module_declaration __pad1__`

Definition at line 8138 of file Parser.y++.

7.36.3.7 `module_notification __pad2__`

Definition at line 8146 of file Parser.y++.

7.36.3.8 `ms_transformer __pad3__`

Definition at line 8169 of file Parser.y++.

7.36.3.9 `ms_transformer_chain __pad4__`

Definition at line 8183 of file Parser.y++.

7.36.3.10 `ms_transformer_set __pad5 __`

Definition at line 8196 of file Parser.y++.

7.36.3.11 `ms_transformer_schain __pad6 __`

Definition at line 8210 of file Parser.y++.

7.36.3.12 `ms_transformer_single __pad7 __`

Definition at line 8220 of file Parser.y++.

7.36.3.13 `rule __pad8 __`

Definition at line 8242 of file Parser.y++.

7.36.3.14 `query_node __pad9 __`

Definition at line 8257 of file Parser.y++.

7.37 ParserAux.cpp File Reference

```
#include "ParserAux.hpp"  
#include "Error.hpp"  
#include "Parser.h++"  
#include <sstream>
```

7.38 ParserAux.hpp File Reference

```
#include <string>
#include <boost/shared_ptr.hpp>
#include <istream>
#include <FlexLexer.h>
#include "LocationTracker.hpp"
#include "CodeBlock.hpp"
#include "Identifier.hpp"
#include "ModuleDeclaration.hpp"
#include "ModuleNotification.hpp"
#include "Program.hpp"
#include "Expression.hpp"
#include "Transformer.hpp"
#include "QueryNode.hpp"
#include "TransformerChain.hpp"
#include "TransformerSet.hpp"
#include "Rule.hpp"
#include "ElseRule.hpp"
#include "TrivialUpdate.hpp"
#include "ExplicitUpdate.hpp"
#include "AndQueryNode.hpp"
#include "OrQueryNode.hpp"
#include "NotQueryNode.hpp"
#include "TrivialQuery.hpp"
#include "ExplicitQuery.hpp"
```

Classes

- class **CJazykLexer**

Specialisation of the standard Flex C++ `yyFlexLexer` class.

Typedefs

- typedef `boost::shared_ptr< CJazykLexer >` **TPtrLexer**

Pointer to the Lexer object.

7.38.1 Typedef Documentation

7.38.1.1 `typedef boost::shared_ptr<CJazzykLexer> TPtrLexer`

Pointer to the Lexer object.

Application creates the Lexer instance and passes it to the parser which then uses it for scanning the input.

Definition at line 153 of file ParserAux.hpp.

7.39 PrettyPrinter.cpp File Reference

```
#include "PrettyPrinter.hpp"  
#include "ParserAux.hpp"
```

7.40 PrettyPrinter.hpp File Reference

```
#include "ExpressionsFwd.hpp"
#include "Traverser.hpp"
#include <iostream>
#include <string>
```

Classes

- class **CPrettyPrinter**
Jazzyk program rretty printig service class.

7.41 Program.cpp File Reference

```
#include "Program.hpp"  
#include "IdentifierMap.hpp"  
#include "ModuleDeclaration.hpp"  
#include "ModuleNotification.hpp"  
#include "TransformerSet.hpp"  
#include "Error.hpp"  
#include "ModuleManager.hpp"  
#include <sstream>
```

7.42 Program.hpp File Reference

```
#include "ExpressionsFwd.hpp"
#include "ModuleDeclaration.hpp"
```

Classes

- class **CProgram**
The Jazzyk program backend structure.

Typedefs

- typedef boost::shared_ptr< **CProgram** > **TPtrProgram**
*Smart pointer type definition to **CProgram** (p. 125).*

7.42.1 Typedef Documentation

7.42.1.1 typedef boost::shared_ptr<CProgram> TPtrProgram

Smart pointer type definition to **CProgram** (p. 125).

Definition at line 133 of file Program.hpp.

7.43 QueryNode.hpp File Reference

```
#include "Expression.hpp"
```

```
#include "Context.hpp"
```

Classes

- class **CQueryNode**
Query node abstract base class.

7.44 Rule.hpp File Reference

```
#include "ExpressionsFwd.hpp"  
#include "Transformer.hpp"
```

Classes

- class **CRule**
If-then rule mental state transformer.

7.45 ShmClientMgr.cpp File Reference

```
#include "ShmClientMgr.hpp"
```

7.46 ShmClientMgr.hpp File Reference

```
#include "ShmemManager.hpp"
```

Classes

- class **CShmClientMgr**
Client side interprocess communication manager.

7.47 ShmemManager.cpp File Reference

```
#include <sstream>
#include <string>
#include <iostream>
#include "ShmemManager.hpp"
#include "boost/interprocess/sync/named_mutex.hpp"
#include "boost/interprocess/sync/scoped_lock.hpp"
```

Functions

- static const std::string **g_szRequestConditionSuffix** ("reqcond")
Constants for named condition type name suffixes.
- static const std::string **g_szResponseConditionSuffix** ("respond")
- static const std::string **g_szReadySemaphoreSuffix** ("readysemaphore")
- static const std::string **g_szMemorySuffix** ("memory")
- static const std::string **g_szMutexSuffix** ("mutex")
- static const std::string **g_szRequestSuffix** ("request")
Suffix of the shared request/response instances name.
- static const std::string **g_szResponseSuffix** ("response")

7.47.1 Function Documentation

7.47.1.1 static const std::string g_szMemorySuffix ("memory") [static]

Referenced by CShmemManager::getMemoryID().

7.47.1.2 static const std::string g_szMutexSuffix ("mutex") [static]

Referenced by CShmemManager::getMutexID().

7.47.1.3 static const std::string g_szReadySemaphoreSuffix ("readysemaphore") [static]

Referenced by CShmemManager::SModuleSync::getReadySemaphoreID().

7.47.1.4 static const std::string g_szRequestConditionSuffix ("reqcond") [static]

Constants for named condition type name suffixes.

Referenced by CShmemManager::SModuleSync::getRequestConditionID().

7.47.1.5 `static const std::string g_szRequestSuffix ("request") [static]`

Suffix of the shared request/response instances name.

Referenced by `CShmemManager::getRequestID()`.

7.47.1.6 `static const std::string g_szResponseConditionSuffix ("respcnd") [static]`

Referenced by `CShmemManager::SModuleSync::getResponseConditionID()`.

7.47.1.7 `static const std::string g_szResponseSuffix ("response") [static]`

Referenced by `CShmemManager::getResponseID()`.

7.48 ShmemManager.hpp File Reference

```
#include <string>
#include <boost/shared_ptr.hpp>
#include "IdentifierMap.hpp"
#include "ModuleCommData.hpp"
```

Classes

- class **CShmemManager**
Service managing the shared memory between the main process and plug-in processes.
- struct **CShmemManager::SModuleSync**
Synchronisation conditions structure.
- struct **ShmemLock**
Specialised shared memory lock.

7.49 ShmServerMgr.cpp File Reference

```
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
#include "boost/date_time/posix_time/posix_time.hpp"
#include <string>
#include <sstream>
#include "Identifier.hpp"
#include "ShmServerMgr.hpp"
#include "Error.hpp"
```

Functions

- static const std::string **g_szSharedMemoryPrefix** ("jazzyk_")
Application shared memory prefix constant.

7.49.1 Function Documentation

7.49.1.1 static const std::string **g_szSharedMemoryPrefix** ("jazzyk_") [static]

Application shared memory prefix constant.

Referenced by CShmServerMgr::initialize().

7.50 ShmServerMgr.hpp File Reference

```
#include "ExpressionsFwd.hpp"  
#include "ShmemManager.hpp"  
#include "ModuleCommData.hpp"
```

Classes

- class **CShmServerMgr**
Server side interprocess communication manager.

7.51 SingleModule.cpp File Reference

```
#include "SingleModule.hpp"
#include "ShmClientMgr.hpp"
#include "Error.hpp"
#include "Logging.hpp"
#include <ltdl.h>
#include <string>
#include <vector>
```

Functions

- static const std::string **cszLibNamePrefix** ("libjz")
- static const std::string **cszLibNameSuffix** ("")

7.51.1 Function Documentation

7.51.1.1 static const std::string cszLibNamePrefix ("libjz") [static]

Referenced by CSingleModule::loadDll().

7.51.1.2 static const std::string cszLibNameSuffix ("") [static]

Referenced by CSingleModule::loadDll().

7.52 SingleModule.hpp File Reference

```
#include "jazzyk_common.hpp"
#include "ExpressionsFwd.hpp"
#include "ModuleCommData.hpp"
#include <ltdl.h>
#include <string>
#include <boost/shared_ptr.hpp>
```

Classes

- class **CSingleModule**
Wrapper for a KR module.

7.53 Transformer.hpp File Reference

```
#include "visitor.hpp"  
#include "Expression.hpp"  
#include "Context.hpp"
```

Classes

- class **CTransformer**
Abstract class for all the various types of mental state transformers.

7.54 TransformerChain.cpp File Reference

```
#include "TransformerChain.hpp"
```

7.55 TransformerChain.hpp File Reference

```
#include "ExpressionsFwd.hpp"
```

```
#include "Transformer.hpp"
```

Classes

- class **CTransformerChain**
Chain of mental state transformers.

7.56 TransformerSet.cpp File Reference

```
#include "TransformerSet.hpp"
```

7.57 TransformerSet.hpp File Reference

```
#include "ExpressionsFwd.hpp"
```

```
#include "Transformer.hpp"
```

Classes

- class **CTransformerSet**
Set of mental state transformers.

7.58 Traverser.cpp File Reference

```
#include <algorithm>
#include "Traverser.hpp"
#include "ParserAux.hpp"
```

7.59 Traverser.hpp File Reference

```
#include "visitor.hpp"  
#include "ExpressionsFwd.hpp"  
#include "Context.hpp"
```

Classes

- class **CTraverser**
Program transformer tree traversal abstract class.

7.60 TrivialQuery.hpp File Reference

```
#include "QueryNode.hpp"
```

Classes

- class **CTrueQuery**
Trivial True query.
- class **CFalseQuery**
Trivial False query.

7.61 TrivialUpdate.hpp File Reference

```
#include "Transformer.hpp"
```

Classes

- class **CNopUpdate**
Empty update class implementation.
- class **CExitUpdate**
Exit command mental state transformer implementation.

7.62 visitor.hpp File Reference

```
#include <stdexcept>
```

Classes

- class `base_visitor`
Visitor pattern library.
- class `visitor< T, R >`
Visitor class template.
- class `visitor_arg< T, A, R >`
Visitor passing an argument class template.
- class `visitable< R >`
Base class for `visitable` (p. 214) classes.
- class `visitable_arg< A, R >`
Version of `visitable` (p. 214) class with argument.

Defines

- `#define DECLARE_VISITABLE()`
- `#define DECLARE_VISITABLE_ARG()`
- `#define DECLARE_VISITABLE()`
- `#define DECLARE_VISITABLE_ARG()`
- `#define DECLARE_VISITABLE()`
Provides the code to be inserted into each visited hierarchy subclass.
- `#define DECLARE_VISITABLE_ARG()`
Provides the code to be inserted into each visited hierarchy subclass for accepting visitors with argument.

7.62.1 Define Documentation

7.62.1.1 `#define DECLARE_VISITABLE()`

Value:

```
public: \  
    virtual TReturnType Accept(base_visitor& guest) \  
    { return AcceptImpl(*this, guest); }
```

Provides the code to be inserted into each visited hierarchy subclass.

Definition at line 208 of file visitor.hpp.

7.62.1.2 #define DECLARE_VISITABLE()

Value:

```
public: \  
    virtual TReturnType Accept(base_visitor& guest) \  
    { return AcceptImpl(*this, guest); }
```

7.62.1.3 #define DECLARE_VISITABLE()

Value:

```
public: \  
    virtual TReturnType Accept(base_visitor& guest) \  
    { return AcceptImpl(*this, guest); }
```

7.62.1.4 #define DECLARE_VISITABLE_ARG()

Value:

```
public: \  
    virtual TReturnType Accept(base_visitor& guest, TArgType argument) \  
    { return AcceptImpl(*this, guest, argument); }
```

Provides the code to be inserted into each visited hierarchy subclass for accepting visitors with argument.

Definition at line 214 of file visitor.hpp.

7.62.1.5 #define DECLARE_VISITABLE_ARG()

Value:

```
public: \  
    virtual TReturnType Accept(base_visitor& guest, TArgType argument) \  
    { return AcceptImpl(*this, guest, argument); }
```

7.62.1.6 #define DECLARE_VISITABLE_ARG()

Value:

```
public: \  
    virtual TReturnType Accept(base_visitor& guest, TArgType argument) \  
    { return AcceptImpl(*this, guest, argument); }
```

Index

- ~AbstractKRModuleInterface
 - jazyk::AbstractKRModuleInterface, 20
- ~CAndQueryNode
 - CAndQueryNode, 28
- ~CCodeBlock
 - CCodeBlock, 31
- ~CCompiler
 - CCompiler, 36
- ~CElseRule
 - CElseRule, 45
- ~CError
 - CError, 49
- ~CExitUpdate
 - CExitUpdate, 52
- ~CExplicitQuery
 - CExplicitQuery, 55
- ~CExplicitUpdate
 - CExplicitUpdate, 59
- ~CExpression
 - CExpression, 63
- ~CFalseQuery
 - CFalseQuery, 65
- ~CIdentifier
 - CIdentifier, 68
- ~CInterpreter
 - CInterpreter, 72
- ~CJazykApp
 - CJazykApp, 81
- ~CJazykLexer
 - CJazykLexer, 89
- ~CLocationTracker
 - CLocationTracker, 92
- ~CModuleDeclaration
 - CModuleDeclaration, 95
- ~CModuleManager
 - CModuleManager, 102
- ~CModuleNotification
 - CModuleNotification, 109
- ~CNopUpdate
 - CNopUpdate, 112
- ~CNotQueryNode
 - CNotQueryNode, 115
- ~COrQueryNode
 - COrQueryNode, 117
- ~CPrettyPrinter
 - CPrettyPrinter, 121
- ~CProgram
 - CProgram, 126
- ~CQueryContext
 - CQueryContext, 132
- ~CQueryNode
 - CQueryNode, 136
- ~CRule
 - CRule, 139
- ~CShmClientMgr
 - CShmClientMgr, 142
- ~CShmServerMgr
 - CShmServerMgr, 161
- ~CShmemManager
 - CShmemManager, 148
- ~CSingleModule
 - CSingleModule, 167
- ~CTransformer
 - CTransformer, 172
- ~CTransformerChain
 - CTransformerChain, 175
- ~CTransformerContext
 - CTransformerContext, 178
- ~CTransformerSet
 - CTransformerSet, 182
- ~CTraverser
 - CTraverser, 187
- ~CTrueQuery
 - CTrueQuery, 197
- ~base_visitor
 - base_visitor, 26
- ~visitable
 - visitable, 215
- ~visitable_arg
 - visitable_arg, 217
- ~visitor
 - visitor, 218
- ~visitor_arg
 - visitor_arg, 221
- __JZDLL_CALL
 - jazyk_common.hpp, 249
- __JZDLL_EXPORT
 - jazyk_common.hpp, 249
- __pad0__
 - Parser.y++, 270

- __pad10__
 - Parser.y++, 270
- __pad11__
 - Parser.y++, 270
- __pad12__
 - Parser.y++, 270
- __pad13__
 - Parser.y++, 270
- __pad1__
 - Parser.y++, 270
- __pad2__
 - Parser.y++, 270
- __pad3__
 - Parser.y++, 270
- __pad4__
 - Parser.y++, 270
- __pad5__
 - Parser.y++, 270
- __pad6__
 - Parser.y++, 271
- __pad7__
 - Parser.y++, 271
- __pad8__
 - Parser.y++, 271
- __pad9__
 - Parser.y++, 271
- AbstractKRModuleInterface
 - jazzyk::AbstractKRModuleInterface, 20
- Accept
 - visitable, 215
 - visitable_arg, 217
- AcceptImpl
 - visitable, 215
 - visitable_arg, 217
- add
 - CProgram, 127
- addFreeVariable
 - SModuleRequest, 208
- addModuleSync
 - CShmServerMgr, 162
- addNotification
 - CModuleDeclaration, 96
- addQueryOperation
 - CModuleDeclaration, 96
- addUpdateOperation
 - CModuleDeclaration, 96
- addVariableSubst
 - SModuleRequest, 208
 - SModuleResponse, 211
- allocateRequest
 - CShmemManager, 150
- allocateResponse
 - CShmemManager, 150
- AndQueryNode.hpp, 223
- append
 - CCodeBlock, 31
- argv
 - CCompiler::SMPInvocationSettings, 42
- base_visitor, 26
 - ~base_visitor, 26
- basic_random
 - Interpreter.cpp, 243
- bBypassMP
 - CJazzykApp::SSettings, 86
- bCheckQuery
 - CJazzykApp::SSettings, 86
- bCompileOnly
 - CJazzykApp::SSettings, 86
- bPreprocessOnly
 - CJazzykApp::SSettings, 85
- bPrettyPrint
 - CJazzykApp::SSettings, 85
- bReadStdIn
 - CJazzykApp::SSettings, 85
- bSetFirst
 - CJazzykApp::SSettings, 86
- CALL_MEMBER_FN
 - jazzyk.hpp, 246
- CAndQueryNode, 27
 - ~CAndQueryNode, 28
 - CAndQueryNode, 28
 - getLeftNode, 28
 - getRightNode, 28
 - m_pLeftNode, 29
 - m_pRightNode, 29
 - operator=, 28
- CCodeBlock, 30
 - ~CCodeBlock, 31
 - append, 31
 - CCodeBlock, 31
 - m_szCode, 32
 - operator const char *, 31
 - operator+=, 31
 - operator=, 31
- CCompiler, 33
 - ~CCompiler, 36
 - CCompiler, 35, 36
 - compile, 37
 - fillIncludePaths, 39
 - getProgram, 37
 - m_pProgram, 39
 - m_vszIncludePaths, 39
 - operator=, 39
 - parse, 37
 - PIPE_READ_END, 35

- PIPE_WRITE_END, 35
- PreAction, 37
- preprocess, 36, 38
- readPipe, 39
- runChildProcess, 38
- runParentProcess, 38
- setIncludePaths, 36
- CCompiler::SMPInvocationSettings, 41
 - argv, 42
 - child_pid, 42
 - child_stderr, 42
 - child_stdin, 42
 - child_stdout, 42
 - cmd, 42
 - configure, 42
 - SMPInvocationSettings, 41
- CElseRule, 44
 - ~CElseRule, 45
 - CElseRule, 45
 - getElseTransformer, 45
 - m_pElseTransformer, 45
 - operator=, 45
- CError, 47
 - ~CError, 49
 - CError, 49
 - EErrorType, 49
 - JZCOMPILER_VALIDITY_INVALID_OPERATION, 49
 - JZCOMPILER_VALIDITY_MODULE_REDEFINED, 49
 - JZCOMPILER_VALIDITY_MODULE_UNDECLARED, 49
 - JZCOMPILER_VALIDITY_TRANSFORMER_REDEFINED, 49
 - JZCOMPILER_VALIDITY_TRANSFORMER_UNDEFINED, 49
 - JZCOMPILER_VARIABLE_UNDECLARED, 49
 - JZINTERPRETER_VARIABLE_REWRITE, 49
 - JZLIBTOOL_ERROR, 49
 - JZMODULE_ERROR, 49
 - JZMPREPROCESSOR_ERROR, 49
 - JZNOERROR, 49
 - JZOTHER_ERROR, 49
 - JZPARSER_PARSE_ERROR, 49
 - JZUNKNOWN_ERROR, 49
 - m_dictErrorMsgs, 51
 - m_errorType, 50
 - m_location, 50
 - m_szErrText, 51
 - m_szReason, 50
 - operator=, 50
 - report, 50
 - resolveErrorType, 50
 - TSrcLocation, 48
 - what, 50
- CExitUpdate, 52
 - ~CExitUpdate, 52
 - CExitUpdate, 52, 53
 - operator=, 53
- CExplicitQuery, 54
 - ~CExplicitQuery, 55
 - CExplicitQuery, 55
 - getCodeBlock, 56
 - getFreeVariables, 56
 - getModuleID, 55
 - getOperation, 56
 - m_pCodeBlock, 57
 - m_pModuleID, 56
 - m_pOperation, 56
 - m_pVariables, 56
 - operator=, 56
- CExplicitUpdate, 58
 - ~CExplicitUpdate, 59
 - CExplicitUpdate, 59
 - getCodeBlock, 60
 - getModuleID, 59
 - getOperation, 60
 - getUsedVariables, 60
 - m_pCodeBlock, 61
 - m_pModuleID, 60
 - m_pOperation, 60
 - m_pVariables, 61
 - operator=, 60
- CExpression, 62
 - ~CExpression, 63
 - CExpression, 63
 - getSrcPosition, 63
 - m_srcLocation, 64
 - operator=, 64
- CFalseQuery, 65
 - ~CFalseQuery, 65
 - CFalseQuery, 65, 66
 - operator=, 66
- child_pid
 - CCompiler::SMPInvocationSettings, 42
- child_stderr
 - CCompiler::SMPInvocationSettings, 42
- child_stdin
 - CCompiler::SMPInvocationSettings, 42
- child_stdout
 - CCompiler::SMPInvocationSettings, 42
- CIdentifier, 67
 - ~CIdentifier, 68
 - CIdentifier, 68

- get, 68
- m_szIdentifier, 69
- operator const char *, 68
- operator const std::string &, 68
- operator=, 68
- CInterpreter, 70
 - ~CInterpreter, 72
 - CInterpreter, 72
 - configure, 72
 - doCycle, 74
 - m_pfRandomGenerator, 75
 - m_pProgram, 74
 - m_settings, 74
 - operator=, 74
 - PostAction, 73, 74
 - run, 72
 - Visit, 73
- CInterpreter::SSettings, 76
 - eTransformerSetPolicy, 77
 - nCycles, 77
 - SSettings, 77
 - TS_FIRST, 76
 - TS_RANDOM, 76
- CIpcPtrDeleter, 78
 - CIpcPtrDeleter, 79
 - m_pSegmentManager, 79
 - operator(), 79
 - TSegmentManager, 79
 - TStoredPtr, 79
- CJazzykApp, 80
 - ~CJazzykApp, 81
 - CJazzykApp, 81
 - error, 82
 - Instance, 81
 - license, 82
 - m_settings, 83
 - operator=, 83
 - processCmdLine, 82
 - read, 83
 - run, 81
 - version, 82
- CJazzykApp::SSettings, 84
 - bBypassMP, 86
 - bCheckQuery, 86
 - bCompileOnly, 86
 - bPreprocessOnly, 85
 - bPrettyPrint, 85
 - bReadStdIn, 85
 - bSetFirst, 86
 - files, 85
 - includePaths, 85
 - modulePaths, 85
 - nCycles, 86
 - nMemSize, 86
 - SSettings, 85
- CJazzykLexer, 88
 - ~CJazzykLexer, 89
 - CJazzykLexer, 89
 - m_pyylloc, 89
 - m_pyylval, 89
 - operator=, 89
 - yylex, 89
- CLocationTracker, 91
 - ~CLocationTracker, 92
 - CLocationTracker, 92
 - get, 92
 - Instance, 92
 - m_srcLocation, 93
 - operator=, 92
 - set, 92
- close
 - CShmemManager::SModuleSync, 156
- closeShmem
 - CShmemManager, 152
- cmd
 - CCompiler::SMPInvocationSettings, 42
- CModuleDeclaration, 94
 - ~CModuleDeclaration, 95
 - addNotification, 96
 - addQueryOperation, 96
 - addUpdateOperation, 96
 - CModuleDeclaration, 95
 - getIdentifier, 96
 - getNotificationCode, 97
 - getSrcPosition, 97
 - getType, 96
 - isValidQueryOperation, 96
 - isValidUpdateOperation, 97
 - m_pCycleCode, 98
 - m_pFinalizeCode, 98
 - m_pIdentifier, 97
 - m_pInitCode, 98
 - m_pType, 97
 - m_queryOperations, 98
 - m_srcLocation, 98
 - m_updateOperations, 98
 - operator=, 97
- CModuleManager, 100
 - ~CModuleManager, 102
 - CModuleManager, 102
 - communicate, 105
 - cycle, 104
 - extractResponse, 105
 - filterVariables, 106
 - finalize, 103
 - initialize, 102
 - Instance, 102
 - killModule, 106

- load, 103, 104
- m_bCheckQuery, 107
- m_mapPIDSubprocesses, 106
- m_vecSearchPaths, 106
- operator=, 106
- query, 103
- unload, 103, 104
- update, 104
- writeRequest, 105
- CModuleNotification, 108
 - ~CModuleNotification, 109
- CModuleNotification, 109
- ENotificationType, 109
- getCode, 110
- getIdentifier, 110
- getSrcPosition, 110
- getType, 110
- m_eType, 110
- m_pCode, 111
- m_pIdentifier, 111
- m_srcLocation, 111
- NOTIFY_CYCLE, 109
- NOTIFY_FINALIZE, 109
- NOTIFY_INITIALIZE, 109
- operator=, 110
- CNopUpdate, 112
 - ~CNopUpdate, 112
- CNopUpdate, 112, 113
- operator=, 113
- CNotQueryNode, 114
 - ~CNotQueryNode, 115
- CNotQueryNode, 115
- getNode, 115
- m_pNegatedNode, 115
- operator=, 115
- CodeBlock.cpp, 224
- CodeBlock.hpp, 225
- communicate
 - CModuleManager, 105
- compile
 - CCompiler, 37
- Compiler.cpp, 226
- Compiler.hpp, 227
- configure
 - CCompiler::SMPInvocationSettings, 42
 - CInterpreter, 72
 - CShmemManager, 148
- Context.cpp, 228
- Context.hpp, 229
- convert_operations
 - CSingleModule, 169
- COrQueryNode, 116
 - ~COrQueryNode, 117
- COrQueryNode, 117
 - getLeftNode, 117
 - getRightNode, 117
 - m_pLeftNode, 118
 - m_pRightNode, 118
 - operator=, 118
- CPrettyPrinter, 119
 - ~CPrettyPrinter, 121
- CPrettyPrinter, 121
- DefaultPostAction, 123
- dump, 121
- indent, 123
- m_level, 124
- m_outStream, 124
- m_pProgram, 123
- operator=, 123
- PreAction, 121–123
- CProgram, 125
 - ~CProgram, 126
- add, 127
- CProgram, 126
- cycle, 128
- getModuleDeclaration, 127
- getTransformer, 127
- isModuleDeclared, 127
- loadModules, 128
- m_moduleDeclarations, 128
- m_pTransformerCollection, 128
- operator=, 128
- retrieveInterfaces, 128
- unloadModules, 128
- CQueryContext, 130
 - ~CQueryContext, 132
- CQueryContext, 132
- CTransformerContext, 179
- EQueryReturnValue, 131
- getResult, 132
- isFalse, 133
- isFalseW, 133
- isTrue, 133
- isTrueW, 133
- m_result, 134
- negate, 134
- operator &=, 134
- operator=, 132
- operator|=, 134
- QFALSE, 132
- QTRUE, 132
- QUNDEFINED, 132
- resetResult, 133
- setResult, 133
- CQueryNode, 136
 - ~CQueryNode, 136
- CQueryNode, 136, 137
- operator=, 137

- create
 - CShmemManager::SModuleSync, 156
- createRequest
 - CShmServerMgr, 163
- createResponse
 - CShmClientMgr, 144
- createShmem
 - CShmemManager, 151
- CRule, 138
 - ~CRule, 139
 - CRule, 139
 - getQuery, 139
 - getThenTransformer, 139
 - m_pQueryList, 140
 - m_pThenTransformer, 140
 - operator=, 139
- CShmClientMgr, 141
 - ~CShmClientMgr, 142
 - createResponse, 144
 - CShmClientMgr, 142
 - CShmemManager, 153
 - finalize, 143
 - initialize, 142
 - Instance, 142
 - m_pModuleSync, 144
 - notifyReady, 143
 - notifyResponse, 143
 - operator=, 144
 - retrieveRequest, 143
 - waitForRequest, 143
- CShmemManager, 145
 - ~CShmemManager, 148
 - allocateRequest, 150
 - allocateResponse, 150
 - closeShmem, 152
 - configure, 148
 - createShmem, 151
 - CShmClientMgr, 153
 - CShmemManager, 148
 - CShmServerMgr, 153
 - deleteShmem, 151
 - finalizeChildProcess, 150
 - finalizeMainProcess, 150
 - getMemoryID, 149
 - getMutex, 149
 - getMutexID, 149
 - getRequestDeleter, 151
 - getRequestID, 152
 - getResponseDeleter, 151
 - getResponseID, 152
 - initializeChildProcess, 149
 - initializeHelpers, 152
 - initializeMainProcess, 149
 - Instance, 149
 - m_pCharAllocator, 153
 - m_pRequestDeleter, 154
 - m_pResponseDeleter, 154
 - m_pShMemory, 153
 - m_pShMutex, 153
 - m_pSubstAllocator, 154
 - m_szPrefix, 154
 - openShmem, 152
 - operator=, 153
 - retrieveRequest, 151
 - retrieveResponse, 151
 - ShmemLock, 153
 - TPtrSModuleSync, 148
- CShmemManager::SModuleSync, 155
 - close, 156
 - create, 156
 - destroy, 156
 - getReadySemaphoreID, 157
 - getRequestConditionID, 156
 - getResponseConditionID, 157
 - m_pShReadySemaphore, 157
 - m_pShReqCondition, 157
 - m_pShRespCondition, 157
 - m_szModuleID, 157
 - open, 156
 - SModuleSync, 156
- CShmServerMgr, 159
 - ~CShmServerMgr, 161
 - addModuleSync, 162
 - createRequest, 163
 - CShmemManager, 153
 - CShmServerMgr, 161
 - finalize, 161
 - freeSyncs, 163
 - initialize, 161
 - Instance, 161
 - m_mapModuleSync, 163
 - notifyRequest, 162
 - operator=, 163
 - retrieveResponse, 163
 - TModuleSyncMap, 160
 - waitForReady, 162
 - waitForResponse, 162
- CSingleModule, 165
 - ~CSingleModule, 167
 - convert_operations, 169
 - CSingleModule, 167
 - extract_names_values, 169
 - extractRequest, 168
 - free_names_values, 169
 - handleModuleError, 170
 - Instance, 167
 - loadDll, 168
 - m_hLibrary, 171

- m_pModule, 171
- m_szModuleID, 170
- m_szModuleType, 170
- m_vecPaths, 170
- operator=, 170
- processRequest, 168
- run, 167
- TPtrModule, 167
- unloadDll, 169
- writeResponse, 168
- cszLibNamePrefix
 - SingleModule.cpp, 288
- cszLibNameSuffix
 - SingleModule.cpp, 288
- CTransformer, 172
 - ~CTransformer, 172
 - CTransformer, 172, 173
 - operator=, 173
- CTransformerChain, 174
 - ~CTransformerChain, 175
 - CTransformerChain, 175
 - getTransformers, 176
 - m_pTransformers, 176
 - operator=, 176
 - push_back, 175
 - push_front, 175
- CTransformerContext, 177
 - ~CTransformerContext, 178
 - CQueryContext, 179
 - CTransformerContext, 178
 - getModuleDeclaration, 178
 - isModuleDeclared, 178
 - m_pProgram, 179
 - operator=, 179
 - unify, 179
- CTransformerSet, 181
 - ~CTransformerSet, 182
 - CTransformerSet, 182
 - getTransformers, 183
 - m_pTransformers, 183
 - operator=, 183
 - push_back, 182
 - push_front, 182
- CTraverser, 184
 - ~CTraverser, 187
 - CTraverser, 187
 - DefaultPostAction, 195
 - DefaultPreAction, 194, 195
 - DefaultRuleMidAction, 195
 - operator=, 196
 - PostAction, 192–194
 - PreAction, 190–192
 - RuleMidAction, 194
 - traverse, 196
 - Visit, 187–190
- CTrueQuery, 197
 - ~CTrueQuery, 197
 - CTrueQuery, 197, 198
 - operator=, 198
- CYCLE
 - ModuleCommData.hpp, 261
- cycle
 - CModuleManager, 104
 - CProgram, 128
 - jazzyk::AbstractKRModule, 16
 - jazzyk::AbstractKRModuleInterface, 21
 - jazzyk::KRModuleInterface, 200
- DECLARE_VISITABLE
 - visitor.hpp, 299, 300
- DECLARE_VISITABLE_ARG
 - visitor.hpp, 300
- DefaultPostAction
 - CPrettyPrinter, 123
 - CTraverser, 195
- DefaultPreAction
 - CTraverser, 194, 195
- DefaultRuleMidAction
 - CTraverser, 195
- deleteShmem
 - CShmemManager, 151
- destroy
 - CShmemManager::SModuleSync, 156
 - jazzyk::AbstractKRModuleInterface, 23
 - jazzyk::KRModuleInterface, 202
- doCycle
 - CInterpreter, 74
- dump
 - CPrettyPrinter, 121
- EErrorType
 - CError, 49
- EKRModuleError
 - jazzyk, 12
- ElseRule.hpp, 230
- EModuleErrCode
 - ModuleCommData.hpp, 260
- EModuleRequestType
 - ModuleCommData.hpp, 260
- ENotificationType
 - CModuleNotification, 109
- EQueryReturnValue
 - CQueryContext, 131
- error
 - CJazzykApp, 82
- Error.cpp, 231
 - operator<<, 231
- Error.hpp, 232

- operator<<, 232
- eTransformerSetPolicy
 - CInterpreter::SSettings, 77
- ExplicitQuery.hpp, 233
- ExplicitUpdate.hpp, 234
- Expression.cpp, 235
- Expression.hpp, 236
 - TSrcLocation, 236
- ExpressionsFwd.hpp, 237
 - TIdentifiers, 237
 - TIdentifierSet, 237
 - TModuleDeclCollection, 238
 - TPtrCodeBlock, 238
 - TPtrExitUpdate, 238
 - TPtrIdentifier, 238
 - TPtrIdentifiers, 238
 - TPtrModuleDeclaration, 238
 - TPtrModuleDeclCollection, 238
 - TPtrModuleNotification, 238
 - TPtrNopUpdate, 238
 - TPtrPlugin, 238
 - TPtrProgram, 238
 - TPtrQueryNode, 239
 - TPtrTransformer, 239
 - TPtrTransformerChain, 239
 - TPtrTransformerContext, 239
 - TPtrTransformers, 239
 - TPtrTransformerSet, 239
 - TPtrVariableSubstitution, 239
 - TPtrVariableValue, 239
 - TSrcLocation, 239
 - TTransformers, 239
 - TVariableSubstitution, 239
 - TVariableValue, 240
 - TVecStrings, 240
 - TVecTransformers, 240
- extract_names_values
 - CSingleModule, 169
- extractRequest
 - CSingleModule, 168
- extractResponse
 - CModuleManager, 105
- FAIL
 - jazzyk, 12
 - ModuleCommData.hpp, 260
- files
 - CJazzykApp::SSettings, 85
- fillIncludePaths
 - CCompiler, 39
- filterVariables
 - CModuleManager, 106
- FINALIZE
 - ModuleCommData.hpp, 261
- finalize
 - CModuleManager, 103
 - CShmClientMgr, 143
 - CShmServerMgr, 161
 - jazzyk::AbstractKRModule, 16
 - jazzyk::AbstractKRModuleInterface, 20
 - jazzyk::KRModuleInterface, 200
- finalizeChildProcess
 - CShmemManager, 150
- finalizeMainProcess
 - CShmemManager, 150
- free_names_values
 - CSingleModule, 169
- free_resource
 - jazzyk::AbstractKRModuleInterface, 22
 - jazzyk::KRModuleInterface, 202
- freeSyncs
 - CShmServerMgr, 163
- g_JZAPI_QUERY
 - ModuleCommData.hpp, 258
- g_JZAPI_UPDATE
 - ModuleCommData.hpp, 258
- g_szMemorySuffix
 - ShmemManager.cpp, 283
- g_szMutexSuffix
 - ShmemManager.cpp, 283
- g_szReadySemaphoreSuffix
 - ShmemManager.cpp, 283
- g_szRequestConditionSuffix
 - ShmemManager.cpp, 283
- g_szRequestSuffix
 - ShmemManager.cpp, 283
- g_szResponseConditionSuffix
 - ShmemManager.cpp, 284
- g_szResponseSuffix
 - ShmemManager.cpp, 284
- g_szSharedMemoryPrefix
 - ShmServerMgr.cpp, 286
- get
 - CIdentifier, 68
 - CLocationTracker, 92
- getCode
 - CModuleNotification, 110
- getCodeBlock
 - CExplicitQuery, 56
 - CExplicitUpdate, 60
- getElseTransformer
 - CElseRule, 45
- getFreeVariables
 - CExplicitQuery, 56
- getIdentifier
 - CModuleDeclaration, 96
 - CModuleNotification, 110

- GETINTERFACE
 - ModuleCommData.hpp, 261
- getLeftNode
 - CAndQueryNode, 28
 - COrQueryNode, 117
- getMemoryID
 - CShmemManager, 149
- getModuleDeclaration
 - CProgram, 127
 - CTransformerContext, 178
- getModuleID
 - CExplicitQuery, 55
 - CExplicitUpdate, 59
- getMutex
 - CShmemManager, 149
- getMutexID
 - CShmemManager, 149
- getNode
 - CNotQueryNode, 115
- getNotificationCode
 - CModuleDeclaration, 97
- getOperation
 - CExplicitQuery, 56
 - CExplicitUpdate, 60
- getProgram
 - CCompiler, 37
- getQuery
 - CRule, 139
- getQueryInterface
 - jazzyk::AbstractKRModuleInterface, 22
 - jazzyk::KRModuleInterface, 200
- getReadySemaphoreID
 - CShmemManager::SModuleSync, 157
- getRequestConditionID
 - CShmemManager::SModuleSync, 156
- getRequestDeleter
 - CShmemManager, 151
- getRequestID
 - CShmemManager, 152
- getResponseConditionID
 - CShmemManager::SModuleSync, 157
- getResponseDeleter
 - CShmemManager, 151
- getResponseID
 - CShmemManager, 152
- getResult
 - CQueryContext, 132
- getRightNode
 - CAndQueryNode, 28
 - COrQueryNode, 117
- getSrcPosition
 - CExpression, 63
 - CModuleDeclaration, 97
 - CModuleNotification, 110
- getThenTransformer
 - CRule, 139
- getTransformer
 - CProgram, 127
- getTransformers
 - CTransformerChain, 176
 - CTransformerSet, 183
- getType
 - CModuleDeclaration, 96
 - CModuleNotification, 110
- getUpdateInterface
 - jazzyk::AbstractKRModuleInterface, 22
 - jazzyk::KRModuleInterface, 201
- getUsedVariables
 - CExplicitUpdate, 60
- handleModuleError
 - CSingleModule, 170
- Identifier.hpp, 241
- IdentifierMap.hpp, 242
- includePaths
 - CJazzykApp::SSettings, 85
- indent
 - CPrettyPrinter, 123
- INITIALIZE
 - ModuleCommData.hpp, 261
- initialize
 - CModuleManager, 102
 - CShmClientMgr, 142
 - CShmServerMgr, 161
 - jazzyk::AbstractKRModule, 15
 - jazzyk::AbstractKRModuleInterface, 20
 - jazzyk::KRModuleInterface, 200
- initializeChildProcess
 - CShmemManager, 149
- initializeHelpers
 - CShmemManager, 152
- initializeMainProcess
 - CShmemManager, 149
- Instance
 - CJazzykApp, 81
 - CLocationTracker, 92
 - CModuleManager, 102
 - CShmClientMgr, 142
 - CShmemManager, 149
 - CShmServerMgr, 161
 - CSingleModule, 167
- Interpreter.cpp, 243
 - basic_random, 243
- Interpreter.hpp, 244
- INVALID_OPERATION
 - jazzyk, 12
- isFalse

- CQueryContext, 133
- isFalseW
 - CQueryContext, 133
- isModuleDeclared
 - CProgram, 127
 - CTransformerContext, 178
- isTrue
 - CQueryContext, 133
- isTrueW
 - CQueryContext, 133
- isValidQueryOperation
 - CModuleDeclaration, 96
- isValidUpdateOperation
 - CModuleDeclaration, 97
- jazzyk, 11
 - EKRModuleError, 12
 - FAIL, 12
 - INVALID_OPERATION, 12
 - OK, 12
 - TJzModuleApiRegisterRoutine, 12
 - TJzModuleApiVersionRoutine, 12
 - TKRVarSubstitution, 12
- jazzyk.hpp, 245
 - CALL_MEMBER_FN, 246
 - JZMODULE_MANIFEST_BEGIN, 246
 - JZMODULE_MANIFEST_END, 246
 - REGISTER_QUERY, 247
 - REGISTER_UPDATE, 247
- jazzyk::AbstractKRModule, 15
 - cycle, 16
 - finalize, 16
 - initialize, 15
- jazzyk::AbstractKRModuleInterface, 17
 - ~AbstractKRModuleInterface, 20
 - AbstractKRModuleInterface, 20
 - cycle, 21
 - destroy, 23
 - finalize, 20
 - free_resource, 22
 - getQueryInterface, 22
 - getUpdateInterface, 22
 - initialize, 20
 - m_concreteKRModule, 24
 - m_QueryOperations, 24
 - m_UpdateOperations, 24
 - map2raw, 23
 - operator delete, 23
 - query, 21
 - raw2map, 24
 - registerQuery, 22
 - registerUpdate, 23
 - TPtrQueryHandler, 19
 - TPtrUpdateHandler, 19
 - TQueryRegister, 19
 - TUpdateRegister, 19
 - update, 21
- jazzyk::KRModuleInterface, 199
 - cycle, 200
 - destroy, 202
 - finalize, 200
 - free_resource, 202
 - getQueryInterface, 200
 - getUpdateInterface, 201
 - initialize, 200
 - operator delete, 202
 - query, 201
 - update, 201
- JAZZYK_API_VERSION
 - jazzyk_common.hpp, 249
- jazzyk_common.hpp, 248
 - __JZDLL_CALL, 249
 - __JZDLL_EXPORT, 249
 - JAZZYK_API_VERSION, 249
 - JZMODULE_API_REGISTER_ROUTINE, 249
 - JZMODULE_API_REGISTER_ROUTINE_ID, 249
 - JZMODULE_API_VERSION_ROUTINE, 249
 - JZMODULE_API_VERSION_ROUTINE_ID, 249
- jazzyk_private, 13
- jazzyk_private::StringCompare, 212
 - operator(), 212
- JazzykApp
 - JazzykApp.hpp, 251
- JazzykApp.cpp, 250
 - MOTTO, 250
 - yydebug, 250
- JazzykApp.hpp, 251
 - JazzykApp, 251
- JZCOMPILER_VALIDITY_INVALID_OPERATION
 - CError, 49
- JZCOMPILER_VALIDITY_MODULE_REDEFINED
 - CError, 49
- JZCOMPILER_VALIDITY_MODULE_UNDECLARED
 - CError, 49
- JZCOMPILER_VALIDITY_TRANSFORMER_REDEFINED
 - CError, 49
- JZCOMPILER_VALIDITY_TRANSFORMER_UNDEFINED
 - CError, 49
- JZCOMPILER_VARIABLE_UNDECLARED

- CError, 49
- JZINTERPRETER_VARIABLE_REWRITE
 - CError, 49
- JZLIBTOOL_ERROR
 - CError, 49
- JZMODULE_API_REGISTER_ROUTINE
 - jazyk_common.hpp, 249
- JZMODULE_API_REGISTER_-
 - ROUTINE_ID
 - jazyk_common.hpp, 249
- JZMODULE_API_VERSION_ROUTINE
 - jazyk_common.hpp, 249
- JZMODULE_API_VERSION_ROUTINE_-
 - ID
 - jazyk_common.hpp, 249
- JZMODULE_ERROR
 - CError, 49
- JZMODULE_MANIFEST_BEGIN
 - jazyk.hpp, 246
- JZMODULE_MANIFEST_END
 - jazyk.hpp, 246
- JZMPREPROCESSOR_EROR
 - CError, 49
- JZNOERROR
 - CError, 49
- JZOTHER_ERROR
 - CError, 49
- JZPARSER_PARSE_ERROR
 - CError, 49
- JZUNKNOWN_ERROR
 - CError, 49
- killModule
 - CModuleManager, 106
- Lexer.l++, 252
 - step, 252
 - This, 252
 - ThisYylloc, 252
 - ThisYylval, 252
 - YY_USER_ACTION, 252
- license
 - CJazykApp, 82
- load
 - CModuleManager, 103, 104
- loadDll
 - CSingleModule, 168
- loadModules
 - CProgram, 128
- LocationTracker.hpp, 253
- LOG
 - Logging.hpp, 254
- Logging.hpp, 254
 - LOG, 254
- m_bCheckQuery
 - CModuleManager, 107
- m_bQueryResult
 - SModuleResponse, 211
- m_charAllocator
 - SModuleRequest, 209
 - SModuleResponse, 211
- m_concreteKRModule
 - jazyk::AbstractKRModuleInterface, 24
- m_dictErrorMsgs
 - CError, 51
- m_eError
 - SModuleResponse, 211
- m_errorType
 - CError, 50
- m_eType
 - CModuleNotification, 110
 - SModuleRequest, 208
- m_hLibrary
 - CSingleModule, 171
- m_level
 - CPrettyPrinter, 124
- m_location
 - CError, 50
- m_mapFreeVariables
 - SModuleRequest, 209
- m_mapModuleSync
 - CShmServerMgr, 163
- m_mapPIDSubprocesses
 - CModuleManager, 106
- m_mapSubstitution
 - SModuleRequest, 209
 - SModuleResponse, 211
- m_moduleDeclarations
 - CProgram, 128
- m_outStream
 - CPrettyPrinter, 124
- m_pCharAllocator
 - CShmemManager, 153
- m_pCode
 - CModuleNotification, 111
- m_pCodeBlock
 - CExplicitQuery, 57
 - CExplicitUpdate, 61
- m_pCycleCode
 - CModuleDeclaration, 98
- m_pElseTransformer
 - CElseRule, 45
- m_pFinalizeCode
 - CModuleDeclaration, 98
- m_pfRandomGenerator
 - CInterpreter, 75
- m_pIdentifier
 - CModuleDeclaration, 97

- CModuleNotification, 111
- m_pInitCode
 - CModuleDeclaration, 98
- m_pLeftNode
 - CAndQueryNode, 29
 - COrQueryNode, 118
- m_pModule
 - CSingleModule, 171
- m_pModuleID
 - CExplicitQuery, 56
 - CExplicitUpdate, 60
- m_pModuleSync
 - CShmClientMgr, 144
- m_pNegatedNode
 - CNotQueryNode, 115
- m_pOperation
 - CExplicitQuery, 56
 - CExplicitUpdate, 60
- m_pProgram
 - CCompiler, 39
 - CInterpreter, 74
 - CPrettyPrinter, 123
 - CTransformerContext, 179
- m_pQueryList
 - CRule, 140
- m_pRequestDeleter
 - CShmemManager, 154
- m_pResponseDeleter
 - CShmemManager, 154
- m_pRightNode
 - CAndQueryNode, 29
 - COrQueryNode, 118
- m_pSegmentManager
 - CipcPtrDeleter, 79
- m_pShMemory
 - CShmemManager, 153
- m_pShMutex
 - CShmemManager, 153
- m_pShReadySemaphore
 - CShmemManager::SModuleSync, 157
- m_pShReqCondition
 - CShmemManager::SModuleSync, 157
- m_pShRespCondition
 - CShmemManager::SModuleSync, 157
- m_pSubstAllocator
 - CShmemManager, 154
- m_pThenTransformer
 - CRule, 140
- m_pTransformerCollection
 - CProgram, 128
- m_pTransformers
 - CTransformerChain, 176
 - CTransformerSet, 183
- m_pType
 - CModuleDeclaration, 97
- m_pVariables
 - CExplicitQuery, 56
 - CExplicitUpdate, 61
- m_pyylloc
 - CJazzykLexer, 89
- m_pyylval
 - CJazzykLexer, 89
- m_QueryOperations
 - jazzyk::AbstractKRModuleInterface, 24
- m_queryOperations
 - CModuleDeclaration, 98
- m_result
 - CQueryContext, 134
- m_settings
 - CInterpreter, 74
 - CJazzykApp, 83
- m_srcLocation
 - CExpression, 64
 - CLocationTracker, 93
 - CModuleDeclaration, 98
 - CModuleNotification, 111
- m_szCode
 - CCodeBlock, 32
- m_szCodeBlock
 - SModuleRequest, 208
- m_szErrText
 - CError, 51
- m_szIdentifier
 - CIdentifier, 69
- m_szModuleID
 - CShmemManager::SModuleSync, 157
 - CSingleModule, 170
- m_szModuleType
 - CSingleModule, 170
- m_szOperation
 - SModuleRequest, 208
- m_szPrefix
 - CShmemManager, 154
- m_szReason
 - CError, 50
- m_UpdateOperations
 - jazzyk::AbstractKRModuleInterface, 24
- m_updateOperations
 - CModuleDeclaration, 98
- m_vecPaths
 - CSingleModule, 170
- m_vecSearchPaths
 - CModuleManager, 106
- m_vsIncludePaths
 - CCompiler, 39
- main
 - main.cpp, 255
 - main.cpp, 255

- main, 255
- map2raw
 - jazzyk::AbstractKRModuleInterface, 23
- ModuleCommData.hpp, 256
 - CYCLE, 261
 - EModuleErrCode, 260
 - EModuleRequestType, 260
 - FAIL, 260
 - FINALIZE, 261
 - g_JZAPI_QUERY, 258
 - g_JZAPI_UPDATE, 258
 - GETINTERFACE, 261
 - INITIALIZE, 261
 - NOP, 261
 - OK, 260
 - QUERY, 261
 - TIPCCharAllocator, 258
 - TIPCString, 258
 - TIPCSubst, 258
 - TIPCSubstAllocator, 258
 - TIPCSubstValueType, 259
 - TPtrIPCCharAllocator, 259
 - TPtrIPCCondition, 259
 - TPtrIPCManagedSharedMemory, 259
 - TPtrIPCMutex, 259
 - TPtrIPCSemaphore, 259
 - TPtrIPCSubstAllocator, 259
 - TPtrSModuleRequest, 259
 - TPtrSModuleRequestDeleter, 260
 - TPtrSModuleResponse, 260
 - TPtrSModuleResponseDeleter, 260
 - TSMModuleRequestDeleter, 260
 - TSMModuleResponseDeleter, 260
 - UPDATE, 261
- ModuleDeclaration.cpp, 262
- ModuleDeclaration.hpp, 263
- ModuleManager.cpp, 264
- ModuleManager.hpp, 265
- ModuleNotification.hpp, 266
- modulePaths
 - CJazzykApp::SSettings, 85
- MOTTO
 - JazzykApp.cpp, 250
- nCycles
 - CInterpreter::SSettings, 77
 - CJazzykApp::SSettings, 86
- negate
 - CQueryContext, 134
- nMemSize
 - CJazzykApp::SSettings, 86
- NOP
 - ModuleCommData.hpp, 261
- NOTIFY_CYCLE
 - CModuleNotification, 109
- NOTIFY_FINALIZE
 - CModuleNotification, 109
- NOTIFY_INITIALIZE
 - CModuleNotification, 109
- notifyReady
 - CShmClientMgr, 143
- notifyRequest
 - CShmServerMgr, 162
- notifyResponse
 - CShmClientMgr, 143
- NotQueryNode.hpp, 267
- OK
 - jazzyk, 12
 - ModuleCommData.hpp, 260
- open
 - CShmemManager::SModuleSync, 156
- openShmem
 - CShmemManager, 152
- operator &=
 - CQueryContext, 134
- operator const char *
 - CCodeBlock, 31
 - CIdentifier, 68
- operator const std::string &
 - CIdentifier, 68
- operator delete
 - jazzyk::AbstractKRModuleInterface, 23
 - jazzyk::KRModuleInterface, 202
- operator <<
 - Error.cpp, 231
 - Error.hpp, 232
- operator()
 - CIPCPtrDeleter, 79
 - jazzyk_private::StringCompare, 212
 - SIdentifierCompare, 204
 - SIdentifierEqual, 205
 - SIPCStringCompare, 206
- operator +=
 - CCodeBlock, 31
- operator =
 - CAndQueryNode, 28
 - CCodeBlock, 31
 - CCompiler, 39
 - CElseRule, 45
 - CError, 50
 - CExitUpdate, 53
 - CExplicitQuery, 56
 - CExplicitUpdate, 60
 - CExpression, 64
 - CFalseQuery, 66
 - CIdentifier, 68
 - CInterpreter, 74

- CJazzykApp, 83
- CJazzykLexer, 89
- CLocationTracker, 92
- CModuleDeclaration, 97
- CModuleManager, 106
- CModuleNotification, 110
- CNoUpdate, 113
- CNotQueryNode, 115
- COrQueryNode, 118
- CPrettyPrinter, 123
- CProgram, 128
- CQueryContext, 132
- CQueryNode, 137
- CRule, 139
- CShmClientMgr, 144
- CShmMemManager, 153
- CShmServerMgr, 163
- CSingleModule, 170
- CTransformer, 173
- CTransformerChain, 176
- CTransformerContext, 179
- CTransformerSet, 183
- CTraverser, 196
- CTrueQuery, 198
- operator |=
 - CQueryContext, 134
- OrQueryNode.hpp, 268
- parse
 - CCompiler, 37
- Parser.y++, 269
 - __pad0 __, 270
 - __pad10 __, 270
 - __pad11 __, 270
 - __pad12 __, 270
 - __pad13 __, 270
 - __pad1 __, 270
 - __pad2 __, 270
 - __pad3 __, 270
 - __pad4 __, 270
 - __pad5 __, 270
 - __pad6 __, 271
 - __pad7 __, 271
 - __pad8 __, 271
 - __pad9 __, 271
- push_back, 269
- TPtrCodeBlock, 269
- TPtrIdentifier, 269
- TPtrIdentifiers, 269
- TPtrTransformer, 270
- yylex, 269
- ParserAux.cpp, 272
- ParserAux.hpp, 273
 - TPtrLexer, 274
- PIPE_READ_END
 - CCompiler, 35
- PIPE_WRITE_END
 - CCompiler, 35
- PostAction
 - CInterpreter, 73, 74
 - CTraverser, 192–194
- PreAction
 - CCompiler, 37
 - CPrettyPrinter, 121–123
 - CTraverser, 190–192
- preprocess
 - CCompiler, 36, 38
- PrettyPrinter.cpp, 275
- PrettyPrinter.hpp, 276
- processCmdLine
 - CJazzykApp, 82
- processRequest
 - CSingleModule, 168
- Program.cpp, 277
- Program.hpp, 278
 - TPtrProgram, 278
- push_back
 - CTransformerChain, 175
 - CTransformerSet, 182
 - Parser.y++, 269
- push_front
 - CTransformerChain, 175
 - CTransformerSet, 182
- QFALSE
 - CQueryContext, 132
- QTRUE
 - CQueryContext, 132
- QUERY
 - ModuleCommData.hpp, 261
- query
 - CModuleManager, 103
 - jazzyk::AbstractKRModuleInterface, 21
 - jazzyk::KRModuleInterface, 201
- QueryNode.hpp, 279
- QUNDEFINED
 - CQueryContext, 132
- raw2map
 - jazzyk::AbstractKRModuleInterface, 24
- read
 - CJazzykApp, 83
- readPipe
 - CCompiler, 39
- REGISTER_QUERY
 - jazzyk.hpp, 247
- REGISTER_UPDATE
 - jazzyk.hpp, 247

- registerQuery
 - jazyk::AbstractKRModuleInterface, 22
- registerUpdate
 - jazyk::AbstractKRModuleInterface, 23
- report
 - CError, 50
- resetResult
 - CQueryContext, 133
- resolveErrorType
 - CError, 50
- retrieveInterfaces
 - CProgram, 128
- retrieveRequest
 - CShmClientMgr, 143
 - CShmemManager, 151
- retrieveResponse
 - CShmemManager, 151
 - CShmServerMgr, 163
- Rule.hpp, 280
- RuleMidAction
 - CTraverser, 194
- run
 - CInterpreter, 72
 - CJazykApp, 81
 - CSingleModule, 167
- runChildProcess
 - CCompiler, 38
- runParentProcess
 - CCompiler, 38
- set
 - CLocationTracker, 92
- setIncludePaths
 - CCompiler, 36
- setResult
 - CQueryContext, 133
- ShmClientMgr.cpp, 281
- ShmClientMgr.hpp, 282
- ShmemLock, 203
 - CShmemManager, 153
 - ShmemLock, 203
- ShmemManager.cpp, 283
 - g_szMemorySuffix, 283
 - g_szMutexSuffix, 283
 - g_szReadySemaphoreSuffix, 283
 - g_szRequestConditionSuffix, 283
 - g_szRequestSuffix, 283
 - g_szResponseConditionSuffix, 284
 - g_szResponseSuffix, 284
- ShmemManager.hpp, 285
- ShmServerMgr.cpp, 286
 - g_szSharedMemoryPrefix, 286
- ShmServerMgr.hpp, 287
- SIdentifierCompare, 204
 - operator(), 204
- SIdentifierEqual, 205
 - operator(), 205
- SingleModule.cpp, 288
 - cszLibNamePrefix, 288
 - cszLibNameSuffix, 288
- SingleModule.hpp, 289
- SIpcStringCompare, 206
 - operator(), 206
- SModuleRequest, 207
 - addFreeVariable, 208
 - addVariableSubst, 208
 - m_charAllocator, 209
 - m_eType, 208
 - m_mapFreeVariables, 209
 - m_mapSubstitution, 209
 - m_szCodeBlock, 208
 - m_szOperation, 208
 - SModuleRequest, 208
- SModuleResponse, 210
 - addVariableSubst, 211
 - m_bQueryResult, 211
 - m_charAllocator, 211
 - m_eError, 211
 - m_mapSubstitution, 211
 - SModuleResponse, 210
- SModuleSync
 - CShmemManager::SModuleSync, 156
- SMPInvocationSettings
 - CCompiler::SMPInvocationSettings, 41
- SSettings
 - CInterpreter::SSettings, 77
 - CJazykApp::SSettings, 85
- step
 - Lexer.l++, 252
- TArgType
 - visitable_arg, 216
 - visitor_arg, 220
- This
 - Lexer.l++, 252
- ThisYlloc
 - Lexer.l++, 252
- ThisYlval
 - Lexer.l++, 252
- TIdentifierMap, 213
- TIdentifiers
 - ExpressionsFwd.hpp, 237
- TIdentifierSet
 - ExpressionsFwd.hpp, 237
- TIpcCharAllocator
 - ModuleCommData.hpp, 258
- TIpcString
 - ModuleCommData.hpp, 258

- TIPCSubst
 - ModuleCommData.hpp, 258
- TIPCSubstAllocator
 - ModuleCommData.hpp, 258
- TIPCSubstValueType
 - ModuleCommData.hpp, 259
- TJzModuleApiRegisterRoutine
 - jazyk, 12
- TJzModuleApiVersionRoutine
 - jazyk, 12
- TKRVarSubstitution
 - jazyk, 12
- TModuleDeclCollection
 - ExpressionsFwd.hpp, 238
- TModuleSyncMap
 - CShmServerMgr, 160
- TPtrCodeBlock
 - ExpressionsFwd.hpp, 238
 - Parser.y++, 269
- TPtrExitUpdate
 - ExpressionsFwd.hpp, 238
- TPtrIdentifier
 - ExpressionsFwd.hpp, 238
 - Parser.y++, 269
- TPtrIdentifiers
 - ExpressionsFwd.hpp, 238
 - Parser.y++, 269
- TPtrIpcCharAllocator
 - ModuleCommData.hpp, 259
- TPtrIpcCondition
 - ModuleCommData.hpp, 259
- TPtrIpcManagedSharedMemory
 - ModuleCommData.hpp, 259
- TPtrIpcMutex
 - ModuleCommData.hpp, 259
- TPtrIpcSemaphore
 - ModuleCommData.hpp, 259
- TPtrIpcSubstAllocator
 - ModuleCommData.hpp, 259
- TPtrLexer
 - ParserAux.hpp, 274
- TPtrModule
 - CSingleModule, 167
- TPtrModuleDeclaration
 - ExpressionsFwd.hpp, 238
- TPtrModuleDeclCollection
 - ExpressionsFwd.hpp, 238
- TPtrModuleNotification
 - ExpressionsFwd.hpp, 238
- TPtrNopUpdate
 - ExpressionsFwd.hpp, 238
- TPtrPlugin
 - ExpressionsFwd.hpp, 238
- TPtrProgram
 - ExpressionsFwd.hpp, 238
 - Program.hpp, 278
- TPtrQueryHandler
 - jazyk::AbstractKRModuleInterface, 19
- TPtrQueryNode
 - ExpressionsFwd.hpp, 239
- TPtrSModuleRequest
 - ModuleCommData.hpp, 259
- TPtrSModuleRequestDeleter
 - ModuleCommData.hpp, 260
- TPtrSModuleResponse
 - ModuleCommData.hpp, 260
- TPtrSModuleResponseDeleter
 - ModuleCommData.hpp, 260
- TPtrSModuleSync
 - CShmemManager, 148
- TPtrTransformer
 - ExpressionsFwd.hpp, 239
 - Parser.y++, 270
- TPtrTransformerChain
 - ExpressionsFwd.hpp, 239
- TPtrTransformerContext
 - ExpressionsFwd.hpp, 239
- TPtrTransformers
 - ExpressionsFwd.hpp, 239
- TPtrTransformerSet
 - ExpressionsFwd.hpp, 239
- TPtrUpdateHandler
 - jazyk::AbstractKRModuleInterface, 19
- TPtrVariableSubstitution
 - ExpressionsFwd.hpp, 239
- TPtrVariableValue
 - ExpressionsFwd.hpp, 239
- TQueryRegister
 - jazyk::AbstractKRModuleInterface, 19
- Transformer.hpp, 290
- TransformerChain.cpp, 291
- TransformerChain.hpp, 292
- TransformerSet.cpp, 293
- TransformerSet.hpp, 294
- traverse
 - CTraverser, 196
- Traverser.cpp, 295
- Traverser.hpp, 296
- TReturnType
 - visitable, 214
 - visitable_arg, 216
 - visitor, 218
 - visitor_arg, 220
- TrivialQuery.hpp, 297
- TrivialUpdate.hpp, 298
- TS_FIRST
 - CInterpreter::SSettings, 76
- TS_RANDOM

- CInterpreter::SSettings, 76
- TSegmentManager
 - CIpcPtrDeleter, 79
- TSMModuleRequestDeleter
 - ModuleCommData.hpp, 260
- TSMModuleResponseDeleter
 - ModuleCommData.hpp, 260
- TSrcLocation
 - CError, 48
 - Expression.hpp, 236
 - ExpressionsFwd.hpp, 239
- TStoredPtr
 - CIpcPtrDeleter, 79
- TTransformers
 - ExpressionsFwd.hpp, 239
- TUpdateRegister
 - jazzyk::AbstractKRModuleInterface, 19
- TVariableSubstitution
 - ExpressionsFwd.hpp, 239
- TVariableValue
 - ExpressionsFwd.hpp, 240
- TVecStrings
 - ExpressionsFwd.hpp, 240
- TVecTransformers
 - ExpressionsFwd.hpp, 240
- unify
 - CTransformerContext, 179
- unload
 - CModuleManager, 103, 104
- unloadDll
 - CSingleModule, 169
- unloadModules
 - CProgram, 128
- UPDATE
 - ModuleCommData.hpp, 261
- update
 - CModuleManager, 104
 - jazzyk::AbstractKRModuleInterface, 21
 - jazzyk::KRModuleInterface, 201
- version
 - CJazzykApp, 82
- Visit
 - CInterpreter, 73
 - CTraverser, 187–190
 - visitor, 219
 - visitor_arg, 221
- visitable, 214
 - ~visitable, 215
 - Accept, 215
 - AcceptImpl, 215
 - TReturnType, 214
- visitable_arg, 216
 - ~visitable_arg, 217
 - Accept, 217
 - AcceptImpl, 217
 - TArgType, 216
 - TReturnType, 216
- visitor, 218
 - ~visitor, 218
 - TReturnType, 218
 - Visit, 219
- visitor.hpp, 299
 - DECLARE_VISITABLE, 299, 300
 - DECLARE_VISITABLE_ARG, 300
- visitor_arg, 220
 - ~visitor_arg, 221
 - TArgType, 220
 - TReturnType, 220
 - Visit, 221
- waitForReady
 - CShmServerMgr, 162
- waitForRequest
 - CShmClientMgr, 143
- waitForResponse
 - CShmServerMgr, 162
- what
 - CError, 50
- writeRequest
 - CModuleManager, 105
- writeResponse
 - CSingleModule, 168
- yy, 14
- YY_USER_ACTION
 - Lexer.l++, 252
- yydebug
 - JazzykApp.cpp, 250
- yylex
 - CJazzykLexer, 89
 - Parser.y++, 269