Java Interface

Reference Manual

XA

Solver Toolkit

Version 4.1

Directory Structure	Files	Meaning
	Note: All run.bat files use an environment variable JDKDIR. Prior to running any XA examples create this environment to point to your root Java Directory, for example, set jdkdir=d:\jdk1.4 New version of Java runtime environment should also work.	
..\Xalib		
	XAOptimizer.jar	Optimizer Class
	n2xav2.dll & xav15.dll	XA Native Library - copy dll files to a path directory or the \windows
..\examples		**Examples**
	example2*.*	MIP model.
	example3*.*	Transshipment Model.
	example4*.*	TSP example, multiple MIP Solves.
	example7*.*	Efficient portfolio (requires Quadratic Programming Opt.)
	example8*.*	Sensitivity Analysis- Objective Coef, Column Activity, and RHS values.
	example9*.*	MPS file Reader.
	example10*.*	RCC Files Reader.
	example11*.*	Graphic Applet TSP Example.
	example12*.*	Infeasible LP with Conflict Analysis.
	example13*.*	Optimize using previous MIP solution.

All examples have …\xalib\XAOptimizer.jar in their classpath. The classpath is defined in run.bat

…\JavaRuntime	j2sdk1_3_0-win.exe	Sun Java Software Development Kit Version 2.0 Release 1.3 for Windows.
	j2sdk1_3_0-doc.zip	Sun Java Software Development Kit Version 2.0 Release 1.3 Documentation
	jre-1_2_2_007-win.exe	Sun Java Runtime Environment Version 1.3
	htmlconv1_3.zip	HTML Applet converter

Optimizer Class

public…………void close( )

- final method to invoke…closes connection to XA and frees memory and closes files.

public…………void closeConnection( )

- close connection to Solver Engine, frees memory and closes files. Invoke openConnection to reactivate Solver Engine with the same object.

public…………void deleteColumn(String colName)

- deletes all 'colName' coefficients from model and the name is removed from the column name list.
- 'colName' can be re-added to model as a fresh new column.

public…………void deletePoint(String rowName , String colName)

- deletes the technology coefficient at the interection of 'rowName' and 'colName'.

public…………void deleteRow(String rowName)

- deletes all 'rowName' coefficients from model and the name is removed from the row name list.
- 'rowName' can be re-added to model as a fresh new row.

public………….int estimateMaxMem(int estimateRows, int estimateColumns, int estimateNonZeros)

- Estimates the 'maxmem' value based on model rows, columns, and nonzeros.

public………double getActivity( )

- getColumnInformation, getRowInformation, getColumnActivitySensitivity, getColumnObjectiveSensitivity and getRowActivitySensitivity methods identify the 'active' column or row name.
- returns activity value of the 'active' column/row.

public………double getBestIntegerObjective( )

- returns the best possible integer solution objective value.
- during the branch and bound process of solving MIP models XA bounds the objective range of possible integer solutions. This value converges on the optimal integer objective value.

public………double getColumnActivity(String colName)

- returns activity value of 'colName'.
- returns zero for 'bogus' colName.

public..………..void getColumnActivitySensitivity(String colName)

- calculates sensitivity analysis information for column 'colName' activity.
- use getActivity, getLowerBoundValue, getLowerBoundRange, getUpperBoundValue, getUpperBoundRange to return specific results.
- returns zero for 'bogus' colName.

Optimizer Class

public………double getColumnDual(String colName)

- returns dual activity value of ‘colName’.
- returns zero for ‘bogus’ colName.

public..………..void getColumnInformation(String colName)

- calculates column information for column ‘colName’.
- use getObjectiveCoef, getActivity, getLowerBound, getUpperBound, getDual and any ‘is….’ methods to return specific results.
- returns zero for ‘bogus’ colName.

public..………..void getColumnObjectiveSensitivity(String colName)

- calculates sensitivity analysis information for column ‘colName’ objective coefficient.
- use getActivity, getLowerBoundValue, getLowerBoundRange, getUpperBoundValue, getUpperBoundRange to return specific results.
- returns zero for ‘bogus’ colName.

public..………..…int getColumns( )

- returns number of columns in model

.

public………double getElapseWallClockTime( )

- number of wall clock seconds since openConnection method was called.

public………double getDual( )

- getColumnInformation, getRowInformation, getColumnActivitySensitivity, getColumnObjectiveSensitivity and getRowActivitySensitivity methods identify the ‘active’ column or row name.
- returns dual value of the ‘active’ column/row.

public..………..…int getExceptionCode( )

- identifies XA reason for throwing an exception.
- use getMessage for textual message.
- a value of zero indicates the Optimizer did not throw the exception. It was throw by problems in your code.
- see Appendix A for definition.

public..………..…int getIPTime( )

- wall clock time including LP time, in seconds, to solve a MIP model.

public..………..…int getIterations( )

- number of iterations to solve the model.

public………double getLowerBound( )

- getColumnInformation and getRowInformation methods identify the ‘active’ column or row name.
- returns the lower bound of the ‘active’ column/row.
- -infinity is represented by a value less than -1.0e23.

Optimizer Class

public……..…String getLowerBoundName( )
- getColumnActivitySensitivity, getColumnObjectiveSensitivity and getRowActivitySensitivity methods identify the ‘active’ column or row name.
- returns column or row name which limits ‘active’ column/row movement below getLowerBoundRange value.

public………double getLowerBoundRange( )
- getColumnActivitySensitivity, getColumnObjectiveSensitivity and getRowActivitySensitivity methods identify the ‘active’ column or row name.
- returns lower bound value which limits ‘active’ column/row movement
- -infinity is represented by a value less than -1.0e23.

public..…………..…int getLPTime( )
- wall clock time, in seconds, to solve the LP part of a model.

public……..…String getMessage( )
- reason XA throws an exception.
- use in conjunction with getExceptionCode.

public..…………..…int getModelStatus( )
- disposition of the solve method.
- indicate the status (optimal, infeasible,…) of model, see Appendix B for codes.
- additional information is found with getSolverStatus( ) and getRc( ) values.

public..…………..…int getNonzeros( )
- number of nonzero technology coefficients in model.

public………double getObjective( )
- value of the objective function.

public………double getObjectiveCoef( )
- getColumnInformation method identifies the ‘active’ column.
- returns objective function coefficient of the ‘active’ column.

public..…………….int getRc( )
- return code value set by each method.
- used to identify or debug coding problems.
- see Appendix C for values.

public………double getRowActivity(String rowName)
- returns activity value of ‘rowName’.
- the left-hand-side value.
- returns zero for ‘bogus’ rowName.

Optimizer Class

public..………..void getRowActivitySensitivity(String rowName)

- calculates sensitivity analysis information for row ‘rowName’ activity.
- use getActivity, getLowerBoundValue, getLowerBoundRange, getUpperBoundValue, getUpperBoundRange to return specific results.
- returns zero for ‘bogus’ rowName’.

public………double getRowDual(String rowName)

- returns dual value of ‘rowName’.
- returns zero for ‘bogus’ rowName.

public..………..void getRowInformation(String rowName)

- calculates row information for row ‘rowName’.
- use getObjectiveCoef, getActivity, getLowerBound, getUpperBound, getDual and any ‘is….’ methods to return specific results.
- returns zero for ‘bogus’ rowName.

public..………..…int getRows( )

- number of rows in model.

public..………..…int getSolverStatus( )

- disposition of the solve method.
- indicate the solver status (normal, time limit exceeded,…) of model, see Appendix B for codes.
- additional information is found with getModelStatus( ) and getRc( ) values.

public………double getUpperBound( )

- getColumnInformation and getRowInformation methods identify the ‘active’ column or row name.
- returns the upper bound of the ‘active’ column/row.
- +infinity is represented by a value greater than +1.0e23.

public……..…String getUpperBoundName( )

- getColumnActivitySensitivity, getColumnObjectiveSensitivity and getRowActivitySensitivity methods identify the ‘active’ column or row name.
- returns column or row name which limits ‘active’ column/row movement above getUpperBoundRange value.

public………double getUpperBoundRange( )

- getColumnActivitySensitivity, getColumnObjectiveSensitivity and getRowActivitySensitivity methods identify the ‘active’ column or row name.
- returns upper bound value which limits ‘active’ column/row movement
- +infinity is represented by a value greater than 1.0e23..

Optimizer Class

public…..…. boolean isBlocked( )

- getColumnInformation and getRowInformation methods identify the 'active' column or row name.
- true value indicates the 'active' column or row is only used if the model is infeasible.

public…..…. boolean isFeasible( )

- getColumnInformation and getRowInformation methods identify the 'active' column or row name.
- true value indicates the 'active' column or row is primal and dual feasible.

public…..…. boolean isInBasis( )

- getColumnInformation and getRowInformation methods identify the 'active' column or row name.
- true value indicates the 'active' column or row is in the basis.

public…..…. boolean isInfeasible( )

- getColumnInformation and getRowInformation methods identify the 'active' column or row name.
- true value indicates the 'active' column or row is infeasible.

public…..…. boolean isInformationAvailable( )

- getColumnInformation, getRowInformation, getColumnActivitySensitivity, getColumnObjectiveSensitivity and getRowActivitySensitivity methods identify the 'active' column or row name.
- true value indicates the 'active' column or row has the requested information.
- false value indicates the 'active' column or row name is 'bogus'.

public…..…. boolean isInteger( )

- getColumnInformation method identify the 'active' column.
- true value indicates the 'active' column is binary or integer.

public…..…. boolean isSemiContinuous( )

- getColumnInformation method identify the 'active' column.
- true value indicates the 'active' column is a semi-continuous column.

public…..…. boolean isSemiContinuousII( )

- getColumnInformation method identify the 'active' column.
- true value indicates the 'active' column is a semi-continuous type II column.

public…..…. boolean isUnbounded( )

- getColumnInformation and getRowInformation methods identify the 'active' column or row name.
- true value indicates the 'active' column or row is unbounded (also infeasible).

public..……….void limitIterations(int maxIterations)

- limits the number of simplex or interior point iterations.

Optimizer Class

public............void limitNumberOfIntegerSolutions(int solutionCount)

- solve( ) method terminates after finding ‘solutionCount’ integer solutions.
- solve( ) method may return before finding ‘solutionCount’ solutions because of other reasons finding the ‘best’ possible solution or time limitations.
- see Appendix B for solver status values.

public............void limitTime(int timeLimit)

- number of wall clock seconds available to solve the model.
- time keeping begins when solve( ) method is invoked.
- the solve( ) method may (hopefully) return before the timeLimit is exceeded.
- if ‘timelimit’ is exceed then
    - getSolverStatus( ) returns 3.
    - check getModelStatus( ) to determine solution availability. MIP models will return the best integer solution found (if one was found).
    - see Appendix B for getSolverStatus and getModelStatus definitions.

public............void limitTime(String timeLimit)

- format of String timeLimit is   hh:mm:ss
- see limitTime(int timeLimit) for more information.

public............void limitTime(String timeLimit)

- format of String timeLimit is   hh:mm:ss
- see limitTime(int timeLimit) for more information.

public............void limitTimeStopAfter(int timeLimit)

- number of wall clock seconds to continue solving model after the first integer solution is found.
- see limitTime(int timeLimit) for more information.

public............void limitTimeStopAfter(String timeLimit)

- format of String timeLimit is hh:mm:ss
- see limitTimeStopAfter(int timeLimit) for more information.

public............void limitTimeStopUnchanged(int timeLimit)

- number of wall clock seconds to continue solving model since the previous integer solution is found.
- Each time an integer solution is found the termination time is extended by ‘timeLimit’ seconds.
- see limitTime(int timeLimit) for more information.

public............void limitTimeStopUnchanged(String timeLimit)

- format of String timeLimit is hh:mm:ss
- see limitTimeStopUnchanged(int timeLimit) for more information.

Optimizer Class

public..………..void loadPoint(String rowName , String colName , double techCoef)

- presents XA Solver Engine with a technology coefficient.
- see Appendix D for reserved row and column names.
- loadPoint should not be invoked with zero A matrix coefficients for performance reasons only.

public..………..void loadRCCData(String fileName)

- reads RCC format data from filename 'fileName' .
- by default the rowNameSize and colNameSize are read for the RCC file header.

public..………..void loadRCCData(String fileName , int rowNameSize , int colNameSize)

- see loadRCCData(String fileName)
- rowNameSize will override the RCC header value if larger.
- colNameSize will override the RCC header value if larger.

public..………..void loadRCCData(String FileName , int rowNameSize , int colNameSize ,
int maxRow , int maxCol , int maxNzo)

- see loadRCCData(String fileName , int rowNameSize , int colNameSize)
- maxRow will override the RCC header value if larger.
- maxCol will override the RCC header value if larger.
- maxNzo will override the RCC header value if larger.

public..………..void loadToCurrentColumn(String rowName)

- rowName is connected to the most recently loaded colName with a technology coefficient of +1.0.
- you must have 'loaded' the desired colName prior to using this method.
- reserved 'colName' does not changed the most recently loaded colName.

public..………..void loadToCurrentColumn(String rowName , double techCoef)

- see loadToCurrentColumn(String rowName)
- rowName is connected to the most recently loaded colName with a technology coefficient of 'techCoef'.
- you must have 'loaded' the desired colName prior to using this method.
- reserved 'colName' does not changed the most recently loaded colName.

public..………..void loadToCurrentRow(String colName)

- colName is connected to the most recently loaded rowName with a technology coefficient of +1.0.
- you must have 'loaded' the desired rowName prior to using this method.
- reserved 'rowName' does not changed the most recently loaded rowName.

public..………..void loadToCurrentRow(String colName , double techCoef)

- see loadToCurrentRow(String colName)
- colName is connected to the most recently loaded rowName with a technology coefficient of 'techCoef'.
- you must have 'loaded' the desired rowName prior to using this method.
- reserved 'rowName does not changed the most recently loaded rowName.

Optimizer Class

public…………void openConnection(String aServer)

- connects Java Environment to XA Optimizer DLL .
- aServer value should be "local" but can be any value. Future XA/Java releases may support XA running on different machines.
- If dongle is missing, or
  expiration date exceeded, or
  n2xav2.dll or xav15.dll are unavailable
  and XAException is thrown.

* public..……………. Optimizer(int maxMem)

- constructor for the Optimizer Class.
- 'maxMem' is amount of memory available to solve the model. Estimate memory requirements at 500 bytes per nonzero coefficients.
- default rowNameSize is 12 characters and colNameSize is 14 characters.
- default values for maxRow and maxCol are calculated from 'maxMem' request.
- 'maxMem' value of zero will request all available memory less eight megabtyes.

* public..……………. Optimizer(int maxMem, int maxRow, int maxCol, int maxNzo, int rowNamSize, int colNameSize)

- constructor for the Optimizer Class.
- see Optimizer(int maxMem).
- 'maxRow', 'maxCol', 'maxNzo', 'rowNameSize' and 'colNameSize' values override their respective default values.

public..………..void outMessage(String TextWrittenToXAOutputDevice)

- sends String 'TextWrittenToXAOutputDevice' to XA output device.
- the default XA output device is the "XA Message Window" which will appear on the Server, use the following method to change this device to a file. (Remember the filename is relative to the Server machine).
  MyModel.setCommand(" Output \\jim\c\xa.log ") ;

public outMessageToTraceLog(String TextWrittenToTraceLog)

- sends String 'TextWrittenToTraceLog' to trace log file.
- useful when debugging.

public…………void setActivationCodes (int Code1 , int Code2)

- codes used to activate the Server machine.
- if Activation codes are required then this method must be called before invoking openConnection( ) method.

public…………void setAlgorithmInteriorPoint( )

- set the XA Server optimization algorithm to the interior point method.
- setting ignored if the interior point method was not licensed.
- if the model is a binary/integer model then the Server automatically switches to the Simplex Algorithm to solve the MIP portion of the model.
- requires Interior Point license.

Optimizer Class

public…………void setAlgorithmInteriorPointCrossover( )
- see setAlgorithmInteriorPoint for more information.
- after solving the model with the interior point algorithm the Server switches to the Simplex Algorithm to achieve a vertex solution.
- if the model is a binary/integer model then the Server automatically switches to the Simplex Algorithm to solve the MIP portion of the model.

public…………void setAlgorithmSimplex( )
- default Server solving algorithm.
- Primal/Dual Simplex algorithm, advance basis restarting with branch and bound MIP solving.

public..………..void setCallbackHandler(RT1Callback callBackHandler)
- used to specify your callback handler.
- see RT1Callback Class definition.
- callback frequency is every iteration which creates a very heavy communication load, so we recommend using setCallbackHandler with a 'callBackFrequency' of five seconds.
- invoke 'setTerminate' method in the callbackHandler to terminate the solve method. When solving MIP model the best integer solution found is returned (if one had been found). Always check getModelStatus( ) and getSolverStatus( ) values to determine the status of your model.

public..………..void setCallbackHandler(RT1Callback callbackHandler , int callBackFrequency)
- see setCallbackHandler for more information.
- callBackFrequency is the interval, in seconds, to call the callbackHandler.
- the callbackHandler always calls these method regardless of callbackFrequency, startingToSolve, integerSolutionFound, and solverComplete.

public..………..void setColumnBigM(String colName)
- indicates a binary column is used in a 'Big M' capacity. The XA Server automatically detects columns that are used in this capacity but this method is available to make sure the column has this setting.

public..………..void setColumnBinary(String colName)
- indicates 'colName' is a binary column.
- a binary column has one of two values either zero (0) or one (1).

public..………..void setColumnBlocked(String colName)
- prevents a column for being used to solve the model unless the model is infeasible, at which time the 'colName' is allow to participate in the solving process.
- block columns that are 'feasible assist' columns. That is, columns you add to your model in the event the model is infeasible. You also need to set the objective function (setColumnObjective(…) coefficients to a large (+/-) values to penalize using these columns once they are activated.

Optimizer Class

public..………..void setColumnBranchingPriority(String colName, int Priority)

- sets the order (sequence) binary/integer columns are branch and bounded (integerized).
- this does not make the column binary or integer, so you must also use setColumnBinary or setColumnInteger method to make a column binary or integer.
- using this feature can significantly improve solve speed and quality of initial solution.

public..………..void setColumnBranchUp(String colName)

- directs the initial branch direction of a binary column.
- this does not make the column binary or integer, so you must also use setColumnBinary or setColumnInteger method to make a column binary or integer.
- when resolving the same instance of a MIP model the branching direction is automatically carried forward.

public..………..void setColumnCoarsiness(String colName , double increment)

- ‘colName’ is make integer.
- increment between success values is ‘increment’ units. By default the increment for a binary or integer column is one (1).
- an ‘increment’ value of zero (0) makes ‘colName’ a continuous column.

public..………..void setColumnFix(String colName , double fixedActivityValue)

- the activity of ‘colName’ is set equal to (fixed at) ‘fixedActivityValue’ value during the solving process.
- There is no deviation from this value.

public..………..void setColumnFree(String colName)

- the activity of ‘colName’ is bound between -infinity and +infinity.

public..………..void setColumnInteger(String colName)

- indicates ‘colName’ is an integer column.
- in the absents of any upper bound specification, the upper bound on ‘colName’ is determined by the ‘Set IBound” value when the first solve( ) method is invoked.
- make sure all your generalized integer columns have finite upper bounds because an column with an infinite bound switches to a bound of 1.0 or a binary column.

public..………..void setColumnInteger(String colName , double maxBound)

- see setColumnInteger(String colName) for more information.
- ‘maxBound’ is used to set the upper bound value for ‘colName’.
- if ‘maxBound’ is infinite (greater than 1.0e23) then “Set IBound” value is used to set the upper bound.

public..………..void setColumnInteger(String colName , double minBound , double maxBound)

- see setColumnInteger(String colName , double maxBound) for more information.
- ‘minBound’ is used to set the lower bound value of ‘colName’.
- ‘minBound’ can be a negative value.
- ‘minBound’ must be less than or equal ‘maxBound’.

Optimizer Class

public..………..void setColumnLowerBoundBase(String colName)
- When a column is defined as an integer/semi-column the default base is zero (0.0), meaning that each sequence must pass through zero (0.0) without regards to the column's feasible region, which usually is the default lower bound for each column..

public..………..void setColumnMax(String colName , double maxBound)
- sets the upper bound value of 'colName' to 'maxBound'.
- in the absents of any upper bound specification 'colName' default upper bound value is +infinite. This default can be changed by setCommand("Set UBColumn …").

public..………..void setColumnMin(String colName , double minBound)
- sets the lower bound value of 'colName' to 'minBound'.
- 'minBound' can be a negative value.
- in the absents of any lower bound specification 'colName' lower bound is 0. This default can be changed by setCommand("Set LBColumn …").

public..………..void setColumnMinMax(String colName , double minBound , double maxBound)
- sets 'colName' lower bound value to 'minBound'.
- sets 'colName' upper bound value to 'maxBound'.
- 'minBound' must be less than or equal to 'maxBound'.

public..………..void setColumnObjective(String colName , double objCoef)
- sets 'colName' objective function coefficient to 'objCoef'.
- the default objective function coefficient for all columns is zero (0).

public..………..void setColumnSemiContinuous(String colName)
- makes 'colName' a semi-continuous column.
- 'minBound' and 'maxBound' must be loaded by other methods
- all semi-continuous columns must have a lower bound greater and or equal to zero.

public..………..void setColumnSemiContinuous(String colName , double minBound)
- see setColumnSemiContinuous for more information.
- set the lower bound value of 'colName' to 'minBound'.
- 'minBound' must be greater than or equal to zero.

public..………..void setColumnSemiContinuous(String colName , double minBound , double maxBound)
- see setColumnSemiContinuous(String colName , double minBound)
- sets the lower bound value of 'colName' to 'minBound'.
- sets the upper bound value of 'colName' to 'maxBound'.
- 'minBound' must be less than or equal to 'maxBound'.

public..………..void setColumnSemiContinuousII(String colName)
- makes 'colName' a semi-continuous type II column.
- 'minBound' and 'maxBound' must be loaded by other methods.
- all semi-continuous type II columns must have a lower bound greater and or equal to zero.

Optimizer Class

public..………..void setColumnSemiContinuousII(String colName , double minBound)

- see setColumnSemiContinuousII for more information.
- set the lower bound value of ‘colName’ to ‘minBound’.
- ‘minBound’ must be greater than or equal to zero.

public..………..void setColumnSemiContinuousII(String colName , double minBound , double maxBound)

- see setColumnSemiContinuousII(String colName , double minBound)
- sets the lower bound value of ‘colName’ to ‘minBound’.
- sets the upper bound value of ‘colName’ to ‘maxBound’.
- ‘minBound’ must be less than or equal to ‘maxBound’.

public..……..void setColumnSuggestIntegerSolution(String colName , double suggestedIntegerSolution)

- sets first branch and bound direction towards ‘suggestedIntegerSolution’.
- this does not make the column binary or integer, so you must also use setColumnBinary or setColumnInteger method to make a column binary or integer.
- after the first solve( ) this value is updated to the new integer solution, if one was found.

public..………..void setCommand(String CLP)

- send Server an command line parameter.
- see Appendix F for more detail on these commands.
- all filename names are relative to the Server machine.
- see Section 3.0 COMMAND LINE PARAMETERS in XA Callable Library manual for command line parameter documentation.

public…………void setConflictAnalysisTypeIOn( )

- When XA detects an LP infeasibility a special routine is run to identify the conflict constraints.
- Since conflict analysis is usually not unique there are two different routines available. Trial and error is used to determine which technique works best on your models.
- The conflicting constraints are identified in the XA log file.

public…………void setConflictAnalysisTypeIIOn( )

- When XA detects an LP infeasibility a special routine is run to identify the conflict constraints.
- Since conflict analysis is usually not unique there are two different routines available. Trial and error is used to determine which technique works best on your models.
- The conflicting constraints are identified in the XA log file.

public…………void setConflictAnalysisOff( )

- Turns of constraint conflict analysis on infeasible LP models.

public..………..void setMaximizeObjective( )

- set the optimization sense to maximization.
- method must be invoked prior to calling solve( ) .

Optimizer Class

public..………..void setMinimizeObjective( )

- set the optimization sense to minimization.
- method must be invoked prior to calling solve( ) .

public..………..void setModelSize(   )

- Previous values of maxMem, maxRow, maxCol, maxNzo, rowNameSize, and colNameSixe are used to start a new model.
- invoking this method deletes any previously loaded data and begins a completely new model.
- see setModelSize(int maxMem) for more information.

public..………..void setModelSize(int maxMem)

- Resets the amount of memory available to XA Solver.
- Units of ‘maxMem’ are kbytes.
- All previous loaded model data is lost.
- Previous rowNameSize and colNameSize values are used.
- maxRow and maxCol values are estimated based upon the amount of requested memory..
- a value of zero instructs the Server to calculate this limit based upon the amount of memory defined by the Optimizer constructor.
- invoking this method deletes any previously loaded data and begins a completely new model.

public..………..void setModelSize(int rowNameSize, int colNameSize)

- ‘rowNameSize’ overrides the current rowNameSize.
- ‘colNameSize’ overrides the current colNameSize.
- see setModelSize(int maxMem int maxRow , int maxCol , int maxNzo) for more information.

public..………..void setModelSize(int maxMem , int maxRow, int maxCol, int maxNzo)

- sets limits on the number of rows, columns, and nonzeros for a model.
- see setModelSize(int maxMem) for more information.

public..………..void setModelSize(int maxMem , int maxRow, int maxCol, int maxNzo, int rowNameSize, int colNameSize)

- ‘rowNameSize’ overrides the current rowNameSize.
- ‘colNameSize’ overrides the current colNameSize.
- see setModelSize(int maxMem int maxRow , int maxCol , int maxNzo) for more information.

Optimizer Class

public……….void setMIPIncumbentAndTerminateOn( )

- If incumbent MIP solution is
  feasible then the LP optimizes the activities of the continuous columns
  and then terminates
  infeasible incumbent MIP solution are drop and XA uses branch and bounding
  techniques to solve.

  Note: This parameter provides a technique for always returning
  the same MIP column activities with optimized continuous
  columns activities.

  Useful when making repeated MIP solves with minor changes.

  After first MIP solve XA retains this solution in internal memory for this purpose.

public……….void setMIPIncumbentAndImproveOn ( )

- If incumbent MIP solution is
  feasible then the LP optimizes the activities of the continuous columns
  and then continues to improve the MIP solution.
  infeasible incumbent MIP solution are drop and XA uses branch and bounding
  techniques to solve.

  Note: Fast way to validate a MIP solution.
  Useful when making repeated MIP solves with minor changes.

  After first MIP solve XA retains this solution in internal memory for this purpose.

public……….void setMIPIncumbentOff ( )

- Turns off validating incumbent MIP solution.
- Defaults setting.

public..………..void setNumberOfProcessors( )

- MIP models are solved using all available processors (CPUs) on the Server machine.
- extra memory should be requested when multi-processor are used.
- results from multi-processor solves are not repeatable because the scheduling of threads is not a repeatable event.

public..………..void setNumberOfProcessors(int numberOfProcessors)

- see setNumberOfProcessors( ) for more information.
- sets the number of processor to use when solving MIP models.
- this method can be used on single processor machines and may yield good results first found integer solutions.

public..………..void setQPObjective(String colName, double qpObjCoef)

- sets ‘colName’ quadratic objective coefficient value to ‘qpObjCoef’.
- quadratic objective coefficient term to ‘qpObjCoef’ * colName * colName.
- requires Quadratic Programming license.

Optimizer Class

public...........void setQPObjective(String colName1 , String colName2 , double qpObjCoef)

- sets quadratic objective coefficient term to ‘qpObjCeof’ * colName1 * colName2 .

public...........void setRowFix(String rowName , double fixedActivityValue)

- the activity of ‘rowName’ (left-hand-side) is set equal to (fixed at) ‘fixedActivityValue’ value during the solving process.
- There is no deviation from this value.

public...........void setRowFree(String rowName)

- the activity of ‘rowName’ is bound between -infinity and +infinity.

public...........void setRowMax(String rowName , double maxBound)

- sets the upper bound of ‘rowName’ (left-hand-side value) to ‘maxBound’.
- in the absents of any upper bound specification ‘rowName’ default upper bound value is +infinite. This default can be changed by setCommand(“Set UBRow …”).

public...........void setRowMin(String rowName , double minBound)

- sets the lower bound value of ‘rowName’ (left-hand-side value) to ‘minBound’.
- in the absents of any lower bound specification, ‘rowName’ default lower bound value is +infinite. This default can be changed by setCommand(“Set LBRow …”).

public...........void setRowMinMax(String rowName , double minBound , double maxBound)

- sets the lower bound of ‘rowName’ (left-hand-side value) to ‘minBound’.
- sets the upper bound of ‘rowName’ (left-hand-side value) to ‘maxBound’.
- ‘minBound’ must be less than or equal to ‘maxBound’.

public…………void setTolerancePrimal(double tolerance)

- set the primal feasibility tolerance value.
- default value is 1.0e-7

public…………void setToleranceDual(double tolerance)

- set the dual feasibility tolerance value.
- default value is 5.0e-7

public…………void setUnicodeEncodingIdentifier(String encodingIdentifier)

- an appropriate ‘endcodingIdentifier’ string.
- must be called before the openConnection( ) method.
- defaults to “UTF8”.

public…………void setXAMessageWindowOff( )

- prevents creation of the “XA Message Window” on MS operating systems.
- mutes all output to the console.
- must be called before the openConnection( ) method.
- use setCommand(“Output c:\\xa.log”) if you wish to capture any XA messages.

Optimizer Class

public..………..void solve( )

- solve the model.
- returns the results of the optimization process in getModelStatus( ) and getSolverStatus( ). See Appendix B for more information.
- Begin calling callBack methods.

All methods throw XAException with a message and return code value. The message can be retrieved with getMessage( ) and return code value (int) with getExceptionCode( ) methods. See Appendix A for explanation of getExceptionCode values.

MPSFileLoader Class

Reads an MPS formatted file and loads into Optimizer instance.

* public……………… MPSFileLoader(Optimizer myModel , String mpsFileName)

- constructor for the MPSFileLoader Class.
- ‘myModel’ Optimizer instance to load data.
- ‘mpsFileName’ is the path/filename of the file containing data in MPS format.

* public……………… MPSFileLoader(Optimizer myModel , String mpsFileName , int Control)

- constructor for the MPSFileLoader Class.
- ‘myModel’ Optimizer instance to load data.
- ‘mpsFileName’ is the path/filename of the file containing data in MPS format.
- ‘control’ variable (defaults to 0) is a bitmap which controls the setting of the following:
    - a) a value of zero (0) indicates
        - new model and mute all information listing.
    - b) a value of one (1) indicates
        - merge data with previous loaded data.
    - c) a value of two (2) indicates
        - list MPS section headers as processed. Minimizes amount of information listed and still has some resemblance of doing something.
    - d) a value of four (4) indicates
        - complete listing of MPS file line by line.

* public……………… MPSFileLoader(Optimizer myModel , String mpsFileName , int Control , int maxRow)

- constructor for the MPSFileLoader Class.
- ‘myModel’ Optimizer instance to load data.
- ‘mpsFileName’ is the path/filename of the file containing data in MPS format.
- ‘control’ variable (defaults to 0) is a bitmap which controls the setting of the following:
    - e) a value of zero (0) indicates
        - new model and mute all information listing.
    - f) a value of one (1) indicates
        - merge data with previous loaded data.
    - g) a value of two (2) indicates
        - list MPS section headers as processed. Minimizes amount of information listed and still has some resemblance of doing something.
    - h) a value of four (4) indicates
        - complete listing of MPS file line by line. This could be a lot of output.
- ‘maxRow’ sets the maximum number of rows in the model. Default value is 100,000 rows. This does not have to be the exact number but must be chosen to accommodate the largest size model.

RCCFileLoader Class

Reads an RCC formatted file and loads into Optimizer instance.

* public................. RCCFileLoader(Optimizer myModel , String rccFileName)

- constructor for the RCCFileLoader Class.
- 'myModel' Optimizer instance to load data.
- 'rccFileName' is the path/filename of the file containing data in RCC format.

* public................. RCCFileLoader(Optimizer myModel , String rccFileName , int Control)

- constructor for the RCCFileLoader Class.
- 'myModel' Optimizer instance to load data.
- 'rccFileName' is the path/filename of the file containing data in RCC format.
- 'control' variable (defaults to 0) is a bitmap which controls the setting of the following:
  - a) a value of zero (0) indicates
    - new model and mute all information listing.
  - b) a value of one (1) indicates
    - merge data with previous loaded data.
  - c) a value of two (2) indicates
    - complete listing of RCC file line by line.

RT1Callback Class

Methods called by Optimizer callback

public…………void	feasibleIteration(RT1CallbackInfo OptimizerInformation)
public…………void	infeasibleIteration(RT1CallbackInfo OptimizerInformation)
public…………void	integerNodeGenerated(RT1CallbackInfo OptimizerInformation)
public…………void	integerSolutionFound(RT1CallbackInfo OptimizerInformation)
public…………void	solverComplete(RT1CallbackInfo OptimizerInformation)
public…………void	startingToSolve(RT1CallbackInfo OptimizerInformation)

RT1CallbackInfo Class

Methods available in your RT1Callback Routine.

public………double	getAmtInf( ) … Primal infeasibility at current iteration.
public………double	getBarrierDual( ) … Interior Point Algorithm dual value.
Public………double	getBarrierPrimal( ) … Interior Point Algorithm primal value. Both primal and dual values converge to the same value.
public………double	getBestPossibleSolution( ) … *Best possible optimal integer solution value. (1)
public………..…int	getBranchIn( ) … Column selected to enter the basis replacing getBranchOut column. (1)
public………..…int	getBranchOut( ) … Integer column with fractional value to 'integerize'. (1)
public………..…int	getBranchOutOpposite( ) … Column selected to enter the basis on opposite branch. (1)
public………double	getDj( ) … Dual value of entering column during simplex iteration.
public………..…int	getIn( ) … Column selected to enter the basis.
public………..…int	getIntFrac( ) … Number of fractional integer columns. (1)
public………double	getIntSol( ) … Best Integer solution available. (1)
public………..…int	getIterations( ) … Number of iterations.
public………double	getLimitSearch( ) … Current LimitSearch value. (1)
public………..…int	getNodesInfo( ) … Number of branch and bound nodes generated. (1)
public………..…int	getNumInf( ) … Number of infeasible columns.
public………..…int	getNumSol( ) … Number of integer solution found. (1)
public………double	getObj( ) … Value of the objective function.
public………..…int	getOut( ) … Column selected to leave the basis.
public………..…int	getType( ) … Reason for callback.
public…………void	setTerminate( ) … Terminate the solver.

(1) **Values available during the branch and bound process only.**

Appendix A ----- getExceptionCode

Return Code (getExceptionCode)	Meaning
0	No XA Problems…Exception thrown by your code.
2	Invalid XA Command Line Parameter.
4	Invalid character in rowName or colName. Use setUnicodeEncodingIdentifier to change encoding identifier.
6	Maximum row limit (rowMax) exceeded.
8	Maximum column limit (colMax) exceeded.
10	Maximum nonzero limit (maxNzo) exceeded.
12	Problem need more memory to solve model (maxMem).
14	Data consistency in model.
16	Integer node table overflow.
18	Solver failure.
20	Callback routine failure.
22	Missing or invalid filename.
24	Unexpected results return from Server.
26	XA Initialization failure.
28	Copy protection device missing.
30	General failure…see getMessage() for explanation.
32	MPS/RCC file format error.
34	Unable to write to filename.
36	Unable to read from filename.
99	Unexpected error condition.
1000	No XA Servers running.
1002	All XA Servers are busy.
1010	Lost i/o connection with Server.
1020	Solve method interrupt.
1030	Lost i/o connection in Callback Routine.

Appendix B - Solver Status Codes.

Code	getModelStatus
1	Optimal (Integer) Solution.
2	Integer Solution (not proven the optimal integer solution).
3	Unbounded solution.
4	Infeasible solution.
5	Callback function indicates Infeasible solution.
6	Intermediate infeasible solution.
7	Intermediate non-optimal solution.
9	Intermediate Non-integer solution.
10	Integer Infeasible.
13	Workspace size exceeded. Increase memory request in XAINIT call.
32	Integer branch and bound process currently active, model has not completed solving.
99	Currently solving model, model has not completed solving.

Note: Integer problems return a Model Status code of 1 if the optimal integer solution is found. If XA has not proven that it’s integer solution is optimal, then a Model Status code of 2 is returned.

Code	getSolverStatus
1	Normal Completion.
2	Iteration Interrupt.
3	Resource Interrupt, like time limit exceeded.
4	Terminated by User, probably a Cntrl Z.
8	Node Table Overflow.
10	Solver Failure.

Appendix C - getRc

Return Code	Message
2	XAINIT XACLP XAMPSI Successful
4	XARCCI/M/R XADBFI/M XALOAD Successful
6	XASOLV Successful
8	XAUNDO Successful
10	XADONE Successful
101	Allocation amount in XAINIT must be greater than or equal to 0. Make sure argument is a positive long_int integer.
102	Allocation amount in XAINIT must be greater than or equal to 0. Make sure argument is a positive long_int integer.
103	Memory allocation error, no memory available.
104	Memory allocation error, requested memory not available.
110	XACLP called out of sequence.
111	XADONE called out of sequence.
112	XASOLV called out of sequence.
113	XAUNDO called out of sequence.
114	XAACT,XADUAL,XAACTC,XADUALC,XABIA,XACBIA XAVAR, XAANAL, XAANALA called out of sequence
115	XALOAD called out of sequence.
116	XAMPSI, XADBFI, XARCCI called out of sequence.
117	XAOK called out of sequence.
118	XARPRT called out of sequence.
122	BXA.DLL not in path. Either move BXA.DLL to a directory in your path statement, or add the directory BXA.DLL is located in to your path statement in your autoexec.bat. You will have to restart Windows and possibly your computer to correct your PATH command.
124	Wrong version of BXA.DLL or your version has been damaged.
126	XA.... routine called after XADONE.
128	Restart Windows, you are having programming difficulties and it has caught up with you.
132	In call to XART1, the XAInfo structure (XAInfoSize value) size is incorrect. Check to see that dataarea.XAInfoSize = sizeof(XAInfo)
134	In call to XART2, the XAMsgLine structure (XAMsgSize value) size is incorrect. Check to see that dataarea.XAMsgSize = sizeof(XAMsgLine)
136	In call to XAMODEL, the XAModel structure (XAModel Size value) size is incorrect. Check to see that dataarea.XAModel Size = sizeof(XAModel)

Return Code	Message
204	XACLP is passed a command line parameter which expects No or Yes as a value.
205	XACLP is passed a misspelled command line parameter.
207	XACLP is passed an unreasonable command line parameter value.
208	XACLP is passed a PAGESIZE, TMARGIN, or BMARGIN command which is incompatible with each others value.
209	XACLP is passed a LINESIZE or LMARGIN command which is incompatible with each others value.
210	XACLP is passed a command line which ends prematurely.
211	XACLP is unable to process the OUTPUT command line parameter value.
214	XACLP is passed a FIELDSIZE or DECIMALS command which is incompatible with each others value.
300	Column number out of range, or misspelled row or column name.
301	Too many or zero rows in problem, increase ***maxrow.***
302	Too many or zero columns in problem, increase ***maxcol***.
303	Free memory error, XA could not release it's memory. Probably XA data storage area has been clobbered.
304	Column lower bound > upper bound.
305	Row lower bound > upper bound.
306	Problem too large for available memory, increase ***use*** parameter in your call to XAINIT.
307	Too many nonzeros, increase ***maxnonzero.***
308	Row number out of range.
309	Duplicate column and/or row name.
310	Arrays overlap. Run XAOK to identify overlapping arrays.
312	Bad ***colptr*** values. Run XAOK to locate problem.
314	Unreasonable ***rownos*** values. Run XA to locate problem.
316	Duplicate ***rownos*** value for the same column.
318	Unreasonable technology coefficient in ***a*** array.
320	Unreasonable value in ***status*** array.
322	Unreasonable value in ***priority*** array, check XASETA call.
324	Unreasonable value in ***increment*** array, check XASETA call.
326	Semi-continuous type 1 or type 2 column missing lower bound.
328	Column can not have both semi-continuous type 1 and type 2 at the same time.
330	Unrecognized line in MPS file.
400	MPS formatted file is missing NAME line. The NAME must start in column 1 and be the first line in the file.
402	Incomplete MPS file. File probably truncated.

Return Code	Message
404	Row relationship error. Rerun with LISTINPUT YES.
406	Expecting a number. Rerun with LISTINPUT YES.
408	Bound relationship error. Rerun with LISTINPUT YES.
410	Split Column declaration. All rows a column intersects must be grouped together. Rerun with LISTINPUT YES.
420	Column has a Power sequence with a negative lower bound, either correct lower bound or add XA_LBBASE status setting.
600	Misspelled or missing DBF table filename. Check for missing or incorrect path.
602	Error reading DBF table.
604	Special row and column name appear together.
606	Row increment setting; increment settings only apply to integer columns.
608	Row priority setting, priority settings only apply to integer columns.
610	Row field name not found in DBF table.
612	Column field name not found in DBF table.
614	Coefficient field name not found in DBF table.
616	No data available in DBF table.
700	No data previously loaded for solving.
702	Row name size not equal to previously loaded name size.
704	Column name size not equal to previously loaded size.
706	Specified name size does not match row or column name size.
900	Unknown error.
901	The first argument in the call to this function must be a pointer to the XA Environment Structure XAOSL. If the pointer address is correct then the XAOSL Structure data is corrupted/destroyed. You must preserve the XAOSL Structure data from the time you call XAINIT until you call XADONE.
9xx-919 xx=02 thru 19	Argument xx has a unreasonable value. This can also occur when you leave off an argument. Check calling sequence or rerun with SET DEBUG YES to help identify arguments.
930	XA is solving an integer programming problem. A node is generated splitting the feasible region of each integer variables. The default maximum number of nodes is sqrt(number of Integers variables) + number of 0/1 and semi-continuous columns + 4000 This default settings is too small. Use the XACLP command line parameter SET MAXNODES # to increase.
991	Maximum number of concurrent users exceeded.
992	Maximum number of processors exceeded.

Return Code	Message
993	XA DLL filename does match.
995	Expiration date exceeded. Call SST.
996	Missing or invalid c:\xa.lic file. Call SST.
997	Email c:\xa.lic to SST for activation.
998	Missing hardware dongle, call SST.
999	Your code has clobbered XA’s internal memory. Check and make sure you do not have any uninitialized pointers, arrays are malloc’ed with the proper lengths, ...; the problem occurs in your code that was executed between the last two XA.... function calls.
	Warning Message all begin with 2xxxx.
20001	Ranging a free/null row with MPS file.
20030	Row name referenced in COLUMNS section is undefined.
20040	Column has no technological coefficients, XA is fixing with zero primal activity.
20045	Column name referenced in BOUNDS section is undefined.
20050	Unable to save 'advance basis' solution. Disk is probably full or you do not have write access to it's disk. Relocate this file on another disk with more space ore remove unnecessary files from disk.
20060	A free column only appears in a FREE row. Use the XACLP command line parameter SET MPSXCOMPATIBLE YES to FIX this column to zero. If you don't XA may find your problem is unbounded.
20080	Duplicate row names in ROWS section.
20090	Unrecognized line in MPS formatted file.