CAPE-OPEN

Delivering the power of component software
and open standard interfaces
in Computer-Aided Process Engineering

Open Interface Specification:

Optimisation Interface

www.colan.org

ARCHIVAL INFORMATION

Filename	Optimisation Interface Specification.doc
Authors	CO-LaN consortium
Status	Public
Date	August 2003
Version	version 2
Number of pages	62
Versioning	version 2, reviewed by Jean-Pierre Belaud, August 2003
	version 1, February 2002
Additional material	
Web location	www.colan.org
Implementation specifications version	CAPE-OPENv1-0-0.idl (CORBA) CAPE-OPENv1-0-0.zip and CAPE-OPENv1-0-0.tlb (COM)
Comments	

IMPORTANT NOTICES

Disclaimer of Warranty

CO-LaN documents and publications include software in the form of *sample code.* Any such software described or provided by CO-LaN --- in whatever form --- is provided "as-is" without warranty of any kind. CO-LaN and its partners and suppliers disclaim any warranties including without limitation an implied warrant or fitness for a particular purpose. The entire risk arising out of the use or performance of any sample code --- or any other software described by the CAPE-OPEN Laboratories Network --- remains with you.

CO-LaN is a non for profit organization established under French law of 1901.

Trademark Usage

Many of the designations used by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in CO-LaN publications, and the authors are aware of a trademark claim, the designations have been printed in caps or initial caps.

Microsoft, Microsoft Word, Visual Basic, Visual Basic for Applications, Internet Explorer, Windows and Windows NT are registered trademarks and ActiveX is a trademark of Microsoft Corporation.

Netscape Navigator is a registered trademark of Netscape Corporation.

Adobe Acrobat is a registered trademark of Adobe Corporation.

SUMMARY

ACKNOWLEDGEMENTS

CONTENTS

LIST OF FIGURES

1. Introduction

This document aims at defining a standard and unified software interface for numerical solvers for both Mixed Integer Linear Programs (MILPs) and Mixed Integer Nonlinear Programs (MINLPs).

1.1 Mixed Integer Linear Programs

Mixed integer linear programming (MILP) problems involve the minimisation or maximisation a linear objective function subject to linear constraints. The optimisation may involve both continuous and discrete (integer-valued) decision variables.

MILPs arise quite frequently in process engineering applications such as:

- supply chain optimisation
- multipurpose plant scheduling
- refinery scheduling
- synthesis of heat exchanger networks

and many others. MILP problems also occur as sub-problems in the solution of mixed integer nonlinear programming problems (see section 1.2 below).

An MILP can be formulated mathematically as follows:

$$\min_{x,y} ax + by$$

$$l \le Ax + By \le u$$

$$xL \le x \le xU$$

$$yL \le y \le yU$$

where x is a vector of n continuous variables, y is a vector of n' integer variables, a, xL, xU are real n-vectors, yL, yU are integer n'-vectors, b is a real n'-vector, A is an $m \times n$ real matrix, B is an $m \times n'$ real matrix and l, u are real m-vectors.

Two important special cases of MILPs are:

- Linear programming (LP) problems.

In this case, all optimisation decision variables are continuous, i.e. $n'=0$. Such problems also appear frequently in practice, either on their own or as sub-problems in the solution of MILP problems.

- Purely Integer Programming (IP) problems.

In this case, all optimisation decision variables are discrete, i.e. $n=0$.

1.2 Mixed Integer Nonlinear Programs

Mixed integer nonlinear programming (MINLP) problems involve the minimisation or maximisation a nonlinear objective function subject to nonlinear constraints. The optimisation may involve both continuous and discrete (integer-valued) decision variables.

MINLPs arise in many process engineering applications, including:

- process synthesis
- process design
- product design

and others.

An MINLP can be formulated mathematically as follows:

$$\min x,y \;\; f(x, y)$$

$$l \leq g(x, y) \leq u$$

$$xL \leq x \leq xU$$

$$yL \leq y \leq yU$$

where x is a vector of n continuous variables, y is a vector of n' integer variables, xL, xU are real n-vectors, yL, yU are integer n'-vectors, f: $\mathbf{R}n \times \mathbf{Z}n'\gamma\ \mathbf{R}$ is a (possibly) nonlinear function, g: $\mathbf{R}n \times \mathbf{Z}n'\gamma\ \mathbf{R}m$ is a list of m (possibly) nonlinear functions and l, u are real m-vectors.

A very important special case of MINLPs is that of nonlinear programming (NLP) problems. In this case, all optimisation decision variables are continuous, i.e. n'=0. Most operational (as opposed to design) optimisation problems in process engineering can be formulated as NLPs. NLPs also occur as sub-problems in the solution of MINLP problems.

1.3 Generalized Mixed Integer Nonlinear Programs

As can be deduced from sections 1.1 and 1.2, MILPs can be viewed just a special case of MINLPs where both the objective function and the constraints are linear. Thus, it would appear to suffice defining a standard software interface for MINLPs of the mathematical form described in section 1.2.

Itis worth noting, however, that numerical solvers for nonlinear problems (including NLPs and MINLPs) are often capable of exploiting partial linearity in the problem being solved. This indicates that the linear and nonlinear parts of objective function and constraints should be treated separately.

Moreover, albeit possible in principle, the representation of a linear problem in terms of a more nonlinear formulation may entail some inefficiencies from the computational point of view.

In view of the above, we employ the following definition of a *generalized MINLP*:

$$\min x,y \;\; ax + by + f(x, y)$$

$$l \leq Ax + By + g(x, y) \leq u \qquad (1)$$

$$xL \leq x \leq xU$$

$$yL \leq y \leq yU$$

where x is a vector of n continuous variables, y is a vector of n' integer variables, a, xL, xU are real n-vectors, yL, yU are integer n'-vectors, b is a real n'-vector, f: $\mathbf{R}n \times \mathbf{Z}n'\gamma\ \mathbf{R}$ is a nonlinear function, A is an $m \times n$ real matrix, B is an $m \times n'$ real matrix, g: $\mathbf{R}n \times \mathbf{Z}n'\gamma\ \mathbf{R}m$ is a list of m nonlinear functions and l, u are real m-vectors.

In fact, for ease of manipulation at both the mathematical and the software levels, it is better to group both continuous and integer variables within a single vector x, rewriting the above as:

$$
\begin{aligned}
&\min_{x} \; ax + f(x) \\
&l \le Ax + g(x) \le u \\
&x^L \le x \le x^U \\
&x_i \in \mathrm{Z}, \forall i \in I
\end{aligned}
\qquad (2)
$$

where x is a vector of n variables , a, xL, xU are real n-vectors, f: **R***n* γ **R** is a nonlinear function, A is an m x n real matrix, g: **R***n* γ **R***m* is a list of m nonlinear functions and l, u are real m-vectors.

We note that:

- the variables in x are characterised by an index i *(=1,..,n)* and are bounded between given lower and upper bounds x^L and x^U respectively;
- some of the variables x_i are restricted to take integer values; these integer variables are identified via an index set $I \subseteq \{1,..,n\}$;
- all constraints are expressed as double inequalities in the form *lower bound* ≤ *function* ≤ *upper bound*, and are indexed over the discrete domain 1,..,m.

Many different variations of this form exist (e.g. involving *equality* constraints of the form *function* = *constant* rather than inequalities, as shown above). However, they are all completely equivalent mathematically and can be transformed to each other via usually trivial mathematical operations.

1.4 Scope and Overview of Interfaces

The interface considered in this document is aimed at general numerical solvers for MILPs and MINLPs. It also accommodates special cases of these (cf. sections 1.1 and 1.2) such as solvers for:

- linear programming problems (LPs)
- integer (linear) programming problems (IPs)
- nonlinear programming problems (NLPs).

All these solvers operate on a formal software description of the problem in terms of an *MINLP object*. The latter corresponds to the generalized MINLP mathematical formulation given by equations (2).

A typical scenario for a client program using the interfaces defined in this document would be as follows:

- The client starts by constructing an MINLP object describing the process engineering problem to be solved.
- Once the client constructs the MINLP object, it can then pass it to a *Solver Manager* corresponding to a particular numerical solver to create an *MINLP system*. The latter represents a combination of the problem to be solved (as embodied in the MINLP object itself) and the numerical solution code (as incorporated within the Solver Manager).
- Once the MINLP system object is constructed, the client may invoke its *Solve* method to effect the solution of the problem of interest.

- The client may then retrieve the final values of the variables by invoking the appropriate method of the MINLP object.

The above scenario is entirely analogous to that adopted by earlier CAPE-OPEN specifications for numerical solvers. In particular:

- no attempt is made to standardize the manner in which the MINLP object is actually constructed[1];

- consequently, the MINLP object interface is focussed on allowing a numerical MILP/MINLP solver to obtain all the information that is necessary to solve the problem;

- the MINLP Solver Manager and MINLP System interfaces integrate the *Common Interfaces* politics.

1.5 Structure of this document

The rest of this document is structured as follows:

- Section 2 presents an analysis of the requirements of the interfaces, via an example and a set of use cases, concluding with a proposed set of classes for tackling the problem.

- Section 3 reviews in more detail the basic concepts which we will need to take into account in defining the methods of the interfaces and their arguments.

- Section 3.3 presents an interface diagram listing these methods, and at the same time showing the relationships between instances of the various classes.

- Sections 3.4-3.6.3 present the proposed interfaces in full detail, including semantic explanations of what each method is required to achieve.

- Section 4 presents COM and CORBA IDL files corresponding to these interfaces.

---

[1] For example, the CAPE-OPEN specifications for numerical solvers for nonlinear algebraic equations and differential-algebraic equations rely on the problem being described in terms of an Equation Set Object (ESO). CAPE-OPEN does *not* specify how an ESO is constructed.

2. Requirements

2.1 Textual requirements

The mathematical description given in section 1.1 presents a "numerical solver's view" of MINLP problems in a very general form. For many applications of practical interest, the number of variables and constraints can be quite large. Thus, problems involving hundreds of thousands of variables (including several thousands of integer variables) and constraints can be solved using existing commercial software.

One key characteristic of the "flat" formulation is that it is completely unstructured: essentially, it makes no distinction between different variables (or constraints) other than treating them as different elements of the same vector. This uniformity certainly facilitates the design and implementation of general numerical codes.

However, from the point of view of the mathematical modeller[2], most representations of MINLP problems arising in practice tend to differ from this in that variables and constraints are naturally grouped into a relatively small number of sets, each of which represents a distinct physical characteristic of the system being studied and its natural or desired behaviour.

The above indicates that there is a certain divergence between the views and requirements of the numerical solver on one hand and of the mathematical modeller on the other. It would be auspicable to try to define an interface which allows the expression of MINLP problems in a way as close to their "physical" representation as possible, while at the same time permitting numerical solvers to operate in the manner natural to them, i.e. the flat representation of section 1.1. This, in fact, had been the object of the first draft of this document. However, it seems too complex an undertaking to be carried out properly at this stage. Thus we now merely *indicate* that it is desirable to have such an interface, but we leave the implementation details outside of the scope of this documents. We will mention the class that should take care of this task but will not specify the data structures or the methods. We will instead focus on the flat form of the MINLP as a necessary "in-between" step between the structured MINLP and the numerical solver.

In the rest of this section, after presenting an example of such a structured problem representation, we will express, in terms of use cases, the essentials of how we might construct and manipulate it through an interface.

Basic Interactions:

A MINLP involves concepts like continuous and integer variables, linear and nonlinear constraints which may or may not involve black-box functions and conditional constraint format. Providing a construction class that takes care of this whole host of features is a huge undertaking. We will therefore limit our discussion to a base class for the representation of a MINLP that gives enough flexibility to software modules so that they can interact with CO-compliant solvers, whereas the actual construction of the MINLP is left as a proprietary implementation.

Thus, we are led to identify the following typical sequence of actions:

(i) Construct an MINLP object which inherits from the base class we describe below (the construction of the MINLP is outside the scope of this document).

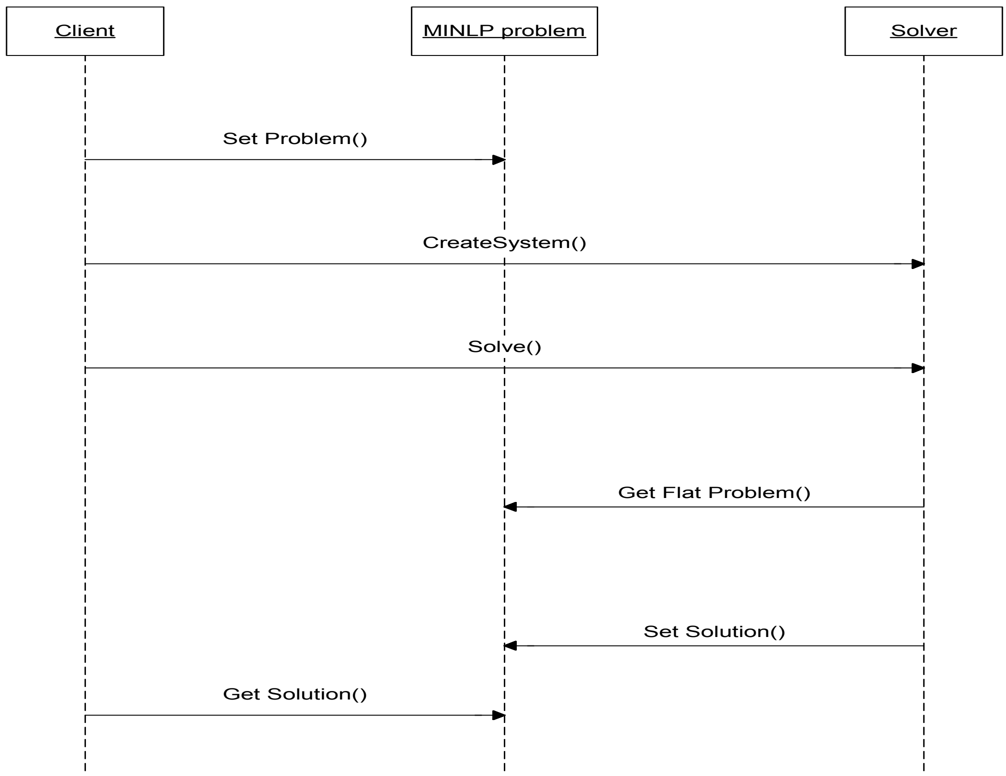
(ii) Solve a given MINLP with a selected solver.

(iii) Obtain solution information.

---

[2] For example, an engineer formulating a particular problem.

The diagram below describes the basic interactions between the various software components.



2.2 Use Cases

In this section we present the conclusion of the previous section more formally using the UML (Use Case methodology).

2.2.1 Actors

We begin the formal UML segment of this document with a catalogue of the various Actors who will participate in the use cases.

Note on relationship to CAPE-OPEN Solvers Use Cases: we have reproduced here (with minor adaptations) some common functionality which relates to the configuration of solvers defined in the CAPE-OPEN project.

Optimisation Builder

The person who

- sets up the structured MINLP,
- chooses the numerical methods for optimising the problem.

This person hands over a working optimisation setup to the [Flowsheet User]. The Flowsheet Builder can act as a [Flowsheet User].

Optimisation User

The person who uses an existing optimisation setup. This person will put new data into the MINLP rather than change its structure. He may also choose numerical methods for optimising the problem.

Optimisation Executive

The client code (e.g. scheduling software) which has a MINLP (or MINLPs) to solve.

MINLP Solver

Another piece of software for solution of MINLPs.

2.2.2 List of Use Cases

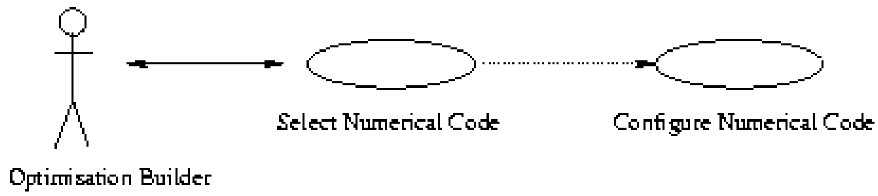
2.2.3 Use Cases maps

2.2.4 Use Cases

This subsection lists all the Use Cases that are relevant for the Solver Interfaces.

Solver Selection, Instantiation and Configuration

Use Case Diagrams



UC-001: SELECT NUMERICAL CODE

Actors: <Optimisation Builder> Executive>, <Solver Manager> Classification: <Solvers Use Cases>, <General Purpose Use Cases>, <Optimisation Context Use Cases> Status: Pre-conditions: <There must be at least one registered MINLP solver> <A complete MINLP problem has been defined> Flow of events: *Basic Path:* The Optimisation Builder requests the Optimisation Executive to carry out an optimisation. If no MINLP solver has yet been configured, the Optimisation Executive asks the Solver Manager for the list of the MINLP codes available on the system. The Optimisation Executive then displays this list to the Optimisation Builder who selects the code to be used. The [Configure Numerical Code] Use Case is then applied.

Post-conditions: <selection succeeded>

Exceptions: <selection failed>

Subordinate Use Cases:

[Configure Numerical Code]

UC-002: CONFIGURE NUMERICAL CODE

Actors: <Solver Manager>

Classification: <Solvers Use Cases>, < Context Use Case>

Status:

Pre-conditions:

<A solver has been selected>

Flow of events:

Basic Path:

The Solver Manager asks the Solver for a list of its parameters: each of which will have a name, a type, a default value and a valid range (for real values).

It *may* then provide this list to the user to give him/her the opportunity to override the default values.

Post-conditions: <Parameter list obtained>

Exceptions: <Required parameter missing>

Subordinate Use Cases:

None

MINLP Creation and Solution

Create and Specify MINLP

The implementation of this step is not covered by the present document. It suffices that the created MINLP inherits from the base MINLP class described below.

UC-003: SOLVE MINLP

Actors: <Optimisation Executive Solver>, <MINLP Construction Tool>

Classification: <Solvers Use Cases>, <Context Use Case>

Status:

Pre-conditions:

- An MINLP object has been created and prepared for solution.
- An MINLP solver has been configured.

Flow of events:

Basic Path:

The Optimisation Executive supplies a constructed MINLP object to the Solver and requests its solution. The Solver obtains information on the MINLP from the MCT and solves the problem, altering the MINLP's current variable values accordingly.

Post-conditions: <MINLP object ready for solution>

Exceptions: None

Subordinate Use Cases:

None

UC-004: OBTAIN SOLUTION

Actors: <Optimisation Executive Construction Tool>

Classification: <Solvers Use Cases>, <Context Use Case>

Status:

Pre-conditions:

A MINLP has been constructed and solved.

Flow of events:

Basic Path:

The Optimisation Executive asks the MCT for the current values of a solved MINLP.

Post-conditions:

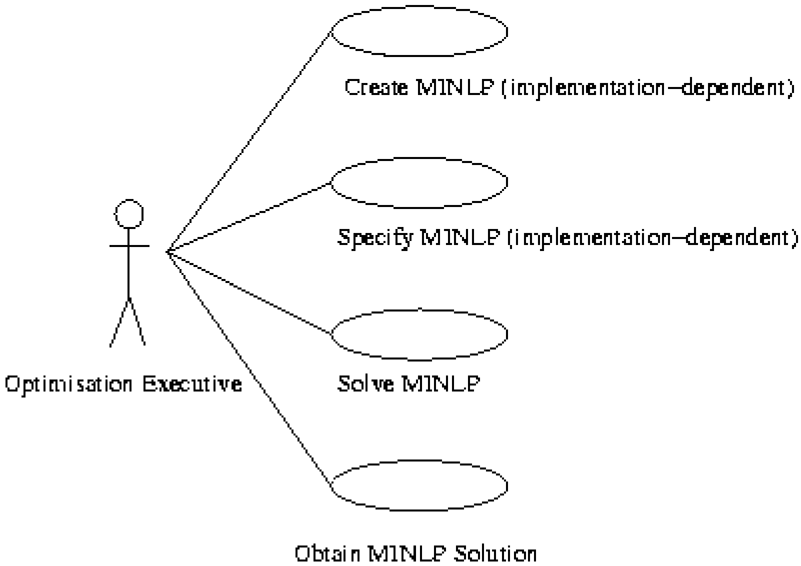
<The Optimisation Executive has the solution values for the variables in the MINLP>

Exceptions: None

Subordinate Use Cases:

None

The following Use Case Diagram shows the relationships of the above Use Cases.



2.3 Sequence diagrams

3. Analysis and Design

3.1 Overview and basic concepts

3.1.1 Object Classes

Following on from the approach used with Numerical Solvers in CAPE-OPEN, we will recognise two major classes of objects, each with its own interface:

1. The MINLP object

An MINLP object is a software representation of an MINLP problem. The corresponding interface(s) must provide the following functionality:

- It must allow access to all information required by typical solvers for them to solve to MINLP.
- It must provide client codes access to all its information.

2. The MINLPSystem object

This is formed by the combination of an MINLP object (see above) with a code ("solver") for the numerical solution of MINLP problems. The corresponding interface must provide the following functionality:

- It must allow the behaviour of the solver to be configured via the specification of any algorithmic parameters that the solver may support.
- It must permit the solution of the MINLP.

3.1.2 The MINLP Object

An MINLP object is characterised by:

- its variables;
- its constraints;
- its objective function.

A variable in an MINLP is characterised by:

- a variable index starting from 1;
- whether the variable is continuous or discrete;
- the variable lower and upper definition bounds.

A constraint (in the inequality form $l \leq Ax + g(x) \leq u$) is characterised by:

- a constraint index starting from 1;
- whether the constraint is only linear or also has a nonlinear part;
- the constraint lower and upper bounds;

- the linear part of the constraint;
- the nonlinear part of the constraint.

The linear part of a constraint is characterised by:

- the list of non-zero coefficients;
- a list of variable indices to which the items in the previous list refer to, in the same order as the previous list.

The nonlinear part of a constraint is characterised by:

- a function that evaluates the nonlinear part of the constraint;
- (optionally) functions that evaluate the first partial derivatives of the nonlinear part with respect to each of the problem variables.

The objective function is characterised by:

- whether the problem is a minimization or a maximization problem;
- the linear part of the objective function;
- the nonlinear part of the objective function.

The linear and nonlinear parts of the objective function are characterised in exactly the same way as the linear and nonlinear parts of constraints.

3.2 Sequence diagrams

SQ-001 COMPONENT SEQUENCE DIAGRAM

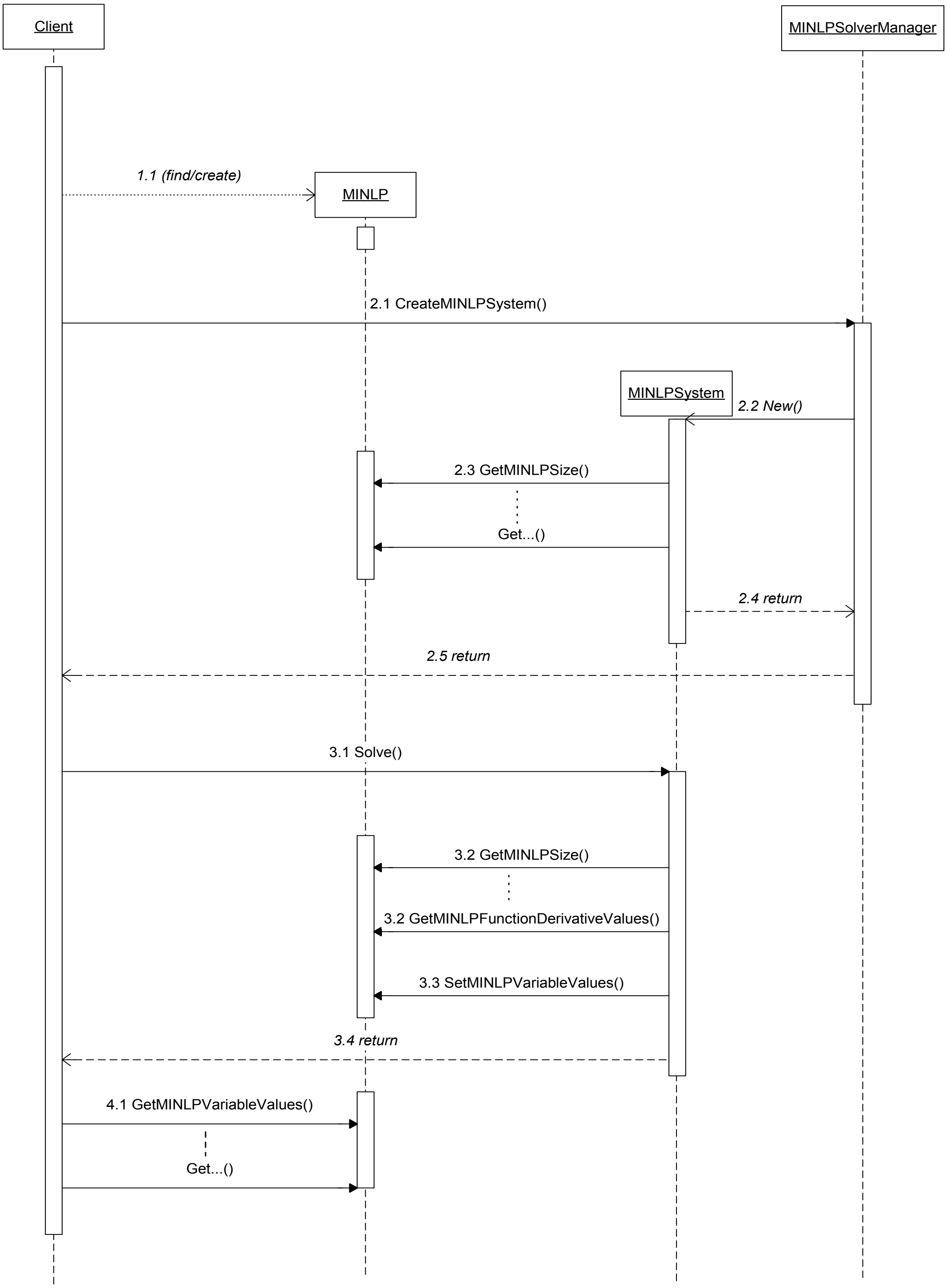


Figure 1 Sequence diagram

3.3 Interface diagram

The following figure presents the proposed interfaces in diagrammatic form.

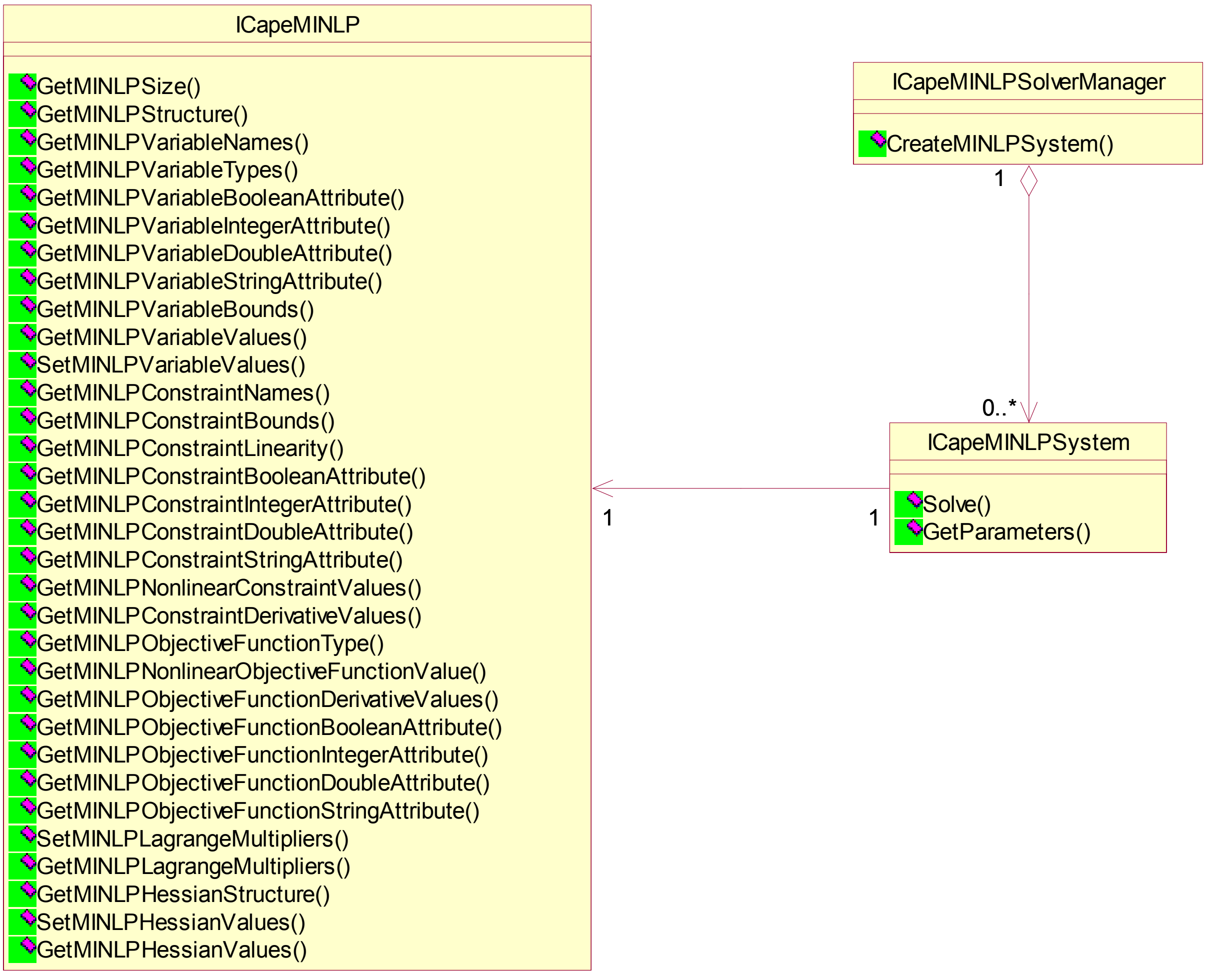


Figure 2 Interface diagram

3.4 State diagrams

None

3.5 Other diagrams

CP-001 COMPONENT DIAGRAM

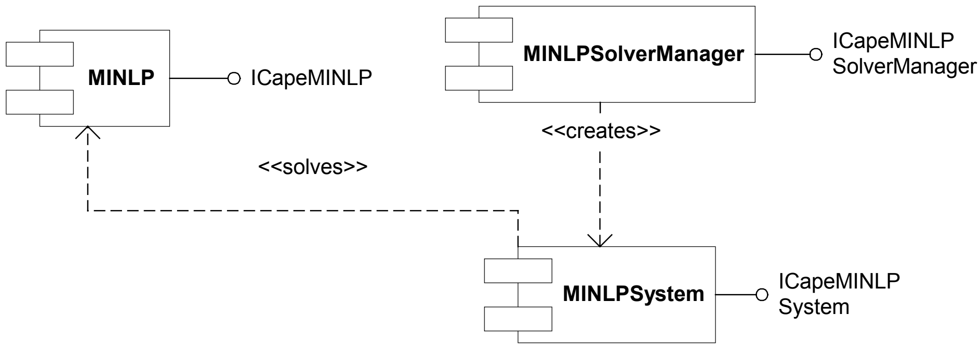


Figure 3 Component diagram

3.6 Interface descriptions

Argument Types

In defining the argument lists of the various methods, our general approach has been to use the simplest possible argument types, namely those defined in the Guidelines document[3]:

- CapeError
- CapeLong
- CapeDouble
- CapeBoolean
- CapeString
- CapeInterface
- CapeArrayLong
- CapeArrayDouble
- CapeArrayBoolean
- CapeArrayString

In addition, the following array type was also used:

- CapeArrayArrayLong: this contains a sequence of CapeArrayLong values, i.e. a sequence of sequences of Long values.

Finally, new enumerated data types were defined to represent valid types of objective function:

[3] CAPE-OPEN Interfaces Guidelines and Review: CO-MGT-Qs-20 (Version 1, November 1997).

1. CapeMINLPObjFunType = {MAX, MIN}

In one case, an Array type is also defined to hold sequences of these types.

Exceptions

This interface specification complies with the CO Error Common Interface. All the methods may raise the generic exception ECapeUnknown. Methods that accept input arguments can raise ECapeInvalidArgument. Methods used for the creation of objects can raise ECapeNoMemory and ECapeOutOfResources. The method Solve(), which invokes an external numerical solver, can also raise exceptions ECapeTimeOut and ECapeSolvingError.

The methods GetMINLPHessianStructure(), GetMINLPHessianValues(), SetMINLPHessianValues() can raise the purpose-defined exception ECapeHessianInfoNotAvailable.

Constants

We need to define +∞ and -∞ as special values in ICapeUtilityDefinitions.

3.6.1 ICapeMINLP

This interface to an MINLP object provides all the information required by a typical solver. The view of the MINLP corresponds to the mathematical description given by equations (2).

Interface Name	ICapeMINLP
Method Name	GetMINLPSize
Returns	--

Description

Get various integers defining the problem size.

Notes:

The number of variables nv includes both continuous and integer variables.

Arguments

Name	Type	Description
[out] nv	CapeLong	Total number of variables in MINLP
[out] niv	CapeLong	number of integer variables in MINLP
[out] nlv	CapeLong	number of variables in MINLP which only appear linearly
[out] nliv	CapeLong	number of integer variables in MINLP which only appear linearly
[out] nc	CapeLong	total number of constraints in MINLP
[out] nlc	CapeLong	number of linear constraints in MINLP
[out] nlz	CapeLong	number of non-zero elements in the matrix of MINLP
[out] nnz	CapeLong	number of non-zero elements in the Jacobian matrix of the nonlinear part of the MINLP constraints
[out] nlzof	CapeLong	number of variables having non-zero coefficients in the linear part of the MINLP objective function
[out] nnzof	CapeLong	number of non-zero first order derivatives in the nonlinear part of the MINLP objective function

Errors

ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPStructure
Returns	--

Description

Get information on the sparsity structure of the objective function and the constraints. This corresponds to one of the following, depending on the request issued by the client:

(i) the linear variable occurrences;

(ii) the nonlinear jacobian elements occurrences;

(iii) the union of the above structures.

Notes:

- The input parameter structuretype must be one of the following strings: "LINEAR", "NONLINEAR", "BOTH" depending on whether the client needs the linear structure, the nonlinear structure or a union of both.
- If option "BOTH" is specified, and a variable occurs both linearly and nonlinearly in the same constraint or in the objective function, then the variable appears as a *single* entry in the returned structure.
- The arrays rowindex and columnindex are both of length nlz, nnz or nlz+nnz depending on whether structuretype is "LINEAR", "NONLINEAR" or "BOTH".
- The array objindex is of length nlzof, nnzof or nlzof+nnzof according as to whether structuretype is "LINEAR", "NONLINEAR" or "BOTH".
- Constraints and variables in the MINLP are numbered starting from 1.

Arguments

Name	Type	Description
[in] structuretype	CapeString	Specifies whether returned structure should be of type (1), (2) or (3) (see "Description" above)
[out] rowindex	CapeArrayLong	List of integers containing the numbers of the constraints in the MINLP from which the nonzero elements originate
[out] columnindex	CapeArrayLong	List of integers containing the numbers of the variables in the MINLP from which the nonzero elements in constraints originate
[out] objindex	CapeArrayLong	List of integers containing the numbers of the variables in the MINLP from which the nonzero elements in the objective function originate

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPVariableNames
Returns	--

Description

Get names of MINLP variables corresponding to a specified list of indices, vids.

Notes:

The variable indices vids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] vids	CapeArrayLong	The indices of a subset of variables in the MINLP
[out] vnames	CapeArrayString	The *i*-th element of this sequence will contain the name of variable vids[*i*]

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPVariableTypes
Returns	--

Description

Get the types (i.e. continuous or integer) of variables corresponding to a specified list of indices, vids.

Notes:

The variable indices vids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] vids	CapeArrayLong	The indices of a subset of variables in the MINLP
[out] isinteger	CapeArrayBoolean	The i-th element of this sequence will be TRUE if variable vids[i] is integer and FALSE otherwise

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPVariableBooleanAttribute
Returns	--

Description

Get the boolean attributes of variables corresponding to a specified list of indices, vids.

Notes:

The variable indices vids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] vids	CapeArrayLong	The indices of a subset of variables in the MINLP
[in] attrib	CapeString	The name of the boolean attribute
[out] values	CapeArrayBoolean	i-th element of sequence contains the value of the attribute

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPVariableIntegerAttribute
Returns	--

Description

Get the integer attributes of variables corresponding to a specified list of indices, vids.

Notes:

The variable indices vids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] vids	CapeArrayLong	The indices of a subset of variables in the MINLP
[in] attrib	CapeString	The name of the integer attribute
[out] values	CapeArrayLong	i-th element of sequence contains the value of the attribute

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPVariableDoubleAttribute
Returns	--

Description

Get the double attributes of variables corresponding to a specified list of indices, vids.

Notes:

The variable indices vids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] vids	CapeArrayLong	The indices of a subset of variables in the MINLP
[in] attrib	CapeString	The name of the double attribute
[out] values	CapeArrayDouble	i-th element of sequence contains the value of the attribute

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPVariableStringAttribute
Returns	--

Description

Get the string attributes of variables corresponding to a specified list of indices, vids.

Notes:

The variable indices vids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] vids	CapeArrayLong	The indices of a subset of variables in the MINLP
[in] attrib	CapeString	The name of the string attribute
[out] values	CapeArrayString	i-th element of sequence contains the value of the attribute

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPVariableBounds
Returns	--

Description

Get lower and upper bounds of MINLP variables corresponding to a specified set of indices, vids.

Notes:

The variable indices vids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] vids	CapeArrayLong	The indices of a subset of variables in the MINLP
[out] LB	CapeArrayDouble	The i-th element of this sequence contains the lower bound of variable vids[i]
[out] UB	CapeArrayDouble	The i-th element of this sequence contains the upper bound of variable vids[i]

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPVariableValues
Returns	--

Description

Get current values of MINLP variables corresponding to a specified set of indices, vids.

Notes:

The variable indices vids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] vids	CapeArrayLong	The indices of a subset of variables in the MINLP
[out] values	CapeArrayDouble	The i-th element of this sequence contains the value of variable vids[i]

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	SetMINLPVariableValues
Returns	--

Description

Alter the values of a subset of variables in the MINLP.

Notes:

The variable indices vids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] vids	CapeArrayLong	The indices of a subset of variables in the MINLP
[in] values	CapeArrayDouble	The i-th element of this sequence contains the new value for variable vids[i]

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPConstraintNames
Returns	--

Description

Get names of a subset of constraints in the MINLP corresponding to a specified list of indices, cids.

Notes:

The constraint indices cids must be in the range 1,..,nc.

Arguments

Name	Type	Description
[in] cids	CapeArrayLong	The indices of a subset of constraints in the MINLP
[out] cnames	CapeArrayString	The i-th element of this sequence will contain the name of constraint cids[i]

Errors

ECapeInvalidArgument, ECapeUnkown

Interface Name	ICapeMINLP
Method Name	GetMINLPConstraintBounds
Returns	--

Description

Get lower and upper bounds of a subset of the constraints in the MINLP corresponding to a specified list of indices.

Notes:

The constraints indices cids must be in the range 1,...,nc.

Arguments

Name	Type	Description
[in] cids	CapeArrayLong	The indices of a subset of constraints in the MINLP
[out] LB	CapeArrayDouble	The i-th element of this sequence contains the lower bound of constraint cids[i]
[out] UB	CapeArrayDouble	The i-th element of this sequence contains the upper bound of constraint cids[i]

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPConstraintLinearity
Returns	--

Description

Obtain information on the linearity of a subset of the constraints in the MINLP corresponding to a specified list of indices, cids.

Notes:

The constraints indices cids must be in the range 1,...,nc.

Arguments

Name	Type	Description
[in] cids	CapeArrayLong	The indices of a subset of constraints in the MINLP
[out] islinear	CapeArrayBoolean	The i-th element of this sequence is TRUE if constraint cids[i]is linear, and FALSE otherwise

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPConstraintBooleanAttribute
Returns	--

Description

Get the boolean attributes of constraints corresponding to a specified list of indices, vids.

Notes:

The constraint indices cids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] cids	CapeArrayLong	The indices of a subset of constraints in the MINLP
[in] attrib	CapeString	The name of the boolean attribute
[out] values	CapeArrayBoolean	i-th element of sequence contains the value of the attribute

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPConstraintIntegerAttribute
Returns	--

Description

Get the integer attributes of constraints corresponding to a specified list of indices, vids.

Notes:

The constraint indices cids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] cids	CapeArrayLong	The indices of a subset of constraints in the MINLP
[in] attrib	CapeString	The name of the integer attribute
[out] values	CapeArrayLong	i-th element of sequence contains the value of the attribute

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPConstraintDoubleAttribute
Returns	--

Description

Get the double attributes of constraints corresponding to a specified list of indices, vids.

Notes:

The constraints indices cids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] cids	CapeArrayLong	The indices of a subset of constraints in the MINLP
[in] attrib	CapeString	The name of the double attribute
[out] values	CapeArrayDouble	i-th element of sequence contains the value of the attribute

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPConstraintStringAttribute
Returns	--

Description

Get the string attributes of constraints corresponding to a specified list of indices, vids.

Notes:

The constraint indices cids must be in the range 1,...,nv.

Arguments

Name	Type	Description
[in] cids	CapeArrayLong	The indices of a subset of constraints in the MINLP
[in] attrib	CapeString	The name of the string attribute
[out] values	CapeArrayString	i-th element of sequence contains the value of the attribute

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPNonlinearConstraintValues
Returns	--

Description

Obtain the values of the nonlinear parts of a subset of the MINLP constraints corresponding to a specified list of indices, cids.

Notes:

The constraints indices cids must be in the range 1,...,nc.

Arguments

Name	Type	Description
[in] cids	CapeArrayLong	The indices of a subset of constraints in the MINLP
[out] values	CapeArrayDouble	The i -th element of this sequence contains the value of the nonlinear part of constraint cids[i]

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPConstraintDerivativeValues
Returns	--

Description

Obtain the values of the first partial derivatives of a subset of the MINLP constraints (corresponding to a specified list of indices, cids) with respect to the variables.

The partial derivatives returned may be one of the following:

(i) the nonzero coefficients in the linear part of the specified constraints;

(ii) the nonzero partial derivatives of the nonlinear parts of the specified constraints, evaluated at the current variable values;

(iii) the union of the preceding vectors.

Notes:

- The input parameter structuretype must be one of the following strings: "LINEAR", "NONLINEAR", "BOTH" according as to whether the client needs the linear part of the constraints, the derivatives, or a union of both.
- If option "BOTH" is specified, and a variable occurs both linearly and nonlinearly in a constraint, then the value returned is the sum of:
- the corresponding linear coefficient, and
- the corresponding partial derivative of the nonlinear part of the constraint.
- The row and column indices of the elements of the vector can be obtained from method GetMINLPStructure as integer vectors rowindex and columnindex respectively.
- The constraints indices cids must be in the range 1,...,nc

Arguments

Name	Type	Description
[in] structtype	CapeString	Specifies whether returned structure should be of type (1), (2) or (3) (see "Description" above)
[in] cids	CapeArrayLong	The indices of a subset of constraints in the MINLP
[out] vals	CapeArrayDouble	A sequence of doubles containing the nonzero elements of the linear parts of the constraints, or of the Jacobian of the nonlinear parts of the constraints, or both

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPObjectiveFunctionType
Returns	--

Description

Determine whether the MINLP involves a maximisation or a minimisation calculation.

Notes:

Possible objective function types are max and min.

Arguments

Name	Type	Description
[out] otype	CapeMINLPObjFunType	Type of the objective function (min/max)

Errors

ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPNonlinearObjectiveFunctionValue
Returns	--

Description

Returns the value of the nonlinear part of the objective function.

Arguments

Name	Type	Description
[out] value	CapeDouble	The value of the nonlinear part of the objective function evaluated at the current variable values

Errors

ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPObjectiveFunctionDerivativeValues
Returns	--

Description

Obtain the values of the first partial derivatives of the objective function with respect to the variables.

The partial derivatives returned may be one of the following:

(i) the nonzero coefficient in the linear part of the objective function;

(ii) the nonzero partial derivatives of the nonlinear parts of the objective function, evaluated at the current variable values;

(iii) the union of the preceding vectors.

Notes:

- The input parameter structuretype must be one of the following strings: "LINEAR", "NONLINEAR", "BOTH" depending on whether the client needs information about the linear or the nonlinear part of the objective function, or both.

- If option “BOTH” is specified, and a variable occurs both linearly and nonlinearly in the objective function, then the value returned is the sum of:

  - the corresponding linear coefficient and

- the corresponding partial derivative of the nonlinear part of the objective function.

- The indices of the variables to which the elements of this vector correspond can be obtained from method GetMINLPStructure as the integer vector objindex.

Arguments

Name	Type	Description
[in] stype	CapeString	Specifies whether returned structure should be of type (1), (2) or (3) (see "Description" above)
[out] v	CapeArrayDouble	A sequence of doubles containing the nonzero elements of the linear part of the objective function, or of the partial derivatives of the nonlinear part of the objective function, or both

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPObjectiveFunctionBooleanAttribute
Returns	--

Description

Returns a boolean attribute of the objective function.

Arguments

Name	Type	Description
[in] attrib	CapeString	Name of the boolean attribute
[out] value	CapeBoolean	Value of the attribute

Errors

ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPObjectiveFunctionIntegerAttribute
Returns	--

Description

Returns an integer attribute of the objective function.

Arguments

Name	Type	Description
[in] attrib	CapeString	Name of the long attribute
[out] values	CapeLong	Value of the attribute

Errors

ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPObjectiveFunctionDoubleAttribute
Returns	--

Description

Returns a double attribute of the objective function.

Arguments

Name	Type	Description
[in] attrib	CapeString	Name of the double attribute
[out] value	CapeDouble	Value of the attribute

Errors

ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPObjectiveFunctionStringAttribute
Returns	--

Description

Returns a string attribute of the objective function.

Arguments

Name	Type	Description
[in] attrib	CapeString	Name of the string attribute
[out] value	CapeString	Value of the attribute

Errors

ECapeUnknown

Interface Name	ICapeMINLP
Method Name	SetMINLPLagrangeMultipliers
Returns	--

Description

Set the Lagrange multipliers of variables/constraints in MINLP. The envisaged usage of this method is for the numerical solvers to be able to record the Lagrange multipliers back into the MINLP.

Notes:

The indices ids must be in the range 1,...,nv (if setting variables Lagrange multipliers) or 1, ..., nc (if setting constraints Lagrange multipliers).

Arguments

Name	Type	Description
[in] lmtype	CapeString	"VARIABLE" if setting Lag. mult. relative to variables, "CONSTRAINT" if setting Lag. mult. relative to constraints
[in] ids	CapeArrayLong	The indices of a subset of variables/constraints in the MINLP
[in] values	CapeArrayDouble	The i-th element of this sequence contains the Lagrange multiplier for variable/constraint ids[i]

Errors

ECapeInvalidArgument, ECapeUnknown

Interface Name	ICapeMINLP
Method Name	GetMINLPLagrangeMultipliers
Returns	--

Description

Get the Lagrange multipliers of variables/constraints in MINLP. They must be set using method SetMINLPLagrangeMultipliers (see above) by the numerical solvers.

Notes:

The indices ids must be in the range 1,...,nv (if getting variables Lagrange multipliers) or 1, ..., nc (if getting constraints Lagrange multipliers).

Arguments

Name	Type	Description
[in] lmtype	CapeString	"VARIABLE" if setting Lag. mult. relative to variables, "CONSTRAINT" if setting Lag. mult. relative to constraints
[in] ids	CapeArrayLong	The indices of a subset of variables/constraints in the MINLP
[out] values	CapeArrayDouble	The i-th element of this sequence contains the Lagrange multiplier for variable/constraint ids[i]

Errors

ECapeInvalidArgument, ECapeUnknown, ECapeFailedInitialisation (returned if SetMINLPLagrangeMultipliers was never called prior to call to GetMINLPLagrangeMultipliers)

Interface Name	ICapeMINLP
Method Name	GetMINLPHessianStructure
Returns	--

Description

Gets the size and sparsity structure of the upper triangular section of the Hessian of the objective function.

Arguments

Name	Type	Description
[in] size	CapeLong	Number of nonzero elements of the upper triangular section of the Hessian
[in] rowindex	CapeArrayLong	The row indices corresponding to the nonzero elements of the upper triangular section of the Hessian
[out] columnindex	CapeArrayLong	The column indices corresponding to the nonzero elements of the upper triangular section of the Hessian

Errors

ECapeUnknown, ECapeHessianInfoNotAvailable

Interface Name	ICapeMINLP
Method Name	SetMINLPHessianValues
Returns	--

Description

Sets the values of the nonzero elements of the upper triangular section of the Hessian of the objective function.

Notes:

This method is a prerequisite for calling the method GetMINLPHessianValues.

Arguments

Name	Type	Description
[in] values	CapeArrayDouble	The values of the nonzero elements of the upper triangular section of the Hessian

Errors

ECapeUnknown, ECapeHessianInfoNotAvailable

Interface Name	ICapeMINLP
Method Name	GetMINLPHessianValues
Returns	--

Description

Gets the values of the nonzero elements of the upper triangular section of the Hessian of the objective function.

Notes:

This method should not be called before SetMINLPHessianValues.

Arguments

Name	Type	Description
[out] values	CapeArrayDouble	The values of the nonzero elements of the upper triangular section of the Hessian

Errors

ECapeUnknown, ECapeHessianInfoNotAvailable

3.6.2 ICapeMINLPSystem

A *MINLP system* represents the conjunction of a selected solver with a particular MINLP problem. It is created via an MINLP Solver Manager (see section 3.6.3).

The ICapeMINLPSystem has GetParameters(). This allows the configuration of the algorithmic parameters affecting the solution of individual MINLPs possible after the MINLPSystem object is created.

The ICapeMINLPSystem interface has only two methods of its own, as described below.

Interface Name	ICapeMINLPSystem
Method Name	Solve
Returns	--

Description

Solve the MINLP with which the system was created.

Notes:

This method must ensure that the final values of the solved variables are set in the MINLP object using the method SetMINLPVariableValues.

Arguments

None

Errors

ECapeOutOfResources, ECapeNoMemory, ECapeTimeOut, ECapeSolvingError, ECapeLicenceError, ECapeUnknown

Interface Name	ICapeMINLPSystem
Method Name	GetParameters
Returns	CapeInterface (ICapeCollection)

Description

Return a collection of parameters.

Arguments

None

Errors

ECapeUnknown, ECapeFailedInitialisation, ECapeBadInvOrder

3.6.3 ICapeMINLPSolverManager

An *MINLP Solver Manager* corresponds to a particular MINLP solver code. Given an MINLP object, exposing an ICapeMINLP interface (cf. section 3.6.1), the solver manager creates an MINLPSystem (cf. section 3.6.2) which can then be solved.

It is possible to configure the behaviour of a solver manager object thanks to *Common Interfaces*, for example by specifying the values of algorithmic parameters; all MINLPSystems subsequently created by this manager will automatically have these algorithmic parameter values.

The ICapeMINLPSolverManager interface has only one method of its own, as described below.

Interface Name	ICapeMINLPSolverManager
Method Name	CreateMINLPSystem
Returns	--

Description

Create an MINLP system corresponding to a given MINLP object.

Notes:

- It is the solver manager's responsibility to examine the MINLP object passed to it to ensure that it is within the scope and capabilities of the corresponding solver. If not, then this method should abort with an ECapeOutsideSolverScope exception. For example:
    - a solver manager corresponding to an LP solver may refuse to solve any MINLP problem that actually involves any nonlinearities
    - a solver manager corresponding to an NLP solver may refuse to solve any MINLP problem that actually involves any integer variables.
- The exact nature of an MINLP (i.e. whether or not it involves nonlinearities and/or integer variables) can easily be ascertained by a call to the MINLP object's GetMINLPSize method.

Arguments

Name	Type	Description
[in] theMINLP	CapeInterface (ICapeMINLP)	The MINLP
[out] theMINLPSystem	CapeInterface (ICapeMINLPSystem)	The created MINLP system

Errors

ECapeInvalidArgument, ECapeOutOfResources, ECapeNoMemory, ECapeLicenceError, ECapeUnknown, ECapeOutsideSolverScope

3.7 Scenarios

4. Interface Specifications

4.1 COM IDL

```
// You can get these intructions in Minlp.idl file from CAPE-OPENv1-0-0.zip
```

4.2 CORBA IDL

```
// You can get these intructions in CAPE-OPENv1-0-0.idl within the
CAPEOPEN100::Business::Numeric::Minlp module
```

5. Notes on the interface specifications

6. Prototypes implementation

7. Specific Glossary Terms

8. Bibliography

9. Appendices