Funktionsgruppe 1: Umgebungs- und Modellhandling

Funktionalität	CPLEX	XPRESS	LINDO	Common
Solver initialisieren (ggf. Lizenzprüfung)	-	XPRSinit	-	-
Solver schließen	-	XPRSfree	-	-
Environment anlegen und initialisieren. (ggf. Lizenzprüfung)	CPXopenCPLEX	-	LScreateEnv	CreateEnv
Environment schließen.	CPXcloseCPLEX	-	LSdeleteEnv	DeleteEnv
Modell anlegen und initialisieren.	CPXcreateprob	XPRScreateprob	LScreateModel	CreateModel
Modell löschen.	CPXfreeprob	XPRSdestroyprob	LSdeleteModel	DeleteModel
Modell replizieren.	CPXcloneprob	XPRScopyprob	-	CloneModel
CPLEX spezifische Copy-Funktionen Informationen aus einem Modell in ein anderes kopieren.				
Basis in Modell kopieren.	CPXcopybase	-	-	-
Statusinfo und duale Norms in Modell kopieren.	CPXcopybasednorms			
Variablentypinformation in Modell kopieren.	CPXcopyctype			
Dual Steepest Edge Norms in Modell kopieren.	CPXcopydnorms			
LP-Modell kopieren.	CPXcopylp			
LP-Modell mit Namensinformation kopieren.	CPXcopylpwnames			
MIP-Startwerte in Modell kopieren.	CPXcopymipstart			
Zielfunktionsnamen in Modell kopieren.	CPXcopyobjname			
Knoten-Prioritätsinformationen in Modell kopieren.	CPXcopyorder			
Partielle Basis in Modell kopieren.	CPXcopypartialbase			
Primale Steepest Edge Norms in Modell kopieren.	CPXcopypnorms			
Geschützte Variablen (dürfen nicht eliminiert werden) in Modell kopieren.	CPXcopyprotected			
SOS kopieren.	CPXcopysos			
Simplex Startinformationen kopieren.	CPXcopystart			
Statusinfo und duale Norms auslesen, bevor mit CPXcopybasednorms kopiert wird.	CPXgetbasednorms			

XPRESS spezifische Copy-Funktionen Informationen in ein Modell hineinkopieren.			
Parameter in Modell kopieren.	`XPRScopycontrols`	-	-
Callbacks in Modell Kopieren.	`XPRScopycallbacks`		
LINDO spezifische Speichermanagementfunktionen			
Hashing-Speicher für Variablennamen freigeben.		`LSfreeHashMemory`	-
Von MIP-Lösung belegten Speicher freigeben.		`LSfreeMIPSolutionMemory`	
Alle Lösungsinformationen und zugehörigen Speicher freigeben.		`LSfreeSolutionMemory`	
Speicher, der für Warmstart- und Zusatzinformationen belegt ist freigeben; Modell und Lösung bleiben erhalten.		`LSfreeSolverMemory`	

Funktionsgruppe 2: File I/O

Funktionalität		CPLEX	XPRESS	LINDO	Common
Modell einlesen.	MPS-Format.	`CPXreadcopyprob` Fix/Free Format, komprimiert und unkomprimiert.	`XPRSreadprob` Fix/Free Format.	`LSreadMPSFile` Fix/Free Format.	`ReadMpsFile`
	LP-Format.	`CPXreadcopyprob`	`XPRSreadprob`	-	`ReadLpFile`
	Solverspezifisches Format.	`CPXreadcopyprob` SAV-Format.	`XPRSrestore` Liest mit `XPRSsave` gespeicherte Modelle.	`LSreadLINDOFile`	-
Modell ausgeben.	MPS-Format.	`CPXwriteprob` Komprimiert und unkomprimiert.	`XPRSwriteprob`	`LSwriteMPSFile` Fix/Free Format.	`WriteMpsFile`
	LP-Format.	`CPXwriteprob`	`XPRSwriteprob`	-	`WriteLpFile`
	Solverspezifisches Format.	`CPXwriteprob` SAV-Format.	`XPRSsave` Speichert auch alle Statusinformation, Parameter, Lösungen etc.	`LSwriteLINDOFile` `LSwriteLINGOFile`	-
Duales Modell ausgeben.	MPS-Format.	`CPXdualwrite`	-	`LSwriteDualMPSFile` Fix/Free Format.	`WriteDualMpsFile`
	LP-Format.	-	-	-	-
	Solverspezifisches Format.	-	-	`LSwriteDualLINDOFile`	-

Lösungsdatei ausgeben.		CPXwritesol Binär- oder ASCII-Datei.	XPRSwritesol Ausgabedetails steuerbar. XPRSwriteprtsol Optimiert für Druckerausgabe. XPRSwriteomni OMNI-Format.	LSwriteSolution	WriteSolFile
Parameter-Datei.	einlesen.	CPXreadcopyparam	-	LSreadEnvParameter LSreadModelParameter	ReadParamFile
	ausgeben.	CPXwriteparam	-	-	WriteParamFile
Basis-Datei.	einlesen.	CPXreadcopybase	XPRSreadbasis	-	ReadBaseFile
	ausgeben.	CPXmbasewrite	XPRSwritebasis	-	WriteBaseFile
Branching-Prioritäten der Integervariablen	einlesen.	CPXreadcopyorder	XPRSreaddirs Liest nicht nur Prioritäten, sondern auch user defined cuts etc.	LSreadVarPriorities	ReadPrioFile
	ausgeben.	CPXordwrite	-	-	WritePrioFile
MIP-Startdatei (Warmstart)	einlesen.	CPXreadcopymipstart	-	-	ReadMipStartFile
	ausgeben.	CPXmstwrite	-	-	WriteMipStartFile
IIS in Datei ausgeben.		CPXiiswrite LP-Format.	XPRSiis	LSwriteIIS Lindo-Format.	WriteIISFile
IUS in Datei ausgeben.		-	-	LSwriteIUS Lindo-Format.	WriteIUSFile
Logfile setzen.		CPXsetlogfile	XPRSsetlogfile	-	SetLogFile
CPLEX spezifische File-Funktionen					
Barrier-Lösungsinformationen	einlesen.	CPXreadcopyvec	-	-	-
	ausgeben.	CPXvecwrite			
SOS-Informationen in MIP-Modell.	einlesen	CPXreadcopysos			
	ausgeben.	CPXsoswrite			
B&B/B&C Baum	einlesen.	CPXreadcopytree			
	ausgeben.	CPXtreewrite			
Presolved Modell ausgeben.		CPXpreslvwrite			
Perturbiertes Modell ausgeben.		CPXpperwrite (primal) CPXdperwrite (dual)			
Pointer auf Logfile liefern.		CPXgetlogfile			

CPLEX spezifische, sonstige File-Funktionen Als Ersatz für entsprechende Funktionen der Programmiersprache.				
Datei schließen.	`CPXfclose`	-	-	-
Datei öffnen, die später mit Channel verbunden wird.	`CPXfopen`			
String in ein offenes File schreiben.	`CPXfputs`			
CPLEX spezifische Channel-Funktionen				
Neues Channel-Objekt für ein Environment instanzieren.	`CPXaddchannel`	-	-	-
File als Ziel für einen Channel definieren.	`CPXaddfpdest`			
Ausgaben eines Channels an eine benutzerdefinierte Funktion leiten.	`CPXaddfuncdest`			
Channel löschen.	`CPXdelchannel`			
File von einem verbundenen Channel trennen.	`CPXdelfpdest`			
Benutzerdefinierte Funktion von einem verbundenen Channel trennen.	`CPXdelfuncdest`			
Channel von allen mit ihm verbundenen Zielen (Files, Funktionen) trennen.	`CPXdisconnectchannel`			
Gepufferten Inhalte eines Channels schreiben.	`CPXflushchannel`			
Gepufferten Inhalte der Standard-Channels schreiben.	`CPXflushstdchannels`			
Pointer zu den vier Standard-Channels liefern.	`CPXgetchannels`			
Ausgabe in den Message-Channel schreiben (C-Version).	`CPXmsg`			
Ausgabe in den Message-Channel schreiben (VB-Version).	`CPXmsgstr`			
XPRESS spezifische File-Funktionen				
Directives (Pseudo-Costs, User Cuts etc.) aus Datei lesen.		`XPRSreaddirs`	-	-
Modelländerungen einlesen.		`XPRSalter`		

Range-Analyse in Datei ausgeben.		`XPRSrange`
		Binärfile
		`XPRSwriterange`
		ASCII-File
		`XPRSwriteprtrange`
		Optimiert für Druckerausgabe		

Funktionsgruppe 3: Modellaufbau und -modifikation

Funktionalität		CPLEX	XPRESS	LINDO	Common
Variablen hinzufügen	mit Nonzeros	`CPXaddcols`	`XPRSaddcols`	`LSaddVariables`	`AddCols`
	ohne Nonzeros	`CPXnewcols`	-	-	`PutCols`
Restriktionen hinzufügen	mit Nonzeros	`CPXaddrows`	`XPRSaddrows`	`LSaddConstraints`	`AddRows`
	ohne Nonzeros	`CPXnewrows`	-	-	`PutRows`

Nonzeros hinzufügen und modifizieren.		`CPXchgcoef`
		Einzelnes Nonzero.
		`CPXchgcoeflist`
		Mehrere Nonzeros.
			`XPRSchgcoef`
			Einzelnes Nonzero.
			`XPRSchgmcoef`
			Mehrere Nonzeros.
				`LSmodifyAij`
				Einzelnes Nonzero.
				`LSmodifyAj`
				Mehrere Nonzeros einer Spalte.
					`PutNonzero`
					`PutNonzeros`

Ganzes Modell hinzufügen.		-	`XPRSloadlp`
			LP-Modell.
			`XPRSloadglobal`
			MIP-Modell ggf. inkl. Sets.
				`LSloadLPData`
				LP-Modell.
				`LSloadVarType`
				Setzt Variablentypen für MIP Modell.
				`LSloadNameData`
				Setzt Namen für Variablen, Restriktionen etc.
					`PutModel`

Restriktionen löschen.		`CPXdelrows`
		Zusammenhängender Block.
		`CPXdelsetrows`
		Unzusammenhängender Block.
			`XPRSdelrows`
			Unzusammenhängender Block.
				`LSdeleteConstraints`
				Unzusammenhängender Block.
					`DelRows`

Variablen löschen.		`CPXdelcols`
		Zusammenhängender Block.
		`CPXdelsetcols`
		Unzusammenhängender Block.
			`XPRSdelcols`
			Unzusammenhängender Block.
				`LSdeleteVariables`
				Unzusammenhängender Block.
					`DelCols`

Modifikation Restriktionen	Namen	`CPXchgrowname`
		Unzusammenhängender Block.
		`CPXchgname`
		Einzelne Restriktion.
			`XPRSaddnames`
			Zusammenhängender Block.
				-	`ChangeRowName`
					`ChangeRowNames`

	Typ	`CPXchgsense` Unzusammenhängender Block.	`XPRSchgrowtype` Unzusammenhängender Block.	`LSmodifyConstraintType` Unzusammenhängender Block.	`ChangeRowTypes`
	RHS	`CPXchgrhs` Unzusammenhängender Block.	`XPRSchgrhs` Unzusammenhängender Block.	`LSmodifyRHS` Unzusammenhängender Block.	`ChangeRhs`
	LHS	`CPXchgrngval` LHS ergibt sich aus Range, Unzusammenhängender Block.	`XPRSchgrhsrange` LHS ergibt sich aus Range, Unzusammenhängender Block.	-	`ChangeLhs`

Modifikation Variablen	Namen	`CPXchgcolname`
		Unzusammenhängender Block.
		`CPXchgname`
		Einzelne Variable.
			`XPRSaddnames`
			Zusammenhängender Block.
				-	`ChangeColName`
					`ChangeColNames`

	Typ	`CPXchgctype` Unzusammenhängender Block.	`XPRSchgcoltype` Unzusammenhängender Block.	`LSmodifyVariableType` Unzusammenhängender Block.	`ChangeColTypes`

	Untere Schranken	`CPXchgbds`
		Unzusammenhängender Block.
		`CPXtightenbds`
		Bei Re-Optimierung.
			`XPRSchgbounds`
			Unzusammenhängender Block.
				`LSmodifyLowerBounds`
				Unzusammenhängender Block.
					`ChangeColLBs`

	Obere Schranken	`CPXchgbds`
		Unzusammenhängender Block.
		`CPXtightenbds`
		Bei Re-Optimierung.
			`XPRSchgbounds`
			Unzusammenhängender Block.
				`LSmodifyUpperBounds`
				Unzusammenhängender Block.
					`ChangeColUBs`

	Zielfunktionskoeffizient	`CPXchgobj` Unzusammenhängender Block.	`XPRSchgobj` Unzusammenhängender Block.	`LSmodifyObjective` Unzusammenhängender Block.	`ChangeObjs`
Semi-Continuous Variablen	hinzufügen	`CPXnewcols`	`XPRSloadglobal`	-	Wie normale Variable.
	deklarieren	`CPXchgctype`	-	`LSloadSemiContData`	`ChangeColTypes`
	modifizieren	Wie normale Variable.	Wie normale Variable.	`LSmodifySemiContVars`	Wie normale Variable.
	löschen			`LSdeleteSemiContVars`	Wie normale Variable.
Sets	deklarieren	`CPXaddsos`	`XPRSloadglobal`	`LSaddSETS` `LSloadSETSData`	`AddSOS`

	modifizieren	-	-	LSmodifySET	ChangeSOS
	löschen	CPXdelsetsos	-	LSdeleteSETS	DelSOS
	Namen setzen	-	XPRSaddsetnames Zusammenhängender Block.	-	-
Modifikation Optimierungsrichtung (Min/Max)		CPXchgobjsen	-	Parameter LS_IPARAM_OBJSENSE	ChangeObjSen
Zielfunktionskonstante modifizieren			XPRSchgobj mit Index -1	LSmodifyObjConstant	ChangeObjConst
Modellnamen setzen.		CPXchgprobname	XPRSsetprobname	LSloadNameData	ChangeModelName
CPLEX spezifische Funktionen					
"Lazy-Constraints" hinzufügen. Diese dürfen nicht verletzt werden, sind aber nicht Teil der Matrix.		CPXaddlazyconstraints	-	-	-
Alle Lazy-Constraints löschen.		CPXfreelazyconstraints			
Alle Zeilen- und Spaltennamen löschen.		CPXdelnames			
Modelltyp (LP, IP, Quadratisch etc.) ändern.		CPXchgprobtype			
Restriktionen inkl. Nonzeros dem Originalmodell und dem Presolved Modell hinzufügen, wobei ein bestehendes Presolved Modell beibehalten wird.		CPXpreaddrows			

Funktionsgruppe 4: Modellabfrage

Funktionalität		CPLEX	XPRESS	LINDO	Common
Variablen auslesen	Namen	`CPXgetcolname` Zusammenhängender Block.	`XPRSgetnames` Zusammenhängender Block.	`LSgetVariableNamej` Einzelner Name.	`GetColName` `GetColNames`
	Typ	`CPXgetctype` Zusammenhängender Block.	`XPRSgetcoltype` Zusammenhängender Block.	`LSgetVarType` Alle Spalten.	`GetColType` `GetColTypes`
	Untere Schranken	`CPXgetlb` Zusammenhängender Block.	`XPRSgetlb` Zusammenhängender Block.	`LSgetLPVariableDataj` Einzelne Spalte, liefert auch Typ und Nonzeros.	`GetColLB` `GetColLBs`
	Obere Schranken	`CPXgetub` Zusammenhängender Block.	`XPRSgetub` Zusammenhängender Block.		`GetColUB` `GetColUBs`
	Zielfunktionskoeffizient	`CPXgetobj` Zusammenhängender Block.	`XPRSgetobj` Zusammenhängender Block.		`GetObj` `GetObjs`
Restriktionen auslesen	Namen	`CPXgetrowname` Zusammenhängender Block.	`XPRSgetnames` Zusammenhängender Block.	`LSgetConstraintNamei` Einzelne Restriktion.	`GetRowName` `GetRowNames`
	Typ	`CPXgetsense` Zusammenhängender Block.	`XPRSgetrowtype` Zusammenhängender Block.	`LSgetConstraintDatai` Einzelne Restriktion.	`GetRowType` `GetRowTypes`
	RHS	`CPXgetrhs` Zusammenhängender Block.	`XPRSgetrhs` Zusammenhängender Block.		`GetRowRHS` `GetRowRHSs`
	LHS	`CPXgetrngval` LHS ergibt sich aus Range, Zusammenhängender Block.	`XPRSgetrhsrange` LHS ergibt sich aus Range, Zusammenhängender Block.	-	`GetRowLHS` `GetRowLHSs`
Nonzeros auslesen	Einzeln	`CPXgetcoef`	-	-	`GetNonzero`
	Zeilenweise	`CPXgetrows` Zusammenhängender Block.	`XPRSgetrows` Zusammenhängender Block.	`LSgetLPConstraintDatai` Einzelne Restriktion.	`GetRowNZs`
	Spaltenweise	`CPXgetcols` Zusammenhängender Block.	`XPRSgetcols` Zusammenhängender Block.	`LSgetLPVariableDataj` Einzelne Spalte.	`GetColNZs`

Namen auslesen	Modell	CPXgetprobname	XPRSgetprobname	LSgetNameData	GetModelName
	Zielfunktion	CPXgetobjname	Parameter OBJNAME		GetObjName
	RHS-, Bounds-Namen etc.	Div. Parameter	Div. Parameter	Div. Parameter	Div. Parameter
Modelldimensionen auslesen	Zeilen	CPXgetnumrows	Parameter ROWS	Attribut LS_IINFO_NUM_CONS	GetNumRows
	Spalten	CPXgetnumcols	Parameter COLS	Attribut LS_IINFO_NUM_VARS	GetNumCols
	Nonzeros	CPXgetnumnz	Parameter ELEMS	Attribut LS_IINFO_NUM_NONZ	GetNumNZs
	Integervariablen	CPXgetnumint	Parameter MIPENTS XPRSgetglobal	Attribut LS_IINFO_NUM_INT	GetNumInt
	Binärvariabeln	CPXgetnumbin		Attribut LS_IINFO_NUM_BIN	GetNumBin
	Semi-Continuous Variablen	CPXgetnumsemicont		LSgetSemiContData	GetNumSemiCont
	Semi-Integer Variablen	CPXgetnumsemiint		-	GetNumSemiInt
	SOS	CPXgetnumsos	Parameter SETS	LSgetSETSData	GetNumSOS
Index suchen	Variablennamen	CPXgetcolindex	XPRSgetindex	LSgetVariableIndex	GetColIndex
	Restriktionsnamen	CPXgetrowindex		LSgetConstraintIndex	GetRowIndex
Komplettes Modell auslesen.		-	-	LSgetLPData	GetModel
Semi-Continuous Variablen auslesen (Ober- und Untergrenze, Indizes).		Wie normale Variablen.	Wie normale Variablen.	LSgetSemiContData	Wie normale Variablen.
Sets auslesen.		CPXgetsos	XPRSgetglobal Auch Information über Semi-Continuous, Partial-Integer etc.	LSgetSETSData LSgetSETSDatai	GetSOS
Optimierungsrichtung auslesen.		CPXgetobjsen	Parameter OBJSENSE	Parameter LS_IPARAM_OBJSENSE	GetObjSen
Branching Priorität Integervariablen auslesen.		CPXgetorder Zusammenhängender Block.	XPRSgetdirs	-	GetPrio
CPLEX spezifische Funktionen					
Geschützte Variablen (werden nicht im Presolve eliminiert) ausgeben.		CPXgetprotected	-	-	-
Modelltyp (LP, MIP etc.) auslesen.		CPXgetprobtype			
Lindo spezifische Funktionen für Blockstrukturen					
Blockstrukturen in der Matrix suchen.				LSfindBlockStructure	-
Gefundene Blockstrukturen auslesen				LSgetBlockStructure	
Blockstrukturen definieren.				LSloadBlockStructure	

Funktionsgruppe 5: Lösungsabfrage und -analyse

Funktionalität		CPLEX	XPRESS	LINDO	Common

Variablen	LP	`CPXgetx`
		Zusammenhängender Block.
		`CPXsolution`
		Alle Variablen.
			`XPRSgetsol`
			Alle Variablen.
				`LSgetSolution`
				`LSgetPrimalSolution`
				Alle Variablen.
					`GetLPSolution`

	MIP	`CPXgetmipx` Zusammenhängender Block.	`XPRSgetsol` Alle Variablen.	`LSgetMIPPrimalSolution` Alle Variablen.	`GetIPSolution`

	Reduzierte Kosten	`CPXgetdj`
		Zusammenhängender Block.
		`CPXsolution`
		Alle Variablen.
			`XPRSgetsol`
			Alle Variablen.
				`LSgetSolution`
				`LSgetReducedCosts`
				LP-Modell, alle Variablen.
				`LSgetMIPReducedCosts`
				MIP-Modell, alle Variablen, Integervariablen fixiert.
					`GetRedCost`

	Basis-Status	`CPXgetbase`
		Alle Variablen.
			`XPRSgetbasis`
			Alle Variablen.
				`LSgetBasis`
				LP-Modell, alle Variablen.
				`LSgetMIPBasis`
				MIP-Modell, alle Variablen.
					`GetBaseStat`

Restriktionen	Activities	`CPXgetax` Zusammenhängender Block.	-	`LScalcConFunc` Einzelne Restriktion.	`GetActivities`

	Dualwerte	`CPXgetpi`
		Zusammenhängender Block.
		`CPXsolution`
		Alle Restriktionen.
			`XPRSgetsol`
			Alle Restriktionen.
				`LSgetSolution`
				`LSgetDualSolution`
				LP-Modell, alle Restriktionen.
				`LSgetMIPDualSolution`
				MIP-Modell, alle Restriktionen, Integervariabeln fixiert.
					`GetDuals`

	Schlupfvariablen	`CPXgetslack`
		LP, zusammenhängender Block.
		`CPXgetmipslack`
		MIP, zusammenhängender Block.
		`CPXsolution`
		Alle Restriktionen.
			`XPRSgetsol`
			Alle Restriktionen.
				`LSgetSolution`
				`LSgetSlacks`
				LP-Modell, alle Restriktionen.
				`LSgetMIPSlacks`
				MIP-Modell, alle Restriktionen.
					`GetSlacks`

	Basis-Status	`CPXgetbase`
		Alle Restriktionen.
			`XPRSgetbasis`
			Alle Restriktionen.
				`LSgetBasis`
				LP-Modell, alle Restriktionen.
				`LSgetMIPBasis`
				MIP-Modell, alle Restriktionen.
					`GetBaseStat`

LP Zielfunktionswert		`CPXgetobjval` `CPXsolution`	Parameter `LPOBJVAL`	`LScalcObjFunc` Auch Parameter wie `LS_DINFO_POBJ` etc.	`GetLPObjVal`
MIP Zielfunktionswert		`CPXgetmipobjval` `CPXsolution`	Parameter `MIPOBJVAL`		`GetIPObjVal`
IIS	Suchen.	`CPXfindiis`	`XPRSiis`	`LSfindIIS`	`FindIIS`

	Auslesen.	`CPXgetiis`
		`CPXdisplayiis`
		Sendet IIS-Information zur Ausgabe an Channnel.
			`XPRSgetiis`	`LSgetIIS`	`GetIIS`

IUS	Suchen.	-	-	`LSfindIUS`	`FindIUS`
	Auslesen.	-	-	`LSgetIUS`	`GetIUS`
Range Analyse	Variablenbounds.	`CPXboundsa`	`XPRSrange` + `XPRSgetcolrange`	`LSgetBoundRanges`	`GetBoundRanges`

	Restriktionen / RHS.	`CPXrhssa`	`XPRSrhssa`
			Ober und Untergrenze RHS.
			`XPRSrange` +
			`XPRSgetrowrange`
			Erweiterte Analyse, schreibt Rangefile.
				`LSgetConstraintRanges`	`GetRowRanges`

	Zielfunktionskoeffizienten	`CPXobjsa`	`XPRSobjsa`	`LSgetObjectiveRanges`	`GetObjRanges`

Lösungsinformationswerte.
Attribute, die Zusatzinformationen zur Lösung (z.B. Anzahl Iterationen, Anzahl Knoten u.v.m) liefern, werden teils über solver-spezifische Funktionen (s. unten), teils über Parameter ausgelesen.
		Unten aufgeführte spezifische Funktionen sowie folgende Funktionen zum Auslesen von Double-, Integer- und String-Attributen:
		`CPXgetdblparam`
		`CPXgetintparam`
		`CPXgetstrparam`
			`XPRSgetdblattrib`
			Auslesen Double-Attribut.
			`XPRSgetintattrib`
			Auslesen Integer-Attribut.
			`XPRSgetstrattrib`
			Auslesen String-Attribut.
				`LSgetInfo`	`GetDblParameter`
					`GetIntParameter`
					`GetStrParameter`

CPLEX spezifische Funktionen zur Abfrage von Lösungsinformationswerten					
Lösungsstatus.	Aktueller Status als Integerwert.	`CPXgetstat`	-	-	ggf. via Parameter
	Aktueller Status als String.	`CPXgetstatstring`			
Simplexiterationen	Kumulierte Iterationszahl in MIP.	`CPXgetmipitcnt`			
	Gesamtzahl Iterationen in LP.	`CPXgetitcnt`			
	Anzahl der Phase 1 Iterationen des primalen oder dualen Simplex.	`CPXgetphase1cnt`			
Crossover	Anzahl primale Tauschiterationen.	`CPXgetcrosspexchcnt`			
	Anzahl primale Pushiterationen.	`CPXgetcrossppushcnt`			
	Anzahl duale Tauschiterationen.	`CPXgetcrossdexchcnt`			
	Anzahl duale Pushiterationen.	`CPXgetcrossdpushcnt`			
Gesamtzahl der erfolgten Barrier-Iterationen bis zur LP Lösung.		`CPXgetbaritcnt`			
Anzahl der für die Lösung des MIP-Modells untersuchten Knoten.		`CPXgetnodecnt`			
Knotennummer des Knotens mit der besten Integerlösung.		`CPXgetnodeint`			
Anzahl der noch nicht untersuchten Knoten eines MIP.		`CPXgetnodeleftcnt`			
Superbasic-Variablen	Anzahl primaler SB-Variablen.	`CPXgetpsbcnt`			
	Anzahl dualer SB-Variablen.	`CPXgetdsbcnt`			

Optimierungsmethode für das letzte MIP-Subproblem auslesen.		`CPXgetsubmethod`			
Optimierungsstatus für letztes MIP-Subproblem auslesen (nach Fehler).		`CPXgetsubstat`			
Bester bisher gefundener Zielfunktionswert.		`CPXgetbestobjval`			
Anzahl der in der MIP-Optimierung hinzugefügten Clique Cuts.		`CPXgetclqcnt`			
Anzahl der in der MIP-Optimierung hinzugefügten Cover Cuts.		`CPXgetcovcnt`			
Double-Wert für diverse Lösungsqualitätsindikatoren.		`CPXgetdblquality`			
Aktuellen Cutoff-Wert während der MIP-Optimierung.		`CPXgetcutoff`			
Anzahl der Clique-Ungleichungen zu Beginn der MIP-Optimierung.		`CPXgetgenclqcnt`			
Sifting-Algorithmus (falls benutzt)	Anzahl Sifting-Iterationen.	`CPXgetsiftitcnt`			
	Anzahl Sifting-Iterationen in Phase 1.	`CPXgetsiftphase1cnt`			
Lösungsmethode (Primal, Dual, Barrier) auslesen.		`CPXgetmethod`			
Diverse Lösungsinformationen (Lösungsmethode, primale und duale Lösbarkeit etc.) abfragen		`CPXsolninfo`			
CPLEX spezifische Funktionen zur erweiterten Lösungsanalyse					
L∞-Norm für Ax - b als Maß des Rundungsfehlers berechnen		`CPXcheckax`	-	-	-
L∞-Norm für cB'-π'B als Maß des Rundungsfehlers der dualen Lösung π.		`CPXcheckpib`			
Marginal-Analyse von Variabeln in Basislösung.		`CPXgetgrad`			
Zeilen oder Spaltenindex einer "Diverging Variable" abfragen.		`CPXgetijdiv`			
Position einer Variablen im Basis-Header abfragen.		`CPXgetijrow`			
Integer-Wert für Indizes diverser Lösungsqualitätsindikatoren auslesen.		`CPXgetintquality`			
Werte der Integervariablen des MIP-Startmodells auslesen.		`CPXgetmipstart`			
Zielfunktionsoffset zwischen Original- und Presolved-Problem abfragen.		`CPXgetobjoffset`			
Status der Zeilen und Spalten im Original- und Presolved-Modell. abfragen.		`CPXgetprestat`			

Pointer auf das Presolved Problem auslesen.		CPXgetredlp			
Steepest-Edge Norms	Primale Norms auslesen.	CPXgetpnorms			
	Duale Norms ausgeben.	CPXgetdnorms			
	Primale Norms löschen.	CPXkillpnorms			
	Duale Norms löschen.	CPXkilldnorms			
"Ray"-Vektor eines unbegrenzten Modells auslesen.		CPXgetray			
"Driebeek penalties" auslesen. (Details s. Manual).		CPXmdleave			
Reduzierte Kosten aus Dualwerten berechnen.		CPXdjfrompi			
Exaktes Kappa berechnen und auslesen.		CPXgetExactkappa			
Geschätztes Kappa berechnen und auslesen.		CPXgetkappa			
Bestimmtes Maß für den Grad dualer Unlösbarkeit ausgeben.		CPXdualfarkas			
Strong Branching Information für Integervariablen berechnen.		CPXstrongbranch			
XPRESS spezifische Funktionen zur Lösungsanalyse					
Double-Wert für eine Vielzahl von Modellattributen (z.B. Summe primaler Unzulässigkeiten, Wert der Branching Variablen etc.) auslesen.			XPRSgetdblattrib	-	-
Integer-Wert für eine Vielzahl von Modellattributen (z.B. Anzahl Zeilen und Spalten, Anzahl Primärer Unzulässigkeiten, LP Status etc.) auslesen.			XPRSgetintattrib		
String-Wert für mehrere Modellattribute (z.B. Matrixname, Zielfunktionsname etc.) auslesen.			XPRSgetstrattrib		
"Condition" als Maß für die Lösungsgenauigkeit der skalierten und unskalierten Basis berechnen.			XPRSbasiscondition		
Liste von infeasible Variablen (primal / dual) auslesen.			XPRSgetinfeas		
Pivot-Folge nach Optimierung auslesen.			XPRSgetpivotorder		
Liste aller Variablen, die potentiell die Matrix verlassen können.			XPRSgetpivots		
Presolved-Basis in Form von Statusarrays für Zeilen und Spalten auslesen.			XPRSgetpresolvebasis		
Mapping von Presolved-Problem zu Original-Problem auslesen.			XPRSgetpresolvemap		
Lösung des Presolved-Modells auslesen.			XPRSgetpresolvesol		
Liste skalierter Variablen, die infeasible sind (primal/dual) auslesen.			XPRSgetscaledinfeas		
Vektor, der die Unbegrenztheit des Modells verursacht auslesen.			XPRSgetunbvec		
LINDO spezifische Funktionen zur Lösungsanalyse					
Wert der primalen Anfangslösung aller Variablen.				LSgetVarStartPoint	-

Funktionsgruppe 6: Lösungsprozesssteuerung

Funktionalität	CPLEX	XPRESS	LINDO	Common

Start LP-Optimierung.	`CPXlpopt`
	Startet mit Methode gem. Parameter CPX_PARAM_LPMETHOD.
	`CPXprimopt`
	Startet mit primalem Simplex.
	`CPXdualopt`
	Startet mit dualem Simplex.
	`CPXbaropt`
	Startet mit Barrier-Methode.
	`CPXhybbaropt`
	Startet mit hybrider Methode: Erst Barrier, dann primaler oder dualer Simplex für Crossover.
		`XPRSmaxim`
		`XPRSminim`
			`LSoptimize`	`Optimize`

Start MIP-Optimierung.	`CPXmipopt`	`XPRSglobal` Nach LP-Optimierung.	`LSsolveMIP`	`Optimize`
Startbasis setzen.	Nur Einlesen aus File.	`XPRSloadbasis`	`LSloadBasis`	`LoadBasis`

Benutzerdefinierte Cuts hinzufügen.	`CPXaddusercuts`	`XPRSaddcuts`
		Fügt Cuts dem Cutpool und dem aktuellen Knoten hinzu.
		`XPRSloadmodelcuts`
		Definiert Modellzeilen als Cuts.
		`XPRSloadcuts`
		Lädt Cuts aus Cutpool und aktivierte diese am aktuellen Knoten.
			-	`AddCuts`

Benutzerdefinierte Cuts löschen.	`CPXfreeusercuts`	`XPRSdelcpcuts` Bestimmte Cuts aus Cutpool. `XPRSdelcuts` Bestimmte Cuts am aktuellen Knoten.	-	`DelCuts`
Simplex Pivotschritt ausführen.	`CPXpivot`	`XPRSpivot`		`Pivot`
Backward-Transformation (BTRAN) $B^tX = Y$ ausführen.	`CPXbtran`	`XPRSbtran`	`LSdoBTRAN`	`Btran`
Forward-Transformation (FTRAN) $B X = Y$ ausführen.	`CPXftran`	`XPRSftran`	`LSdoFTRAN`	`Ftran`
CPLEX spezifische Funktionen zur Lösungsprozesssteuerung				
Start der Lösung eines benutzerdefiniert relaxierten Modells, mit dem Ziel auf jeden Fall eine gültige Lösung zu bekommen.	`CPXfeasopt`	-	-	-
Presolve durchführen.	`CPXpresolve`			
Lineare Form des Originalmodells auf Presolved-Modell übertragen.	`CPXcrushform`			
Lineare Form des Presolved-Modells auf Originalmodell übertragen.	`CPXuncrushform`			
Dualwerte des Originalmodells auf die des Presolved Modells übertragen.	`CPXcrushpi`			
Dualwerte des Presolved Modells auf die des Originalmodells übertragen.	`CPXuncrushpi`			
Lösungswerte des Originalmodells auf die des Presolved Modells übertragen.	`CPXcrushx`			
Lösungswerte des Presolved Modells auf die des Originalmodells übertragen.	`CPXuncrushx`			
Presolved Modell löschen (erfolgt i.d.R. automatisch).	`CPXfreepresolve`			
Liste von Schlupfvariablen in die Basis pivotieren.	`CPXpivotin`			
Liste von Variablen aus der Basis pivotieren.	`CPXpivotout`			
Liste von Schlupfwerten zu vorgegebener primaler Lösung berechnen.	`CPXslackfromx`			
Basis-Header auslesen.	`CPXgetbhead`			
Bound-Verschärfung durchführen und redundante Restriktionen suchen (Erstellt kein Presolved Modell).	`CPXbasicpresolve`			

Zielfunktionskoeffizienten im Original- und im Presolved-Modell ändern.	`CPXprechgobj`			
LP-Optimierung abschließen (Aufruf normalerweise nicht erforderlich).	`CPXcompletelp`			
Modell deskalieren.	`CPXunscaleprob`			
Gleichungsystem Bx = Aj lösen (mit Basis B und Spalte Aj).	`CPXbinvacol`			
Zeile i eines Simplextableaus berechnen.	`CPXbinvarow`			
Spalte j der Basisinversen berechnen.	`CPXbinvcol`			
Zeile i der Basisinversen berechnen.	`CPXbinvrow`			
XPRESS spezifische Funktionen zur Lösungsprozesssteuerung				
Branching-Prioritäten und -Richtungen sowie Pseudokosten für MIP-Modelle auslesen.		`XPRSgetdirs`	-	-
Benutzerdefinierte Directions und entsprechende Pseudokosten setzen.		`XPRSloaddirs`		
Benutzerdefinierte Directions (Force up/down) für die Presolved-Matrix setzen.		`XPRSloadpresolvedirs`		
Presolved Basis durch Angabe des Basisstatus für alle Variablen definieren.		`XPRSloadpresolvebasis`		
Aktuelle Basis neu skalieren.		`XPRSscale`		
Presolve für einen Cut durchführen, so dass dieser einer presolved Basis hinzugefügt werden kann.		`XPRSpresolvecut`		
Spalten und Zeilen bestimmen, die vom Presolve unberührt bleiben sollen.		`XPRSloadsecurevecs`		
Knoten aus der Liste der zu untersuchenden Knoten löschen.		`XPRSdelnode`		
Integervariablen auf ihre Lösungswerte fixieren.		`XPRSfixglobal`		
Anzahl angegebener Cuts aus dem Cut Pool liefern.		`XPRSgetcpcuts`		
Indexliste der Cuts aus dem Cut Pool liefern.		`XPRSgetcpcutlist`		
Indexliste der Cuts des aktuellen Knotens liefern.		`XPRSgetcutlist`		
Cuts in den Cut Pool speichern ohne sie am aktuellen Knoten anzuwenden.		`XPRSstorecuts`		
Globalen Suchbaum eines Modells reinitialisieren.		`XPRSinitglobal`		
LINDO spezifische Funktionen zur Lösungsprozesssteuerung				
Branching-Prioritäten für Variablen setzten.			`LSloadVarPriorities`	-
Engste Ober- und Untergrenzen für Variablen ermitteln.			`LSgetBestBounds`	

Funktionsgruppe 7: Parameterhandling

Funktionalität	CPLEX	XPRESS	LINDO	Common
Double-Parameter auslesen.	CPXgetdblparam	XPRSgetdblcontrol	LSgetEnvDouParameter LSgetModelDouParameter	GetDblParameter
Integer-Parameter auslesen.	CPXgetintparam	XPRSgetintcontrol	LSgetEnvIntParameter LSgetModelIntParameter	GetIntParameter
String-Parameter auslesen.	CPXgetstrparam	XPRSgetstrcontrol	LSgetEnvParameter LSgetModelParameter	GetStrParameter
Typunabhängigen Parameter auslesen.	-	-	LSgetEnvParameter LSgetModelParameter	GetParameter
Double-Parameter setzen.	CPXsetdblparam	XPRSsetdblcontrol	LSsetEnvDouParameter LSsetModelDouParameter	SetDblParameter
Integer-Parameter setzen.	CPXsetintparam	XPRSsetintcontrol	LSsetEnvIntParameter LSsetModelIntParameter	SetIntParameter
String-Parameter setzen.	CPXsetstrparam	XPRSsetstrcontrol	LSsetEnvParameter LSsetModelParameter	SetStrParameter
Typunabhängigen Parameter setzen.	-	-	LSsetEnvParameter LSsetModelParameter	SetParameter
Parameter auf Default setzen.	CPXsetdefaults	XPRSsetdefaults	-	SetDefaults
Default, Minimum und Maximum für Double-Parameter auslesen.	CPXinfodblparam	-	-	GetDblParameterInfo
Default, Minimum und Maximum für Integer-Parameter auslesen.	CPXinfointparam	-	-	GetIntParameterInfo
Default für String-Parameter auslesen.	CPXinfostrparam	-	-	GetStrParameterInfo
Zu Parametercode gehörigen Parameternamen finden.	CPXgetparamname	-	-	GetParametername
Zu Parameternamen gehörigen Parametercode finden.	CPXgetparamnum	-	-	GetParameterID

Funktionsgruppe 8: Callbacks

Funktionalität		CPLEX	XPRESS	LINDO	Common
Callback-Funktion setzten	bei Fehler im B&B/B&C oder Ressourcenlimit.	`CPXsetbranchnosolncallbackfunc`	-	`LSsetCallback` Setzt benutzerdefinierte Callbackfunktion, die in zeitlichem Intervall aufgerufen wird und nicht bei bestimmten Events.	`SetCallbackX` Setzt Callbackfunktion die bei Event X aufgerufen wird. `SetCallbackTimer` Setzt Callbackfunktion die in zeitlichen Intervallen aufgerufen wird.
	nach Auswahl Branchingvariable.	`CPXsetbranchcallbackfunc`	`XPRSsetcbchgbranch`		
	wenn MIP-Subproblem als LP lösbar ist und unterhalb der Cuts liegt.	`CPXsetcutcallbackfunc`	-		
	wenn Knoten gelöscht wird.	`CPXsetdeletenodecallbackfunc`	-		
	nach optimaler Lösung des MIP-Subproblems.	`CPXsetheuristiccallbackfunc`	`XPRSsetcboptnode`		
	nach jeder LP-Iteration.	`CPXsetlpcallbackfunc`			
	vor jedem Lösen eines Knotens.	`CPXsetmipcallbackfunc`			
	nachdem ein Knoten zur Lösung ausgewählt wurde, aber bevor dieser weiterbearbeitet wird.	`CPXsetnodecallbackfunc`	`XPRSsetcbchgnode`		
	zur Lösung eines MIP-Subproblems.	`CPXsetsolvecallbackfunc`	-		
	wenn bessere Integerlösung gefunden wurde.	`CPXsetincumbentcallbackfunc`	-		
	bei jeder Iteration des Barrier-Algorithmus.	-	`XPRSsetcbbarlog`		
	wenn ein Log-Message bei einem Cut geschrieben wird.	-	`XPRSsetcbcutlog`		
	an jedem Knoten um bestimmte Schätzwerte zu ermitteln (Details s. Manual).	-	`XPRSsetcbestimate`		
	bei jedem globalen Logging.	-	`XPRSsetcbgloballog`		
	bei jedem Logging im Simplex.	-	`XPRSsetcblplog`		
	bei jedem Textoutput.	-	`XPRSsetcbmessage`		
	wenn Knoten unlösbar ist.	-	`XPRSsetcbinfnode`		
	wenn ein Knoten aufgrund besserer Integerlösung abgeschnitten wird.	-	`XPRSsetcbnodecutoff`		
	nach Preprocessing, aber vor Optimierung eines Knotens.	-	`XPRSsetcbprenode`		
	zur Auswahl der Separierung (Branch oder Cut) an jedem Knoten.	-	`XPRSsetcbsepnode`		
	wenn neue Integerlösung gefunden wurde.	-	`XPRSsetcbintsol`	`LSsetMIPCallback`	

	wenn Logging erfolgt.	-	-	LSsetModelLogFunc	
	wenn Logging für das Environment erfolgt.	-	-	LSsetEnvLogFunc	
Pointer auf Callbackfunk–tion auslesen	nach Auswahl Branchingvariable.	CPXgetbranchcallbackfunc	-	-	GetCallbackX GetCallbackTimer
	wenn MIP-Subproblem als LP lösbar ist und unterhalb der Cuts liegt.	CPXgetcutcallbackfunc	-	-	
	wenn Knoten gelöscht wird.	CPXgetdeletenodecallbackfunc	-	-	
	nach optimaler Lösung des MIP-Subproblems.	CPXgetheuristiccallbackfunc	-	-	
	wenn bessere Integerlösung gefunden wurde.	CPXgetincumbentcallbackfunc	-	-	
	nach jeder LP-Iteration.	CPXgetlpcallbackfunc	-	-	
	vor jedem Lösen eines Knotens.	CPXgetmipcallbackfunc	-	-	
	nachdem ein Knoten zur Lösung ausgewählt wurde, aber bevor dieser weiterbearbeitet wird.	CPXgetnodecallbackfunc	-	-	
	zur Lösung eines MIP- Subproblems.	CPXgetsolvecallbackfunc	-	-	
Lösungs-informations-funktionen zum Aufruf nur aus Callback-funktionen.	Variablentypen auslesen.	CPXgetcallbackctype	-	LSgetCallbackInfo aus LP-Callbacks LSgetMIPCallbackInfo aus LP- und MIP-Callbacks (Nicht für alle links stehenden Informationen gibt es eine Entsprechung)	-
	Beste bekannte MIP-Untergrenze auslesen.	CPXgetcallbackgloballb	-		
	Beste bekannte MIP-Obergrenze auslesen.	CPXgetcallbackglobalub	-		
	Lösungswerte der besten MIP-Lösung auslesen.	CPXgetcallbackincumbent	-		
	Diverse Lösungsinformationswerte auslesen (Zielfunktionswerte, Iterationszahl etc.)	CPXgetcallbackinfo	-		
	Pointer auf Modell, das den Callback aufrief	CPXgetcallbacklp	-		
	Knotenbezogene Information (Tiefe im B&C Baum etc.) auslesen.	CPXgetcallbacknodeinfo	-		
	Integer-feasibility von Variablen eines Blocks.	CPXgetcallbacknodeintfeas	-		
	Untergrenze der Variablen des Subproblems am aktuellen Knoten.	CPXgetcallbacknodelb	-		
	Pointer auf das LP Subproblem am aktuellen Knoten.	CPXgetcallbacknodelp	-		
	Zielfunktionswert des Subproblems am aktuellen Knoten.	CPXgetcallbacknodeobjval	-		

	Lösungsstatus des Subproblems am aktuellen Knoten.	`CPXgetcallbacknodestat`	-		
	Obergrenze der Variablen des Subproblems am aktuellen Knoten.	`CPXgetcallbacknodeub`	-		
	Primaler Lösungswert der Variablen am aktuellen Knoten.	`CPXgetcallbacknodex`	-		
	Prioritätenliste des MIP-Modells.	`CPXgetcallbackorder`	-		
	Up-und Down-Pseudokosten für einen Variablenblock eines MIP-Modells.	`CPXgetcallbackpseudocosts`	-		
	Sequenznummer eines Knotens.	`CPXgetcallbackseqinfo`	-		
	Information über SOS (Feasibility, Priorität etc.) während der MIP-Optimierung.	`CPXgetcallbacksosinfo`	-		
CPLEX spezifische callbackbezogene Funktionen					
Branches für benutzerspezifisches Branching spezifizieren.		`CPXbranchcallbackbranchbds`	-	-	-
Restriktionen für benutzerspezifisches Branching spezifizieren.		`CPXbranchcallbackbranchconstraints`			
Branches für benutzerspezifisches Branching mit zusätzlichen Restriktionen und Bound-Änderungen der Variablen spezifizieren.		`CPXbranchcallbackbranchgeneral`			
Benutzerdefinierten Cut hinzufügen.		`CPXcutcallbackadd`			
Benutzerdefinierten, lokalen Cut hinzufügen.		`CPXcutcallbackaddlocal`			
XPRESS spezifische callbackbezogene Funktionen					
Benutzerdefinierte Branching-Cuts setzten.			`XPRSsetbranchcuts`	-	-
Callbackfunktion setzten als benutzerdefinierten Cut-Manager an jedem Knoten.			`XPRSsetcbcutmgr`		
Initialisierung des benutzerdefinierten Callback-Cut-Managers.			`XPRSsetcbinitcutmgr`		
Terminierung des benutzerdefinierten Callback-Cut-Managers.			`XPRSsetcbfreecutmgr`		
Speichern der Bounds in Callbackfunktion `XPRSsetcbsepnode`.			`XPRSstorebounds`		
`Setzten der in XPSRstorebounds` gespeicherten Bounds.			`XPRSsetbranchbounds`		

Funktionsgruppe 9: Fehlermeldungen

Funktionalität	CPLEX	XPRESS	LINDO	Common
Fehlermeldungen bzgl. Dateizugriff	Funktionen liefern Returncodes, welche in explizite Fehlermeldung gewandelt werden via `CPXgeterrorstring`	-	`LSgetFileError`	via Returncodes
Fehlermeldungen bzgl. Lizenz		`XPRSgetlicerrmsg`	-	
Allgemeine Fehlermeldungen		`XPRSgetlasterror`	`Nur Retuncodes`	
Errorstring zu Errorcode liefern.	`CPXgeterrorstring`	-	`LSgetErrorMessage`	`GetErrorString`
LINDO spezifische Funktionen				
Zeilenindex, wo ein numerischer Fehler auftrat, liefern.			`LSgetErrorRowIndex`	-

Funktionsgruppe 10: Sonstige Funktionen

Funktionalität	CPLEX	XPRESS	LINDO	Common
Lizenzinformation	-	`XPRSgetdaysleft`	`LSloadLicenseString`	`GetLicenseInfo`
Versionsinformation	`CPXversion`	`XPRSgetbanner` `XPRSgetversion`	`LSgetVersionInfo`	`GetVersionInfo`
CPLEX spezifische Check-Funktionen				
Validation Argumente der Routine `CPXaddcols`	`CPXcheckaddcols`	-	-	-
Validation Argumente der Routine `CPXaddrows`	`CPXcheckaddrows`			
Validation Argumente der Routine `CPXchgcoeflist`	`CPXcheckchgcoeflist`			
Validation Argumente der Routine `CPXcopyctype`	`CPXcheckcopyctype`			
Validation Argumente der Routine `CPXcopylp`	`CPXcheckcopylp`			
Validation Argumente der Routine `CPXcopylpwnames`	`CPXcheckcopylpwnames`			
Validation Argumente der Routine `CPXcopysos`	`CPXcheckcopysos`			
Validation Argumente der Routine `CPXvals`	`CPXcheckvals`			
CPLEX spezifische Ersatz-Funktionen Ersetzten bestimme Funktionen der Programmiersprache oder des Systems.				
Umgebungsvariable des Betriebssystems setzen.	`CPXputenv`	-	-	-
Zeichenketten kopieren.	`CPXstrcpy`			
Länge einer Zeichenkette ermitteln.	`CPXstrlen`			

Anhang 2: Klassenübersicht BCL 3.0

XPRB		
Methode	Aufruf C-Funktion	Funktionalität
init	XPRBinit	Globale Initialisierung und Lizenzcheck.
license		Angaben des Pfads zum Lizenzfile.
setColOrder	XPRBsetcolorder	Spaltensortierung (alphabetisch, normal).
setMsgLevel	XPRBsetmsglevel	Setzt message level.
setRealFmt	XPRBsetrealfmt	Setzt Format für Dezimalzahlen.
getVersion	XPRBgetversion	Liefert Versionsinformation.
getTime	XPRBgettime	Liefert Systemzeit.

XPRBprob		
Methode	Aufruf C-Funktion	Funktionalität
XPRBprob	XPRBnewprob	Konstruktor für neues Modell.
addCuts	XPRBaddcuts	Fügt mehrere Cuts hinzu.
clearDir	XPRBcleardir	Löscht branching directives.
delCtr	XPRBdelctr	Löscht Restriktion.
delCut	XPRBdelcut	Löscht Cut.
delSos	XPRBdelsos	Löscht SOS.
exportProb	XPRBexportprob	Schreibt Modell in Datei.
getCtrByName	XPRBgetbyname	Liefert Referenz auf Restriktion von gegebenem Namen.
getIIS	XPRBgetiis	Liefert ein IIS.
getIndexSetByName	XPRBgetbyname	Liefert Referenz auf Indexset von gegebenem Namen.
getLPStat	XPRBgetlpstat	Liefert LP Status.
getMIPStat	XPRBgetmipstat	Liefert MIP Status.
getName	XPRBgetprobname	Liefert Modellnamen.
getNumIIS	XPRBgetnumiis	Liefert die Anzahl der IIS.
getObjVal	XPRBgetobjval	Liefert Zielfunktionswert.
getProbStat	XPRBgetprobstat	Liefert Modellstatus.
getSense	XPRBgetsense	Liefert Optimierungsrichtung.
getSosByName	XPRBgetbyname	Liefert Referenz auf SOS von gegebenem Namen.
getVarByName	XPRBgetbyname	Liefert Referenz auf Variable von gegebenem Namen.
getXPRSprob	XPRBgetXPRSprob	Liefert Pointer auf Optimizer-Modell.
loadBasis	XPRBloadbasis	Lädt Basis.
loadMat	XPRBloadmat	Überträgt Modell in XPRESS-Optimizer.
maxim	XPRBmaxim	Startet Maximierung.
minim	XPRBminim	Startet Minimierung.
newCtr	XPRBnewctr	Erzeugt neue Restriktion und liefert Referenz darauf.
newCut	XPRBnewcut	Erzeugt neuen Cut und liefert Referenz darauf.
newIndexSet	XPRBnewidxset	Erzeugt neues Indexset und liefert Referenz darauf.
newSos	XPRBnewsos	Erzeugt neues SOS und liefert Referenz darauf.
newVar	XPRBnewvar	Erzeugt neue Variable und liefert Referenz darauf.
print	XPRBprintprob	Ausgabe des Modells.
reset	XPRBresetprob	Modell zurücksetzen.
saveBasis	XPRBsavebasis	Speichert Basis.
setColOrder	XPRBsetcolorder	Setzt Spaltenabfolge (alphabetisch, normal).
setCutMode	XPRBsetcutmode	Setzt Cut-Modus.
setDictionarySize	XPRBsetdictionarysize	Setzt Größe der Hashtabelle für alle Namen.
setMsgLevel	XPRBsetmsglevel	Setzt Message Level.
setObj	XPRBsetobj	Setzt Restriktion als Zielfunktion.
setRealFmt	XPRBsetrealfmt	Setzt Ausgabeformat für Dezimalzahlen.
setSense	XPRBsetsense	Setzt Optimierungsrichtung.
solve	XPRBsolve	Startet Optimierungslauf.
sync	XPRBsync	Synchronisiert BCL mit dem Solver.
writeDir	XPRBwritedir	Schreibt directives in Datei.

XPRBvar		
Methode	Aufruf C-Funktion	Funktionalität
XPRBvar		Konstruktor für neue Variable.

fix	XPRBfixvar	Fixiert Variable auf einen Wert.
getBounds	XPRBgetbounds	Liefert Ober- und Untergrenze.
getColNum	XPRBgetcolnum	Liefert Spaltennummer der Matrix.
getLB	XPRBgetbounds	Liefert Untergrenze.
getLim	XPRBgetlim	Liefert Limit für Partial-Integer oder Semi-Continuous Variable.
getName	XPRBgetvarname	Liefert Variablennamen.
getRCost	XPRBgetrcost	Liefert reduzierte Kosten.
getRNG	XPRBgetvarrng	Liefert Ranging Information für die Variable.
getSol	XPRBgetsol	Liefert Lösungswert.
getType	XPRBgetvartype	Liefert Variablentyp.
getUB	XPRBgetbounds	Liefert Obergrenze.
isValid		Prüft Validität.
print	XPRBprintvar	Ausgabe Variable.
setDir	XPRBsetvardir	Setzt Branching Directions.
setLB	XPRBsetlb	Setzt Untergrenze.
setLim	XPRBsetlim	Setzt Limit für Partial-Integer oder Semi-Continuous Variable.
setType	XPRBsetvartype	Setzt Variablentyp.
setUB	XPRBsetub	Setzt Obergrenze.

XPRBctr		
Methode	Aufruf C-Funktion	Funktionalität
XPRBctr		Konstruktor für neue Restriktion.
addTerm	XPRBaddterm	Term hinzufügen.
delTerm	XPRBdelterm	Term löschen.
getAct	XPRBgetact	Liefert Activity.
getDual	XPRBgetdual	Liefert Dualwert.
getName	XPRBgetctrname	Liefert Name.
getRange	XPRBgetrange	Liefert Ober- und Untergrenze einer Range-Restriktion.
getRangeL	XPRBgetrange	Liefert Untergrenze einer Range-Restriktion.
getRangeU	XPRBgetrange	Liefert Obergrenze einer Range-Restriktion.
getRHS	XPRBgetrhs	Liefert RHS.
getRNG	XPRBgetctrrng	Liefert Range-Informationen.
getRowNum	XPRBgetrownum	Liefert Zeilenindex der Matrix.
getSlack	XPRBgetslack	Liefert Slack.
getType	XPRBgetctrtype	Liefert Restriktionstyp (<=, =, =>).
isModCut	XPRBgetmodcut	Cut oder normale Restriktion.
isValid		Prüft Validität.
print	XPRBprintctr	Ausgabe Restriktion.
setModCut	XPRBsetmodcut	Festlegen ob Cut oder normale Restriktion.
setRange	XPRBsetrange	Setzt Ober- und Untergrenze für Range.
setTerm	XPRBsetterm	Setzt Term.
setType	XPRBsetctrtype	Setzt Restriktionstyp.
Operator =		Zuweisung linearer Ausdruck.
Operator +=		Hinzufügen linearer Ausdruck.
Operator -=		Hinzufügen negativer linearer Ausdruck.

XPRBlinExp		
Methode	Aufruf C-Funktion	Funktionalität
XPRBlinExp		Konstruktor für neuen linearen Ausdruck.
add		Fügt Variablen, Koeffizienten und ganze lineare Ausdrücke hinzu.
getSol()		Liefert Lösungswert durch Einsetzen der Variablenwerte.
neg		Kehrt Vorzeichen um.
setTerm		Fügt Term hinzu.
Operator =		Weist dem linearen Ausdruck einen anderen lin. Ausdruck zu.
Operator +=		Fügt linearen Ausdruck hinzu.
Operator -=		Fügt negativen linearen Ausdruck hinzu.

Weitere Möglichkeiten zur Verwendung linearer Ausrücke (Definition außerhalb der Klasse `XPRBlinExp`):

```
- linexp
linexp1 + linexp2
linexp1 - linexp2
```

```
linexp * val
val * linexp
var * val
val * var
- var
```

XPRBindexSet		
Methode	Aufruf C-Funktion	Funktionalität
XPRBindexSet		Konstruktor für neues Indexset.
addElement	XPRBaddidxel	Fügt Element hinzu.
getIndex	XPRBgetidxel	Liefert Index eines Elements.
getIndexName	XPRBgetidxelname	Liefert Name eines Elements.
getName	XPRBgetidxsetname	Liefert Namen.
getSize	XPRBgetidxsetname	Liefert Größe.
isValid		Prüft Validität.
print	XPRBprintidxset	Ausgabe des Indexsets.
Operator +=		Fügt Element dem Indexset hinzu.
Operator []		Zugriff auf Element via Index oder Name.

XPRBsos		
Methode	Aufruf C-Funktion	Funktionalität
XPRBsos		Konstruktor für neues SOS.
addElement	XPRBaddsosel	Hinzufügen gewichtete Variable.
delElement	XPRBdelsosel	Löscht Variables aus Set.
getName	XPRBgetsosname	Liefert Namen.
getType	XPRBgetsostype	Liefert SOS-Typ.
isValid		Prüft Validität.
print	XPRBprintsos	Ausgabe des SOS.
setDir	XPRBsetsosdir	Branching Directive setzen.
Operator =		Zuweisung linearer Ausdruck.
Operator +=		Hinzufügung linearer Ausdruck.

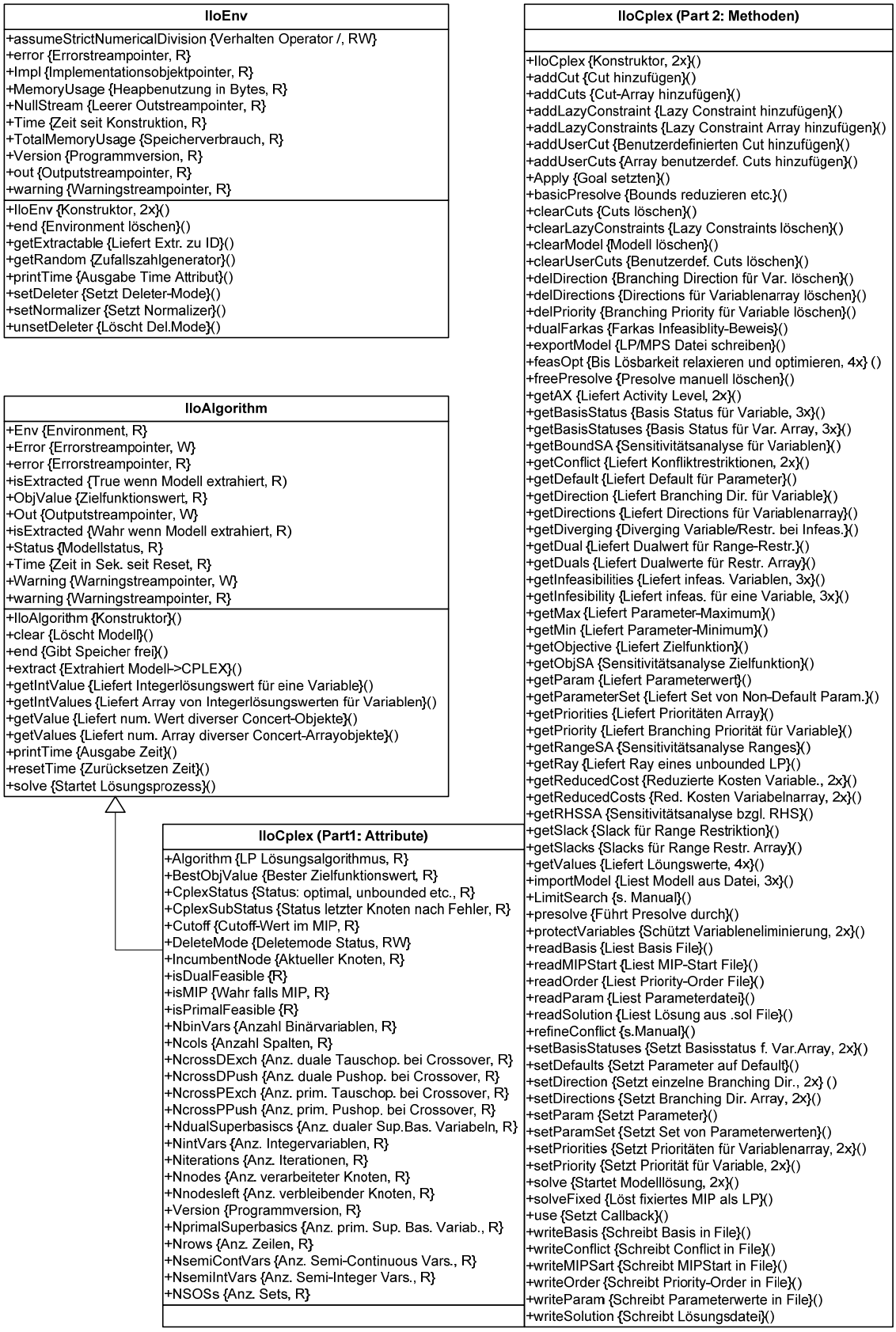
XPRBcut		
Methode	Aufruf C-Funktion	Funktionalität
XPRBcut		Konstruktor für neuen Cut.
add		Fügt linearen Ausdruck hinzu.
addTerm	XPRBaddcutterm	Fügt Term zum Cut hinzu.
delTerm	XPRBdelcutterm	Löscht Term.
getID	XPRBgetcutid	Liefert ID
getRHS	XPRBgetcutrhs	Liefert RHS
getType	XPRBgetcuttype	Liefert Cuttyp (<=, =, =>).
isValid		Prüft Validität.
print	XPRBprintcut	Ausgabe des Cuts.
setID	XPRBsetcutid	Setzt ID.
setTerm	XPRBsetcutterm	Setzt Term.
setType	XPRBsetcuttype	Setzt Cuttyp (<=, =, =>).
Operator =		Zuweisung linearer Ausdruck.
Operator +=		Hinzufügen linearer Ausdruck.
Operator -=		Hinzufügen negativer linearer Ausdruck.

XPRBbasis		
Methode	Aufruf C-Funktion	Funktionalität
XPRBbasis		Konstruktor für neue Basis.
isValid		Prüft Validität.

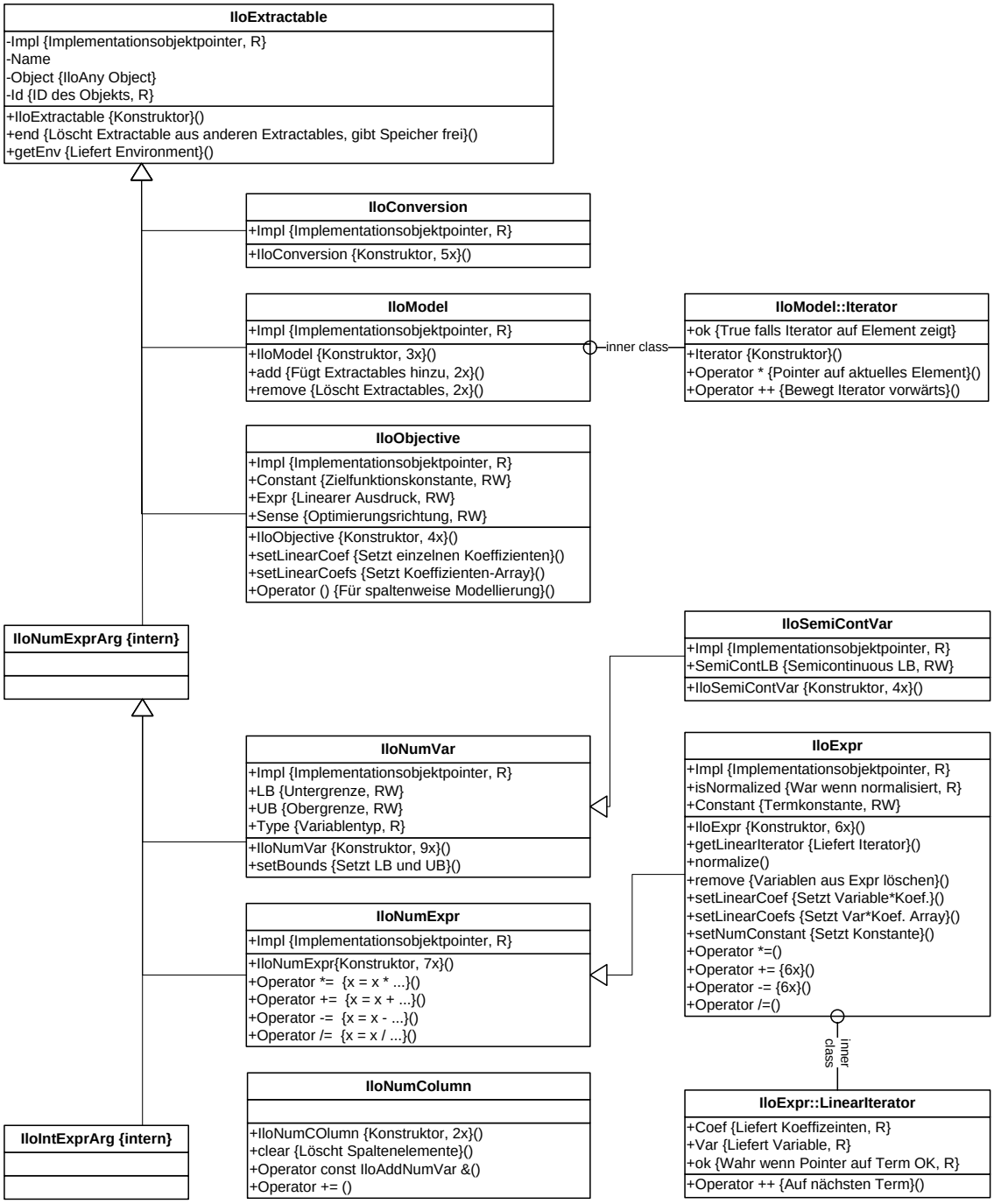
XPRBerror		
Methode	Aufruf C-Funktion	Funktionalität
getMessage		Liefert Fehlermeldung.
getErrorCode		liefert Fehlercode.

Anhang 3: Klassenübersicht ILOG Concert Technology

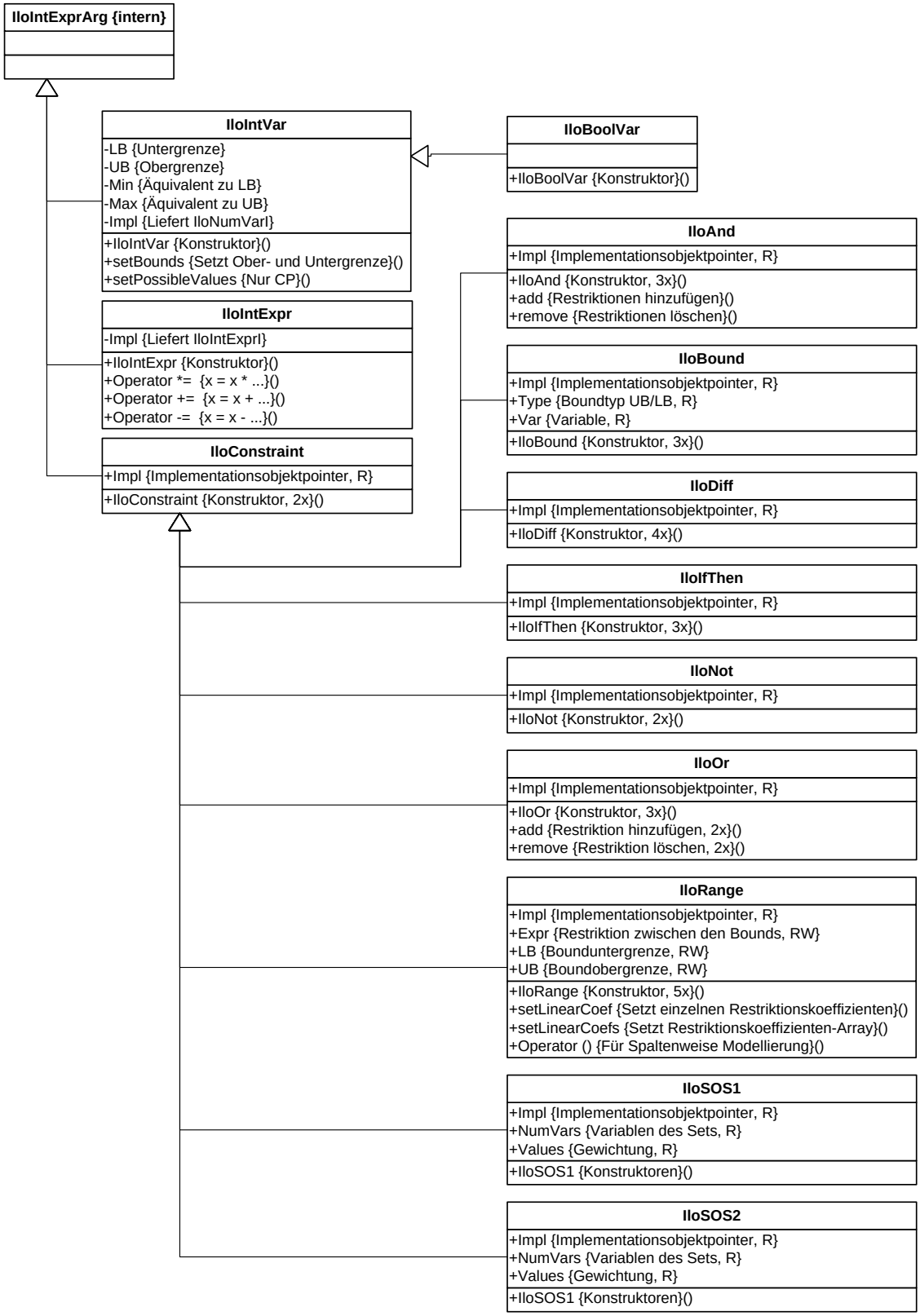
Environment, Algorithm, Cplex:



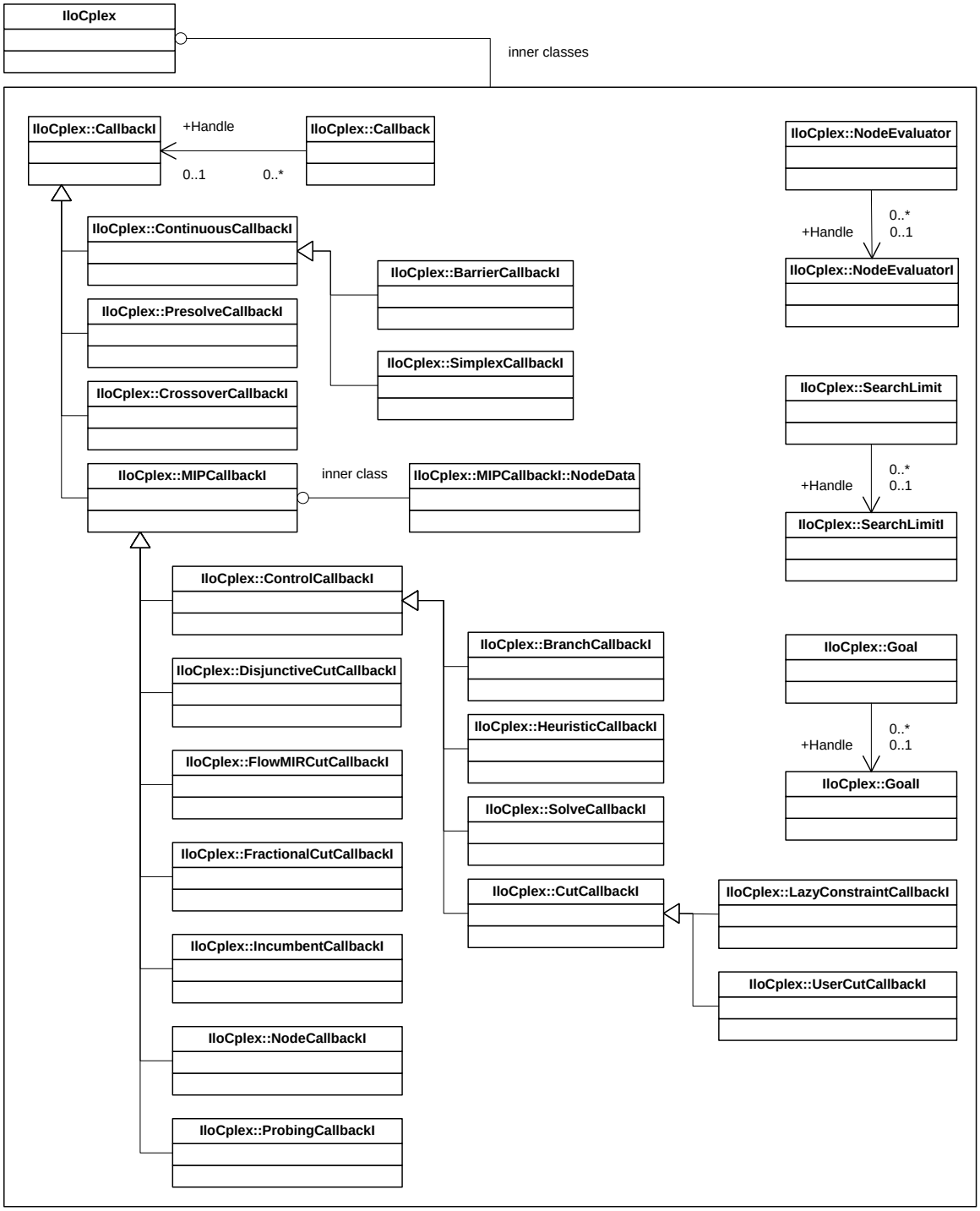
Extractables und abgeleitete Klassen, Teil 1:



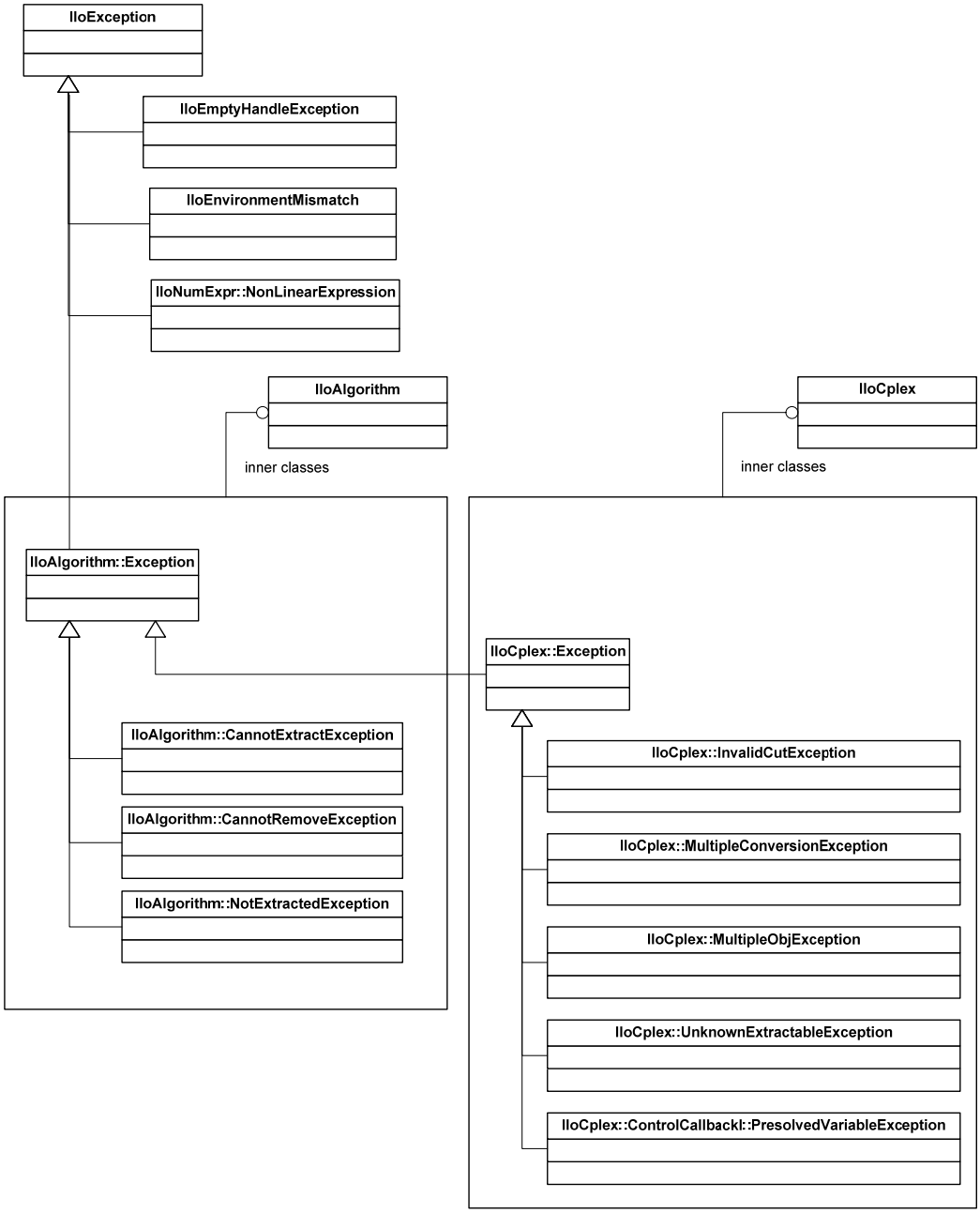
Extractables und abgeleitete Klassen, Teil 2:



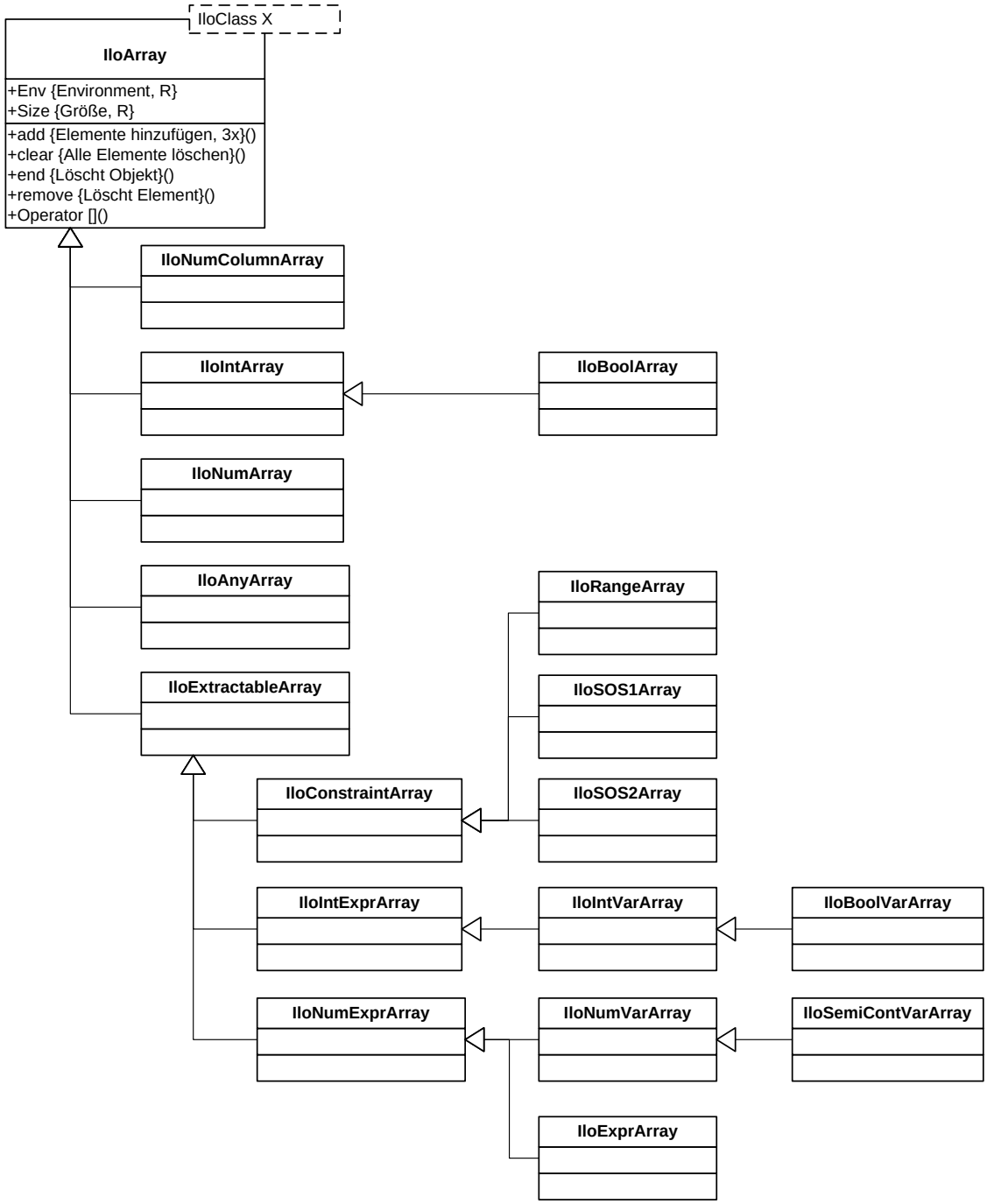
Callbacks:


IloCplex
inner classes
IloCplex::CallbackI
+Handle
0..1
0..*
IloCplex::Callback
IloCplex::NodeEvaluator
0..*
0..1
+Handle
IloCplex::NodeEvaluatorI
IloCplex::ContinuousCallbackI
IloCplex::BarrierCallbackI
IloCplex::PresolveCallbackI
IloCplex::SimplexCallbackI
IloCplex::CrossoverCallbackI
IloCplex::SearchLimit
0..*
0..1
+Handle
IloCplex::SearchLimitI
IloCplex::MIPCallbackI
inner class
IloCplex::MIPCallbackI::NodeData
IloCplex::ControlCallbackI
IloCplex::BranchCallbackI
IloCplex::Goal
IloCplex::DisjunctiveCutCallbackI
IloCplex::HeuristicCallbackI
0..*
0..1
+Handle
IloCplex::GoalI
IloCplex::FlowMIRCutCallbackI
IloCplex::SolveCallbackI
IloCplex::FractionalCutCallbackI
IloCplex::CutCallbackI
IloCplex::LazyConstraintCallbackI
IloCplex::IncumbentCallbackI
IloCplex::UserCutCallbackI
IloCplex::NodeCallbackI
IloCplex::ProbingCallbackI

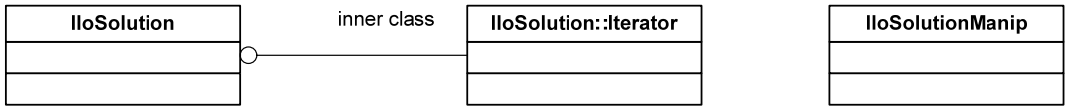
Exceptions:


IloException
IloEmptyHandleException
IloEnvironmentMismatch
IloNumExpr::NonLinearExpression
IloAlgorithm
inner classes
IloCplex
inner classes
IloAlgorithm::Exception
IloCplex::Exception
IloAlgorithm::CannotExtractException
IloAlgorithm::CannotRemoveException
IloAlgorithm::NotExtractedException
IloCplex::InvalidCutException
IloCplex::MultipleConversionException
IloCplex::MultipleObjException
IloCplex::UnknownExtractableException
IloCplex::ControlCallbackI::PresolvedVariableException

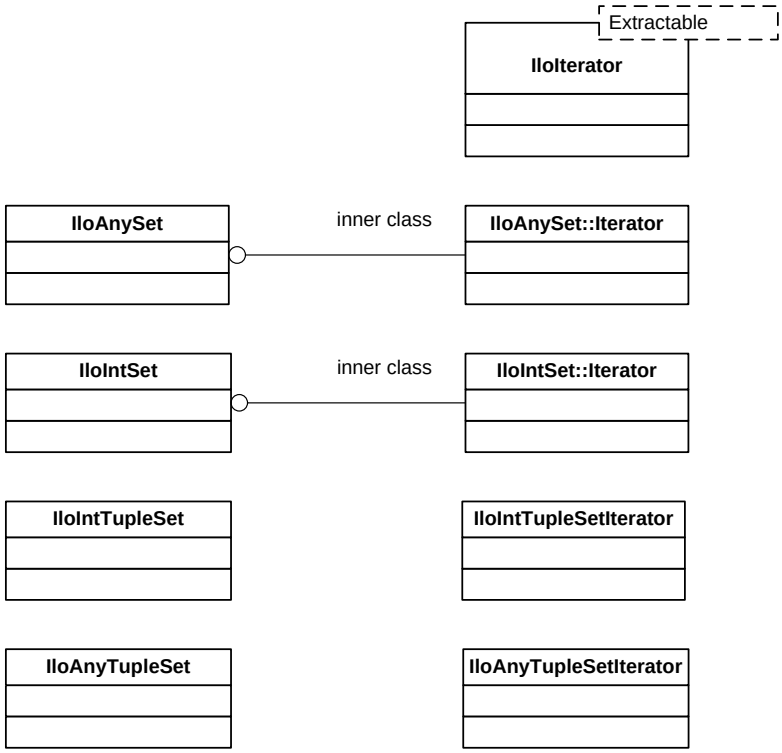
Arrays:


IloClass X
IloArray
+Env {Environment, R}
+Size {Größe, R}
+add {Elemente hinzufügen, 3x}()
+clear {Alle Elemente löschen}()
+end {Löscht Objekt}()
+remove {Löscht Element}()
+Operator []()
IloNumColumnArray
IloIntArray
IloBoolArray
IloNumArray
IloAnyArray
IloRangeArray
IloExtractableArray
IloSOS1Array
IloConstraintArray
IloSOS2Array
IloIntExprArray
IloIntVarArray
IloBoolVarArray
IloNumExprArray
IloNumVarArray
IloSemiContVarArray
IloExprArray

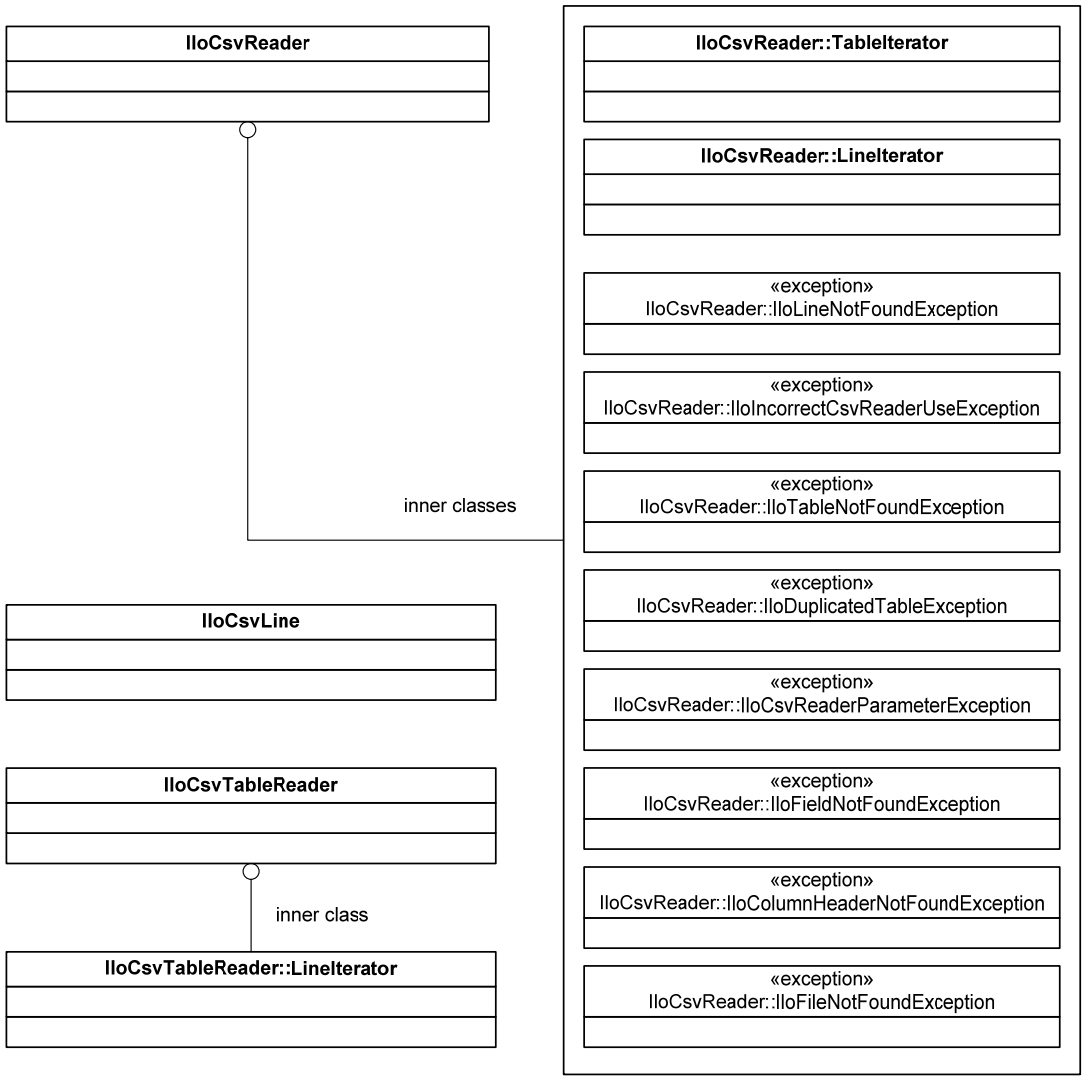
Lösung:


IloSolution
inner class
IloSolution::Iterator
IloSolutionManip

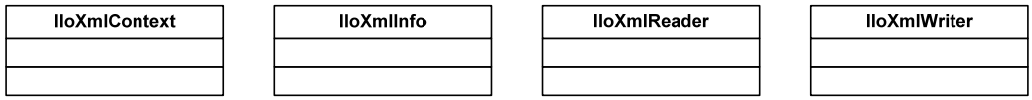
Sets:


Extractable
IloIterator
IloAnySet
inner class
IloAnySet::Iterator
IloIntSet
inner class
IloIntSet::Iterator
IloIntTupleSet
IloIntTupleSetIterator
IloAnyTupleSet
IloAnyTupleSetIterator

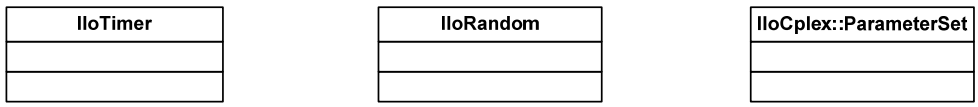
CSV- Unterstützung:



XML-Unterstützung:

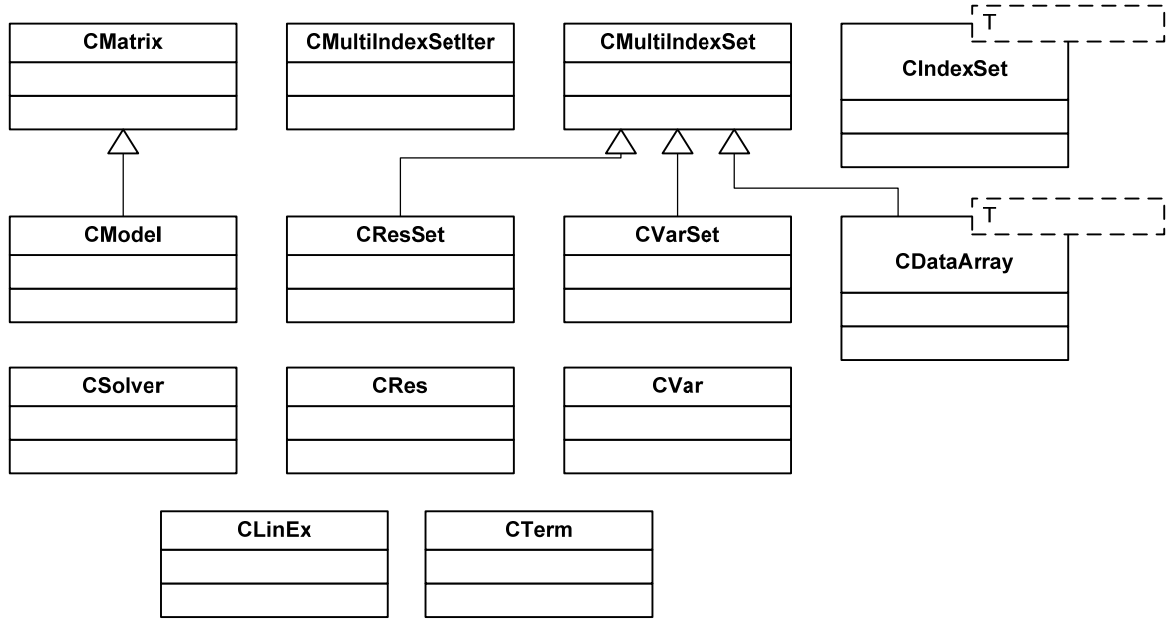


Diverse:



Anhang 4: Gemeinsame Modellierungsklassen

Klassen in der Übersicht:



Klassen im Detail[1]:

CMatrix
Name:String

Modell-Handling
Delete()
CloneFrom(Matrix:CMatrix)
SubmitModel(Solver:CSolver)
GetSolution(Solver:CSolver)
File I/O
ReadModel(Filename,Format)
ReadAMPLModel(MODFilename,DATFilename1,…) // Ebenso für andere Sprachen
WriteModel(Filename,Format)
WriteSolFile(Filename)
Modellaufbau und -modifikation
AddCol(UB,LB,Cost,Typ,Name)
AddCols(n,UBs,LBs,Costs,Typs,Names)
InsCol(j,UB,LB,Cost,Typ,Name)
InsCols(j,n,UBs,LBs,Costs,Typs,Names)
DelCol(j)
DelCols(Startj,Endj)
AddRow(LHS,RHS,Name)
AddRows(n,LHSs,RHSs,Names)
InsRow(i,LHS,RHS,Name)
InsRows(i,n,LHSs,RHSs,Names)
DelRow(i)
DelRows(Starti,Endi)
PutNonzero(j,i,Value)
PutNonzeros(n,ColIndices,RowIndices,Values)
ChColLB(j,LB)
ChColLBs(Startj,Endj,LBs)
ChColUB(j,UB)
ChColUBs(Startj,Endj,UBs)
ChColCost(j,Cost)
ChColCosts(Startj,Endj,Costs)

[1] In der UML-ähnlichen Pseudocodenotation wurden Typangaben für Argumente weggelassen falls diese offensichtlich sind. Private Methoden und Eigenschaften sind ebenfalls ausgelassen.

```
ChColType(j,Type)
ChColTypes(Startj,Endj,Types)
ChColName(j,Name)
ChColNames(Startj,Endj,Names)
ChRowRHS(i,RHS)
ChRowRHSs(Starti,Endi,RHSs)
ChRowLHS(i,LHS)
ChRowLHSs(Starti,Endi,LHSs)
ChRowName(i,Name)
ChRowNames(Starti,Endi,Names)
AddSOS(Type,n,ColIndices,Order,{out}SOSID)
DelSOS(SOSID)
PutModel(m,n,nz,ColLBs,ColUBs,Costs,ColTypes,ColNames,
         LHSs,RHSs,RowNames,ColIndices,RowIndices,Nonzeros)
```

Modellabfrage

```
GetColLB(j,{out}LB)
GetColLBs(Startj,Endj,{out}LBs)
GetColUB(j,{out}UB)
GetColUBs(Startj,Endj,{out}UBs)
GetColType(j,{out}Type)
GetColTypes(Startj,Endj,{out}Types)
GetColCost(j,{out}Cost)
GetColCosts(Startj,Endj,{out}Costs)
GetColName(j,{out}Name)
GetColNames(Startj,Endj,{out}Names)
GetRowLHS(i,{out}LHS)
GetRowLHSs(Starti,Endi,{out}LHSs)
GetRowRHS(i,{out}RHS)
GetRowRHSs(Starti,Endi,{out}EHSs)
GetRowName(i,{out}Name)
GetRowNames(Starti,Endi,{out}Names)
GetNonzero(j,i,{out}Value)
GetNonzeros(Colindices,RowIndices,{out}Values)
GetColNonzeros(j,{out}RowIndices,{out}Values)
GetRowNonzeros(i,{out}ColIndices,{out}Values)
GetSOSnElements(SOSID,{out}n)
GetSOS(SOSID,{out}ColIndices,{out}Order)
GetColIndex(ColName,{out}j)
GetColIndices(n,ColNames,{out}ColIndices)
GetRowIndex(RowName,{out}i)
GetRowIndices(n,RowNames,{out}RowIndices)
GetModel({out}m,{out}n,{out}nz,
         {out}ColLBs,{out}ColUBs,{out}Costs,{out}ColTypes,
         {out}ColNames,{out}LHSs,{out}RHSs,{out}RowNames,
         {out}ColIndices,{out}RowIndices,{out}Nonzeros)
```

Lösungsabfrage

```
GetLPSol(j,{out}Value)
GetLPSols(Startj,Endj,{out}Values)
GetIPSol(j,{out}Value)
GetIPSols(Startj,Endj,{out}Values)
GetRedCost(j,{out}Value)
GetRedCosts(Startj,Endj,{out}Values)
GetBaseStat(j,{out}Value)
GetbaseStats(Startj,Endj,{out}Values)
GetActivity(i,{out}Value)
GetActivities(Starti,Endi,{out}Values)
GetDual(i,{out}Value)
GetDuals Starti,Endi,{out}Values)
GetSlack(i,{out}Value)
GetSlacks Starti,Endi,{out}Values)
GetRowBaseStat(i,{out}Value)
GetRowBaseStats(Starti,Endi,{out}Values)
```

Parameterhandling

```
SetParameter(ParameterID,Value)        //Überladen für dbl, int, string
GetParameter(ParameterID,{out}Value) //Überladen für dbl, int, string
SetDefaults()
GetParameterInfo(ParameterID,{out}Min,
                 {out}Max,{out}Default) //Überladen für dbl, string
GetParametername(ParameterID,{out}ParameterName)
```

```
GetParameterID(ParameterName,{out}ParameterID)
```

CModel

```
NewVar(Name,Type,LB,UB,Cost):CVar    // Pararamter sind optional
NewVarSet():CVarSet
NewVarSet(NumberOfIndexSets):CVarSet
NewVarSet(IndexSet1:CIndexSet,IndexSet2:CIndexSet,…):CVarSet
NewVarSet(MultiIndexSet:CMultiIndexset):CVarSet

NewRes(Name,LB,UB):CRes       // Pararamter sind optional
NewResSet():CResSet
NewResSet(NumberOfIndexSets):CResSet
NewResSet(IndexSet1:CIndexSet,IndexSet2:CIndexSet,…):CResSet
NewResSet(MultiIndexSet:CMultiIndexSet):CResSet

AddVar(Var:CVar)
AddVarSet(VarSet:CVarSet)
AddRes(Res:CRes)
AddResSet(ResSet:CResSet)

AddObj(Var:CVar)
AddObj(Var:CVar,Coef:dbl)
AddObj(Term:CTerm)
AddObj(LinEx:CLinEx)
AddObj(VarSet:CVarSet)
AddObj(VarSet:CVarSet,Coefs:CDataArray)
DelObj(Var:CVar)
DelObj(VarSet:CVarSet)

// Weiterhin Iteratoren für alle Variablen, Restriktionen und Sets
```

CSolver

```
CSolver(Solvertyp:int)       // Konstruktor
```

File I/O

```
ReadParamFile(Filename)
WriteParamFile(Filename)
```

Lösungsprozesssteuerung

```
Optimize()
```

Parameterhandling

```
SetParameter(ParameterID,Value)          //Überladen für dbl, int, string
GetParameter(ParameterID,{out}Value) //Überladen für dbl, int, string
SetDefaults()
GetParameterInfo(ParameterID,{out}Min,
                 {out}Max,{out}Default) //Überladen für dbl, string
GetParametername(ParameterID,{out}ParameterName)
GetParameterID(ParameterName,{out}ParameterID)
```

T

CIndexSet

```
Size:int
Name:string
```

Konstruktoren

```
CIndexSet()
CIndexSet(Element1:T, Element2:T…)
```

Aufbau und Modifikation

```
Clear()
Add(Element1:T, Element2:T)
AddSeries(Start:dbl,End:dbl,Step:dbl)
InsertBeforeIndex(i,Element:T)
InsertBeforeElement(Element:T,Element:T)
DelAt(i:int)
```

```
Del(Element:T)
Adddition(Value:dbl)
Multiplication(Value:dbl)
SetAllowDuplicates(Allow:bool)
```
Mengenmethoden
```
CloneFrom(Source:CIndexSet)
Union(Source:CIndexSet)
Diff(Source:CIndexSet)
Intersect(Source:CIndexSet)
```
Zugriffsmethoden
```
GetIndex(Element:T,{out}i:int)
GetElement(i:int,{out}Element:T)
Contains(Element:T):bool
```
Iteratormethoden
```
MoveFirst()
MoveNext()
GetCurrentIndex():int
GetCurrentElement():T
IsLast()
IsBeyondLast()
```

CMultiIndexSet

```
Name: string
NumberOfIndexSets: int
TotalNumberOfElements:int
```
Konstruktoren
```
CMultiIndexSet()
CMultiIndexSet(NumberOfIndexSets:int)
CMultiIndexSet(MultiIndexSet:CMultiIndexSet)
CMultiIndexSet(IndexSet1:CIndexSet,IndexSet2:CIndexSet,…)
```
Zugriff auf einzelne Indexsets
```
SetNumberOfIndexSets(n:int)
SetIndexSet(i:int,IndexSet:CIndexSet)
GetIndexSet(i:int,{out}IndexSet:CIndexSet)
SetIndexSets(MultiIndexSet:CMultiIndexSet)
GetIndexSets({out}MultiIndexSet:CMultiIndexSet)
SetIndexSets(IndexSet1:CIndexSet,IndexSet2:CIndexSet,…)
GetIndexSets({out}IndexSet1:CIndexSet,{out}IndexSet2:CIndexSet,…)
```
Zugriff auf Elemente einzelner Sets
```
GetSizeOfIndexSet(i:int,{out}Size:int)
GetElementByNumber(IndexNoi:int,Position:int,
                   {out}Element:string)     // Überladen für int und dbl
```
Filter
```
SetIterFilter(IndexSet1,IndexSet2,…)
GetIterFilter({out}IndexSet1,{out}IndexSet2,…)
ClearIterFilter()
```
Iterator
```
MoveFirst()
MoveNext()
GetCurrentFullIndex({out}IndexName:string)
GetCurrentElements({out}Element1:string,
                   {out}Element2:string, …) // Überladen für int und dbl
GetCurrentTuple():CTuple
IsLast()
IsBeyondLast()
```

CMultiIndexSetIter

```
NumberOfIndexSets:int
```
Konstruktoren
```
CMultiIndexSetIter()
CMultiIndexSetIter(IndexSet:CIndexSet)
CMultiIndexSetIter(MultiIndexSet:CMultiIndexSet)
```
Initialisierung
```
Init(IndexSet:CIndexSet)
Init(MultiIndexSet:CMultiIndexSet)
```

Iterator
```
MoveFirst()
MoveNext()
GetCurrentFullIndex({out}IndexName:string)
GetCurrentElements({out}Element1:string,
                   {out}Element2:string, …) // Überladen für int und dbl
GetCurrentTuple({out}Tuple:CTuple)
IsLast()
IsBeyondLast()
```

CTuple

```
NumberOfIndexSets: int
```

Konstruktoren
```
CTuple()
CTuple(NumberOfElements:int)
CTuple(Index1:string,Index2:string…)  // Überladen für int und dbl
```
Initialisierung
```
Init(NumberOfElements:int)
Init(Index1:string,Index2:string…)      // Überladen für int und dbl
```
Zugriff auf Elemente
```
SetElement(i:int,Element:string)      // Überladen für int und dbl
GetElement(i:int,Element:string)      // Überladen für int und dbl
```

CDataArray <T>

Konstruktoren
```
CDataArray()
CDataArray(NumberOfIndexSets:int)
CDataArray(IndexSet1:CIndexSet,IndexSet2:CIndexSet,…)
CDataArray(MultiIndexSet:CMultiIndexSet)
```
Zugriff auf Elemente
```
SetElementAt(Element:T,Index1:string,
                       Index2:string…)        // Überladen für int und dbl
SetElementAt(Element:T,Tuple:CTuple)
GetElementAt({out}Element:T,Index1:string,
                            Index2:string…) // Überladen für int und dbl
GetElementAt({out}Element:T,Tuple:CTuple)
// Setzt Element an allen Positionen, die durch den Indexset-Filter bezeichnet werden
SetElements(Element, IndexSet1,IndexSet2,…)
```
Iterator
```
GetCurrentField():T      // Aktuell iteriertes Feld
                         // Weitere Iteratormethoden von Elternklasse CMultiIndexSet
```

CVar

```
Name:string
LB:dbl
UB:dbl
Cost:dbl
Type:int
LPSol:dbl
IPSol:dbl
RedCost:dbl
Status:int
ColIndex :int
ParentModel:CModel
IsAttached:bool
```

Konstruktoren
```
CVar()
CVar(Model:CModel)                  // Add
CVar(Model:CModel,ColIndex:int)     // Attach
```
Zufügung zu Modell
```
AddToModel(Model:CModel)
AttachToModel(Model:CModel,ColIndex:int)
```

AttachToModel(Model:CModel,ColName:string)

CRes
Name:string LHS:dbl RHS:dbl Activity:dbl Dual:dbl Slack:dbl Status:int RowIndex:int ParentModel:CModel IsAttached:bool

Konstruktoren
CRes()
CRes(Model:CModel) //Add
CRes(Model:CModel,RowIndex:int) //Attach
Zufügung zu Modell
AddToModel(Model:CModel)
AttachToModel(Model:CModel,RowIndex:int)
AttachToModel(Model:CModel,RowName:string)
Hinzufügung Variablen und Ausdrücke
AddVar(Var:CVar)
AddVar(Var:CVar,Koef:dbl)
AddTerm(Term:CTerm)
AddLinEx(LinEx:CLinEx)
Löschen Variablen und Ausdrücke
DelVar(Var:CVar)
DelTerm(Term:CTerm)
DelLinEx(LinEx:CLinEx)
Iterator
MoveFirstVar()
MoveNextVar()
GetCurrentVar():CVar
IsLastVar()
IsBeyondLastVar()

CVarSet
Name:string ColIndexStart:int ColIndexEnd:int ParentModel:CModel

Konstruktoren
CVarSet(Model:CModel)
CVarSet(Model:CModel,NumberOfIndexSets:int)
CVarSet(Model:CModel,IndexSet1:CIndexSet,IndexSet2:CIndexSet,…)
CVarSet(Model:CModel,MultiIndexSet:CMultiIndexSet)
Zugriff auf Variablenobjekte
SetVarAt(Var:CVar,Index1:string,Index2:string…) // Überladen für int, dbl
GetVarAt({out}Var:CVar,Index1:string,Index2:string…) // Überladen s.o
SetVarAt(Var:CVar,Tuple:CTuple)
GetVarAt({out}Var:CVar,Tuple:CTuple)
Setzen von Attributen
SetLB(LB:dbl)
SetLBs(LB:dbl,IndexSet1:CIndexSet,IndexSet2:CIndexSet,…)
SetLBs(LBs:CDataArray)
// *Ebenso für UB, Type, Cost*
Iterator
GetCurrentVar():CVar
// Weitere Iteratormethoden geerbt von CMultiIndexSet

CResSet
Name:string

```
RowIndexStart:int
RowIndexEnd:int
ParentModel:CModel
```

Konstruktoren

`CResSet(Model:CModel)`

`CResSet(Model:CModel,NumberOfIndexSets:int)`

`CResSet(Model:CModel,IndexSet1:CIndexSet,IndexSet2:CIndexSet,…)`

`CResSet(Model:CModel,MultiIndexSet:CMultiIndexSet)`

Zugriff auf Restriktionsobjekte

```
SetResAt(Res:CRes,Index1:string,Index2:string…) // Überladen für int, dbl
GetResAt({out}Res:CRes,Index1:string,Index2:string…) // Überladen s.o.
SetResAt(Res:CRes,Tuple:CTuple)
GetResAt({out}Res:CRes,Tuple:CTuple)
```

Setzen von Attributen

```
SetLHS(LHS:dbl)
SetLHSs(LHS:dbl,IndexSet1:CIndexSet,IndexSet2:CIndexSet,…)
SetLHSs(LHSs:CDataArray)
// Ebenso für RHS
```

Hinzufügung Variablen und Ausdrücke

```
Add(Var:CVar)
Add(Var:CVar,Koef:dbl)
Add(Term:CTerm)
Add(LinEx:CLinEx)
// Add… ebenso mit Spezifikation des CTuple für eine einzelne Restriktion
```

Löschen Variablen und Ausdrücke

```
Del(Var:CVar)
Del(Term:CTerm)
Del(LinEx:CLinEx)
// Del… ebenso mit Spezifikation des CTuple für eine einzelne Restriktion
```

Iterator

```
GetCurrentRes():CRes
// Weitere Iteratormethoden geerbt von CMultiIndexSet
```

CLinEx

Konstruktoren

`CLinEx(…)` // Konstruktor hat gleiche Überladungen wie Add

Hinzufügen und Löschen Variablen und Ausdrücke

```
Add(Var:CVar)
Add(Var:CVar,Koef:dbl)
Add(VarSet:CVarSet)
Add(VarSet:CVarSet,Koefs:CDataArray)
Add(Term:CTerm)
Add(LinEx:CLinEx)
Del(Var:CVar)
Del(VarSet:CVarSet)
Del(Term:CTerm)
Del(LinEx:CLinEx)
```

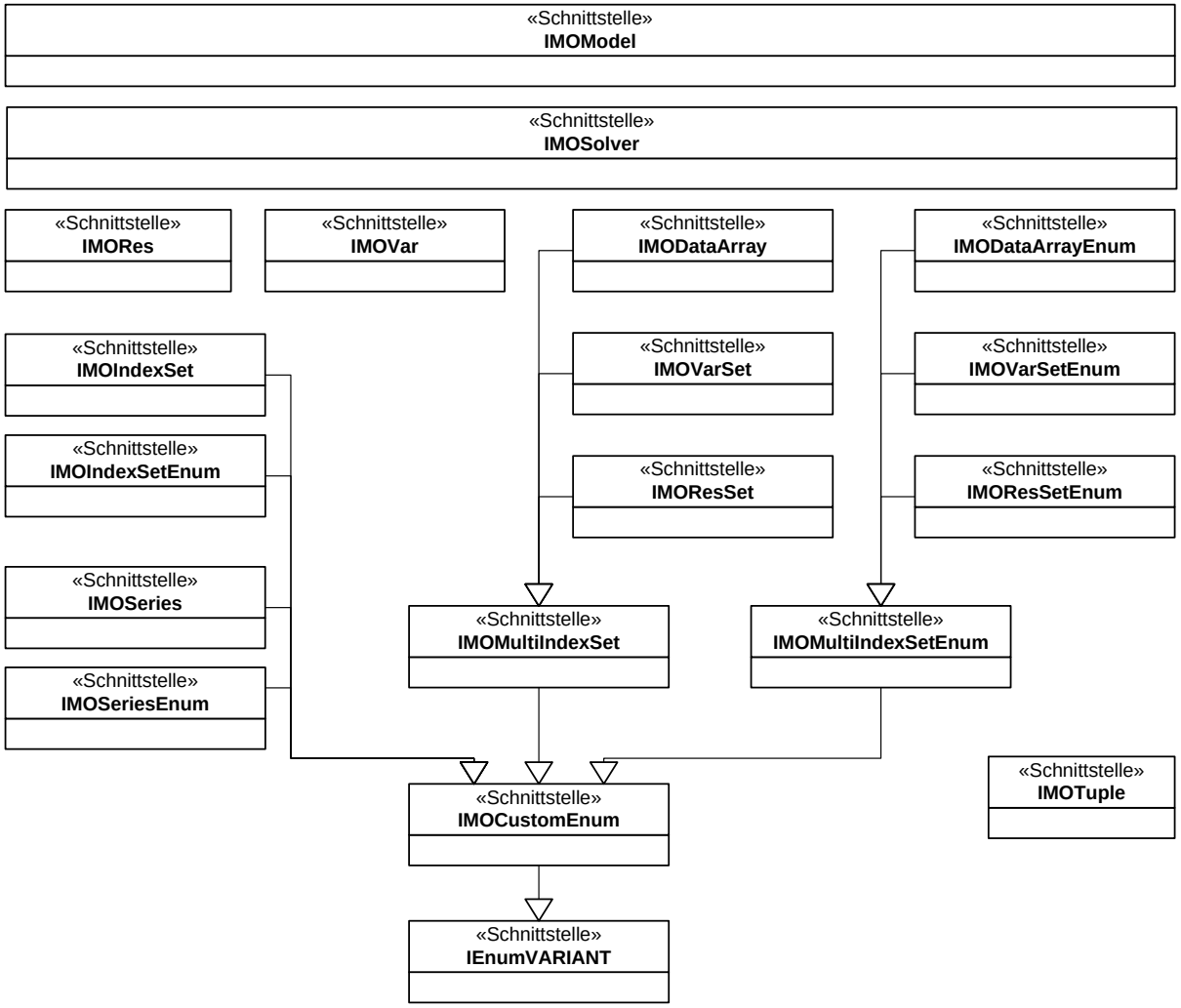
Iterator

```
MoveFirstVar()
MoveNextVar()
GetCurrentVar():CVar
GetCurrentKoef():dbl
IsLastVar()
IsBeyondLastVar()
```

CTerm

```
CTerm(Var:CVar,Koef:dbl)
SetVar(Var:CVar,Koef:dbl)
GetVar({out}Var:CVar,{out}Koef:dbl)
```

Anhang 5: Interfaces der COM-Bibliothek



Detailansicht Komponenten-Interfaces[1]

«interface» **IMOModel**
// Modelleigenschaften *IsIPModel:bool {r}* ModelName:string *{rw}* Name:string *{r}* // Interner Komponentenname *NumberOfCols:int {r}* *NumberOfNZ:int {r}* *NumberOfRows:int {r}* *NumberOfSOSElements(SOSID:int):int* Parameter(Name:string):variant {rw} **// Spalteneigenschaften**

[1] Die Implementation weicht aus Aufwandsgründen teilweise vom konzeptionellen Entwurf ab. Daher sind nicht alle aufgeführten Eigenschaften und Methoden im Prototyp umgesetzt; auch wurden MOModel und MOSolver im Prototyp zu einer Komponente zusammengefasst, da zunächst nur ein Solver angebunden wurde. Kursiv gedruckte Eigenschaften und Methoden sind im Konzept eigentlich nicht vorgesehen, wurden aber zum besseren Handling im Prototypen implementiert (z.B. für MOPS Studio).

```
ColCost(Col:variant):variant {rw}
ColIPSol(Col:variant):variant {r}
ColIndex(Col:string):int {r}
ColLB(Col:variant):variant {rw}
ColLPSol(Col:variant):variant {r}
ColName(ColIndex:int):string {rw}
ColRedCost(Col:variant):variant {r}
ColStatus(Col:variant):variant {r}
ColType(Col:variant):variant {rw}
ColUB(Col:variant):variant {rw}

// Zeileneigenschaften
RowDual(Row:variant):variant {r}
RowIPSol(Row:variant):variant {r}
RowIndex(Row:string):int {r}
RowLHS(Row:variant):variant {rw}
RowLPSol(Row:variant):variant {r}
RowName(RowIndex:int):string {rw}
RowStatus(Row:variant):variant {r}
RowRHS(Row:variant):variant {rw}

// Nonzeros
NZ(Row:variant,Col:variant):double {rw}
NZs(Rows:variant,Cols:variant):variant {rw}

// Lösungseigenschaften
IPObjFunctBestPossible:double {r}
IPObjFunctValue:double {r}
LPObjFunctValue:double {r}
LPSumInfeas:double {r}
NumberOfIPIntSolutions:int {r}
NumberOfIPNodesProcessed:int {r}
NumberOfLPInfeas:int {r}
NumberOfLPIterations:int {r}
SolutionStatus:enum {r}

// Object Factories
NewVar({LB:variant},{UB:variant},{Type:variant},
       {Cost:variant},{Name:variant}):IMOVar {r}
NewVarSet({Indexsets:variant}):IMOVarSet {r}
NewRes({LHS:variant},{RHS:variant},{Name:variant}):IMORes {r}
NewResSet({Indexsets:variant}):IMOResSet {r}
Res(Row:variant):IMORes {rw}     // Setzt oder liefert MORes Objekt
Var(Col:variant):IMOVar {rw}     // Setzt oder liefert MOVar Objekt

// Sonstige Eigenschaften
AMPLOutputBuffer:string {r}
AMPLOutputBufferMaxSize:int {rw}
INF:double {r}
LocationOfAMPLEXE:string {rw}
LocationOfMOAMPLEXE:string {rw}
LocationOfMOPSDLL:string {rw}
LocationOfMathProgDLL:string {rw}
MODLLVersion:string {r}
```

```
// Methoden Modelbuilding
AddCol(LB:dbl,UB:dbl,Type:enum,{Cost:variant},{Name:variant})
AddCols(n:int,UBs:variant,LBs:variant,
        Types:variant,{Costs:variant},{Names:variant})
AddNZ(Row:variant,Col:variant,NZ:dbl)
AddObj(Vars:variant[])
AddObjSum(CoefsAndVars:variant[])   // alternierend Koeffizienten und Variablen
AddRes(Res:IMORes)
AddResSet(ResSet:IMOResSet)
AddRow(LHS:dbl,RHS:dbl,{Name:variant})
AddRows(n:int,LHSs:variant,RHSs:variant,{Names:variant})
AddSOS(Type:enum,n:int,Cols:variant,Order:variant,{out}SOSID:int)
AddVar(Var:IMOVar)
AddVarSet(VarSet:IMOVarSet)
InsCol(j:variant,UB:dbl,LB:dbl,Type:enum,{Cost:variant},{Name:variant})
```

```
InsCols(j:variant,n:int,UBs:variant,LBs:variant,{Costs:variant},{Names:variant})
InsRow(i:variant,LHS:double,RHS:double,{Name:variant})
InsRows(i:variant,n:int,LHSs:variant,RHSs:variant,{Names:variant})
ObjFuncSum(CoeffsAndCols:variant[])

// Methoden Modellmodifikation
AttachRes(Res:IMORes,RowIndex:variant)
AttachVar(Var:IMOVar,ColIndex:variant)
CloneModelFrom(Model:IMOModel)
CloneParameters(SourceModel:IMOModel)
DelCol(Col:variant)
DelCols(StartCol:variant,EndCol:variant)
DelModel()
DelNZ(Col:variant,Row:variant)
DelObj(Col:variant[])
DelRow(Row:variant)
DelRows(SartRow:variant,EndRow:variant)
DelSOS(SOSID:int)

// Modellabfrage
GetSOS(SOSID:int,{out}ColIndices:variant,{out}Order:variant)

// Modell I/O
ReadAMPLModel(MODFileName:string, DATFileNames:variant[])
ReadMathProgModel(MODFileName:string, DATFileNames:variant[])
ReadMPLModel(MODFileName:string)
ReadModel(FileName:string, FileFormat:enum)
WriteDebugModel(FileName:string,{StartRow:variant},{EndRow:variant},
                                {StartCol:variant},{EndCol:variant})
WriteModel(FileName:string, FileFormat:enum)
WriteSolFile(Filename:string)

// Sonstige Methoden
AutoRenameVariables()
GenFileNames(BaseName:string)
GetSolution(Solver:ISolver)
ShowPropertyPage({out}IsDirty:bool)
SubmitModel(Solver:ISolver)
```

«interface»
IMOSolver

```
StopOptimizeAsync()
Parameter(Name:string):variant {rw}
IsBusy:bool {r}
SolverStatus:enum {r}
LastSolverMessage:string {r}
```

```
ExecuteAMPLModel(MODFileName:string)
GetParameterInfo(Parameter:string,{out}Min:dbl,{out}Max:dbl,{out}Default:dbl)
Optimize()
OptimizeAsync()
ResetParameters()
ReadProfile(FileName:string)
WriteProfile(FileName:string)
```

Einfache IndexSets

«interface»
IMOIndexset

```
AllowIdenticalItems:bool {rw, default=false}
Count:int {r}
Contains(Elements:variant[]):bool {r}
DotProduct(Vector:variant):double {r}
Enum({Start:variant},{End:variant},{Step:variant}):IMOSetEnum {r}
Index(Element:variant):int {r}
IsIn(Set:variant):bool {r}
Item(Index:int):variant {rw}
```

```
ItemDataType:enum {rw, default=vt_variant}
Name:string {rw}
FindNextIndex(Element:variant,StartIndex:int):int {r}
Set(Elements:variant[]):IMOIndexset {r}
---
Clear()
Add(Elements:variant)            // auch für Union()
Addition(Values:variant)
Del(Elements:variant)            // auch für Diff()
DelAt(Index:int)
InsertAfter(Index:int,Elements:variant[])
InsertBefore(Index:int,Elements:variant[])
Intersect(Source:CIndexset)
```

«interface»
IMOSeries

```
Count:int {r}
Series(Start:dbl,End:dbl,{Step:variant}):IMOSeries {r}
---
Init(Start:dbl,End:dbl,{Step:variant})
```

Multi-IndexSets und abgeleitete Interfaces

«interface»
IMultiIndexset

```
CurrentTuple:IMOTuple {r}
Indexset(i:int):variant {rw}
IndexsetElement(IndexsetNo:int,PositionInSet:int):variant {r}
Name:string {rw}
NumberOfIndexsets:int {rw}
SizeOfIndexset(i:int):int {r}
TotalNumberOfElements:int {r}

// Enumerator Factory
Enum({FilterIndexSet1:variant,…,FilterIndexSet5:variant}):IUnknown
---
// Allgemeine Methoden
CloneIndexSetsFrom(MultiIndexset:IMultiIndexset)
InitIndexSets(Indexsets:variant[])

// Enumerator Filter
ClearEnumFilter()
SetEnumFilter(Indexsets:variant[])
GetEnumFilter({out}Indexsets:IMultiIndexset)
```

«interface»
IMODataArray

```
CurrentValue:variant {rw}
Value(Index1:variant,… {Index5:variant}):variant {rw}
ValueByFullIndexName(Indexname:string):variant {rw}
ValueByNumIndex(Index1:int,… {Index5:int}):variant {rw}
ValueDataType:enum {rw, default=vt_variant}
---
AddMatrix(Matrix:variant)
AddScalar(Value:dbl)
Clear()
CloneFrom(Array:variant)
MultiplyScalar(Value:dbl)
MultiplyMatrices(Matrix1:variant, Matrix2:variant)
SetValues(Values:variant[])          // Werte in aufeinander folgender Ordnung
TransposeMatrix()
```

«interface» **IMOVarSet**
CurrentVar:IMOVar {r} Parent:IMOModel Var((Index1:variant,… {,Index5:variant}):IMOVar **// Variableneigenschaften** LB({Index1:variant,…,Index5:variant}):variant {rw} UB({Index1:variant,…,Index5:variant}):variant {rw} Cost({Index1:variant,…,Index5:variant}):variant {rw} Type({Index1:variant,…,Index5:variant}):variant {rw} Status{Index1:variant,…,Index5:variant}):variant {r} LPSol({Index1:variant,…,Index5:variant}):variant {r} IPSol({Index1:variant,…,Index5:variant}):variant {r} RedCost({Index1:variant,…,Index5:variant}):variant {r} VarName(PositionInSet:int):string {r}
Init(Model:IMOModel,Name:string,Indexsets:variant[]) Clear()

«interface» **IMOResSet**
CurrentRes:IMORes {r} Parent:IMOModel Res((Index1:variant{,…,Index5:variant}):IMORes {r} **// Restriktionseigenschaften** LHS({Index1:variant,…,Index5:variant}):variant {rw} RHS({Index1:variant,…,Index5:variant}):variant {rw} Status{Index1:variant,…,Index5:variant}):variant {r} LPSol({Index1:variant,…,Index5:variant}):variant {r} IPSol({Index1:variant,…,Index5:variant}):variant {r} Dual({Index1:variant,…,Index5:variant}):variant {r} ResName(PositionInSet:int):string {r}
Add(CoefsAndVars:variant[]) Clear() Del(Vars:varaint[]) Init(Model:IMOModel,Name:string,IndexSets:variant[]) SetLT(RHS:dbl) SetEQ(LHSRHS:dbl) SetGT(LHS:dbl)

Enumerator-Interfaces

«interface» **IEnumVARIANT**
// Methoden des OLE Standard-Interface IEnumVARIANT (für For…Each Schleifen) Clone(CloneEnumerator:IEnumVARIANT**{out}) Next(Celt:int, Values:variant*{out}, Fetched:int{out}) Skip(Celt:int) Reset()

«interface» **IMOCustomEnum**
CurrentIndexName:string {r} IsBeyondLast:bool {r} IsLast:bool {r}
MoveFirst() MoveNext() get__NewEnum(Enum:IUnknown**)

«interface» **IMOIndexSetEnum**
Parent:IMOIndexset {r} CurrentIndexNum:int {r} CurrentItem:variant {r}
Init(Parent:IMOIndexset,{Start:variant},{End:variant},{Step:variant})

«interface» **IMOSeriesEnum**
// nur geerbte Eigenschaften und Methoden

«interface» **IMultiIndexSetEnum**
CurrentTuple:IMOTuple {r} Indexset(i:int):variant {r} IndexsetElement(IndexSetNo:int,PositionInSet:int):variant {r} NumberOfIndexsets:int {r} Parent:IUnknown {r} SizeOfIndexset(i:int):int {r} TotalNumberOfElements:int {r}
// Enumerator Filter ClearEnumFilter() SetEnumFilter(Indexsets:variant[]) GetEnumFilter({out}Indexsets:IMultiIndexset)

«interface» **IMODataArrayEnum**
CurrentItem:variant {r}
Init(Parent:IMODataArray,{FilterIxSet1:variant,…,FilterIxSet5:variant})

«interface» **IMOVarSetEnum**
CurrentVar:IMOVar {r}
Init(Parent:IMOVarSet,{FilterIxSet1:variant,…,FilterIxSet5:variant})

«interface» **IMOResSetEnum**
CurrentRes:IMORes {r}
Init(Parent:IMODataArray,{FilterIxSet1:variant,…,FilterIxSet5:variant})

Variablen- und Restriktionen

«interface» **IMOVar**
Parent:IMOModel {r} IndexInModel:int {r} // **Variableneigenschaften** Cost:dbl {rw} IPSol:dbl {r} LB:dbl {rw} LPSol:dbl {r} Name:string {rw} RedCost:dbl {r} Status:int {r} Type:int {rw} UB:dbl {rw}
AddToModel(Model: IMOModel) Attach(Model:IMOModel,ColIndex:variant) Detach()

«interface» **IMORes**
Parent:IMOModel {r} IndexInModel:int {r} // **Restriktionseigenschaften** Dual:dbl {r} IPSol:dbl {r} LHS:dbl {rw} LPSol:dbl {r} Name:string {rw} RHS:dbl {rw} Status:int {r}
Add(CoefsAndVars:variant[]) AddToModel(Model: IMOModel) Attach(Model:IMOModel,RowIndex:variant) Detach() Del(Vars:varaint[]) SetLT(RHS:dbl) SetEQ(LHSRHS:dbl) SetGT(LHS:dbl)

Sonstige

«interface» **IMOTuple**
NumberOfElements:int {rw} Element(i:int):variant {rw} FullIndexName:string {rw} Set(Elements:variant[]):IMOTuple {r}
Init(Elements:variant[])

Anhang 6: Prozedurales Model Manager Interface

Zur Identifikation eines verwalteten Modells wird eine eindeutige ID (Long-Wert) verwendet. Gleiches gilt für die vom Solver Manager verwalteten Solver-Instanzen, auch diese werden über eine Solver-ID (SID) unterschieden. So kann das selbe Modell z.B. an zwei unterschiedliche Solver übergeben werden und von diesen parallel optimiert werden. Alle Funktionen geben einen Long-Wert als Status zurück. Um die Funktionszahl zu begrenzen werden alle Einzelwerte (z.B. Modelldimensionen, Solvereinstellungen etc.) über `GetParameter-`/`SetParameter`-Funktionen bearbeitet. Aus Platzgründen ist der Typ der Argumente weggelassen.

Model- und Solverhandling

Name	Bedeutung
`CreateModel([out]ID)`	Neues Modell anlegen (ID = Model-ID)
`DeleteModel(ID)`	Model löschen
`CloneModel(IDSource,IDDestination)`	Model duplizieren
`CreateSolver(Solvertyp,[out]SID)`	Neue Solverinstanz erzeugen (SID = Solver-ID)
`DeleteSolver(SID)`	Solverinstanz löschen

File I/O

Name	Bedeutung
`ReadModel(ID,Filename,Format)`	Modell einlesen (diverse Formate)
`ReadAMPLModel(ID,MODFileName,DATFileName1,…)`	AMPL Modell einlesen. Für weitere Modellierungssprachen ähnliche Funktionen.
`WriteModel(ID,Filename,Format)`	Modell ausgeben (diverse Formate)
`WriteSolFile(ID,Filename)`	Lösungsdatei ausgeben
`ReadParamFile(SID,Filename)`	Solver-Parameterdatei einlesen
`WriteParamFile(SID,Filename)`	Solver-Parameterdatei schreiben

Modellaufbau und -modifikation

Name	Bedeutung
`AddCol(ID,UB,LB,Cost,Typ,Name)`	Variable hinzufügen
`AddCols(ID,n,UBs,LBs,Costs,Typs,Names)`	n Variablen hinzufügen (Arrays)
`InsCol(ID,j,UB,LB,Cost,Typ,Name)`	Variable vor Index j einfügen
`InsCols(ID,j,n,UBs,LBs,Costs,Typs,Names)`	n Variablen vor Index j einfügen
`DelCol(ID,j)`	Variable mit Index j löschen
`DelCols(ID,Startj,Endj)`	Variablen löschen (Start- bis Endindex)
`AddRow(ID,LHS,RHS,Name)`	Restriktion hinzufügen
`AddRows(ID,n,LHSs,RHSs,Names)`	n Restriktionen (Arrays)
`InsRow(ID,i,LHS,RHS,Name)`	Restriktion vor Index i einfügen
`InsRows(ID,i,n,LHSs,RHSs,Names)`	n Restriktionen vor i einfügen. (Arrays)
`DelRow(ID,i)`	Restriktion mit Index i löschen
`DelRows(ID,Starti,Endi)`	Restr. löschen (Start- bis Endindex)
`PutNonzero(ID,j,i,Value)`	Nonzero setzen/überschreiben.
`PutNonzeros(ID,n,ColIndices,RowIndices,Values)`	n Nonzeros setzen/überschreiben
`ChColLB(ID,j,LB)`	Variablenuntergrenze ändern
`ChColLBs(ID,Startj,Endj,LBs)`	Variablenuntergrenzen ändern
`ChColUB(ID,j,UB)`	Variablenobergrenze ändern
`ChColUBs(ID,Startj,Endj,UBs)`	Variablenobergrenzen ändern
`ChColCost(ID,j,Cost)`	Zielfunktionskoeffizient ändern
`ChColCosts(ID,Startj,Endj,Costs)`	Zielfunktionskoeffizienten ändern
`ChColType(ID,j,Type)`	Variablentyp ändern.
`ChColTypes(ID,Startj,Endj,Types)`	Variablentypen ändern
`ChColName(ID,j,Name)`	Variablenname ändern
`ChColNames(ID,Startj,Endj,Names)`	Variablennamen ändern.
`ChRowRHS(ID,i,RHS)`	Rechte Seite ändern
`ChRowRHSs(ID,Starti,Endi,RHSs)`	Rechte Seiten ändern
`ChRowLHS(ID,i,LHS)`	Linke Seite ändern

`ChRowLHSs(ID,Starti,Endi,LHSs)`	Linke Seiten ändern
`ChRowName(ID,i,Name)`	Restriktionsname ändern
`ChRowNames(ID,Starti,Endi,Names)`	Restriktionsnamen ändern
`AddSOS(ID,Type,n,ColIndices,Order,[out]SOSID)`	SOS hinzufügen
`DelSOS(ID,SOSID)`	SOS löschen. In: SOS-ID

`PutModel(ID,m,n,nz,`
`ColLBs,ColUBs,Costs,ColTypes,ColNames,`
`LHSs,RHSs,RowNames,`
`ColIndices,RowIndices,Nonzeros)`
	Ganzes Modell übergeben

Modellabfrage

Name	Bedeutung
`GetColLB(ID,j,[out]LB)`	Untergrenze auslesen
`GetColLBs(ID,Startj,Endj,[out]LBs)`	Untergrenzen auslesen (Array)
`GetColUB(ID,j,[out]UB)`	Obergrenze auslesen
`GetColUBs(ID,Startj,Endj,[out]UBs)`	Obergrenzen auslesen (Array)
`GetColType(ID,j,[out]Type)`	Typ auslesen
`GetColTypes(ID,Startj,Endj,[out]Types)`	Typen auslesen (Array)
`GetColCost(ID,j,[out]Cost)`	Zielfunktionskoeffizient auslesen
`GetColCosts(ID,Startj,Endj,[out]Costs)`	Zielfunktionskoeffizienten auslesen (Array)
`GetColName(ID,j,[out]Name)`	Variablenname auslesen
`GetColNames(ID,Startj,Endj,[out]Names)`	Variablennamen auslesen (Array)
`GetRowLHS(ID,i,[out]LHS)`	Linke Seite auslesen
`GetRowLHSs(ID,Starti,Endi,[out]LHSs)`	Linke Seiten auslesen (Array)
`GetRowRHS(ID,i,[out]RHS)`	Rechte Seite auslesen
`GetRowRHSs(ID,Starti,Endi,[out]EHSs)`	Rechte Seiten auslesen (Array)
`GetRowName(ID,i,[out]Name)`	Restriktionsname auslesen
`GetRowNames(ID,Starti,Endi,[out]Names)`	Restriktionsnamen auslesen (Array)
`GetNonzero(ID,j,i,[out]Value)`	Nonzero auslesen
`GetNonzeros(ID,ColIndices,RowIndices,[out]Values)`	Nonzeros auslesen (Array)
`GetColNonzeros(ID,j,[out]RowIndices,[out]Values)`	Nonzeros einer Spalte auslesen
`GetRowNonzeros(ID,i,[out]ColIndices,[out]Values)`	Nonzeros einer Zeile auslesen
`GetSOSnElements(ID,SOSID,[out]n)`	Anzahl Elemente SOS auslesen.
`GetSOS(ID,SOSID,[out]ColIndices,[out]Order)`	SOS auslesen.
`GetColIndex(ID,ColName,[out]j)`	Spaltenindex suchen
`GetColIndices(ID,n,ColNames,[out]ColIndices)`	Spaltenindizes suchen (Array)
`GetRowIndex(ID,RowName,[out]i)`	Zeilenindex suchen
`GetRowIndices(ID,n,RowNames,[out]RowIndices)`	Zeilenindices suchen (Array)

`GetModel(ID,[out]m, [out]n, [out]nz,`
`[out]ColLBs, [out]ColUBs, [out]Costs,`
`[out]ColTypes, [out]ColNames,`
`[out]LHSs, [out]RHSs, [out]RowNames,`
`[out]ColIndices, [out]RowIndices, [out]Nonzeros)`
	Komplettes Modell auslesen

Lösungsabfrage

Name	Bedeutung
`GetLPSol(ID,j,[out]Value)`	LP Lösungswert auslesen
`GetLPSols(ID,Startj,Endj,[out]Values)`	LP Lösungswerte auslesen (Array)
`GetIPSol(ID,j,[out]Value)`	IP Lösungswert auslesen
`GetIPSols(ID,Startj,Endj,[out]Values)`	IP Lösungswerte auslesen (Array)
`GetRedCost(ID,j,[out]Value)`	Reduzierte Kosten auslesen
`GetRedCosts(ID,Startj,Endj,[out]Values)`	Reduzierte Kosten auslesen (Array)
`GetBaseStat(ID,j,[out]Value)`	Basis Status auslesen
`GetbaseStats(ID,Startj,Endj,[out]Values)`	Basis Status auslesen (Array)
`GetActivity(ID,i,[out]Value)`	Row Activities auslesen
`GetActivities(ID,Starti,Endi,[out]Values)`	Row Activities auslesen (Array)
`GetDual(ID,i,[out]Value)`	Dualwert auslesen
`GetDuals(ID,Starti,Endi,[out]Values)`	Dualwerte auslesen (Array)
`GetSlack(ID,i,[out]Value)`	Slackwert auslesen

GetSlacks(ID,Starti,Endi,[out]Values)	Slackwerte auslesen (Array)
GetRowBaseStat(ID,i,[out]Value)	Zeilenbasisstatus auslesen
GetRowBaseStats(ID,Starti,Endi,[out]Values)	Zeilenbasisstatus auslesen (Array)

Lösungsprozesssteuerung

Name	Bedeutung
SubmitModel(ID,SID)	Überträgt Model ID an Solver SID
Optimize(SID)	Start Optimierung mit Solver SID
GetSolution(ID,SID)	Überträgt Lösung aus Solver SID nach Model ID

Parameterhandling

Name	Bedeutung
GetParameterName(ParameterID,[out]ParameterName)	Zu Parametercode gehörigen Parameternamen finden.
GetParameterID(ParameterName,[out]ParameterID)	Zu Parameternamen gehörigen Parametercode finden.
GetDblParameterInfo(ParameterID, [out]Min,[out]Max,[out]Default)	Default, Minimum und Maximum für Double-Parameter auslesen.
GetIntParameterInfo(ParameterID, [out]Min,[out]Max,[out]Default)	Default, Minimum und Maximum für Integer-Parameter auslesen.
GetStrParameterInfo(ParameterID,[out]Default)	Default für String-Parameter auslesen.
Model-Parameter	
SetDblMParameter(ID,ParameterID,Value)	Parameter vom Typ Double setzen
SetIntMParameter(ID,ParameterID,Value)	Parameter vom Typ Integer setzen
SetStrMParameter(ID,ParameterID,Value)	Parameter vom Typ String setzen
GetDblMParameter(ID,ParameterID,[out]Value)	Parameter vom Typ Double auslesen
GetIntMParameter(ID,ParameterID,[out]Value)	Parameter vom Typ Integer auslesen
GetStrMParameter(ID,ParameterID,[out]Value)	Parameter vom Typ String auslesen
CloneMParameters(IDSource,IDDestination)	Kompletten Parametersatz zwischen zwei Modellen kopieren
SetMDefaults(SID)	Model-Parameter auf Defaults zurücksetzen
Solver-Parameter	
SetDblSParameter(SID,ParameterID,Value)	Solver-Parameter vom Typ Double setzen
SetIntSParameter(SID,ParameterID,Value)	Solver-Parameter vom Typ Integer setzen
SetStrSParameter(SID,ParameterID,Value)	Solver-Parameter vom Typ String setzen
GetDblSParameter(SID,ParameterID,[out]Value)	Solver-Parameter vom Typ Double auslesen
GetIntSParameter(SID,ParameterID,[out]Value)	Solver-Parameter vom Typ Integer auslesen
GetStrSParameter(SID,ParameterID,[out]Value)	Solver-Parameter vom Typ String auslesen
CloneSParameters(IDSource,IDDestination)	Kompletten Parametersatz zwischen zwei Solvern kopieren
SetSDefaults(SID)	Solver-Parameter auf Defaults zurücksetzen

Callbacks

Name	Bedeutung
SetCallback(SID, CBType, CBFuncP)	Setzt Callback für einen bestimmten CB-Typ. CBFuncP ist ein Pointer auf die CallbackFunktion. Zurücksetzen mit CBFuncP = 0