NLPAPI: An API to Nonlinear Programming Problems. **Reference**

Michael E. Henderson
IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598
mhender@watson.ibm.com

June 24, 2003

1 Contents

1.0.1 The Objective

1.0.2 Equality Constraints

1.0.3 Inequality Constraints

1.1 Error Handling

1.2 Vectors

1.3 Matrices

1.4 Group Functions

1.5 Element Functions

1.5.1 Nonlinear Elements

1.5.2 The list of groups

1.5.3 The nonlinear elements (of each group)

1.5.4 The list of Types

1.6 Lancelot

1.6.1 Construction

1.6.2 Solving Problems

1.6.3 Setting LANCELOT Parameters

NLCreateProblem

Purpose

Allocates and initializes an NLProblem data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
P=NLCreateProblem(probName,nV);
```

`NLProblem`	*P*	The problem.
`char`	**probName*	The problem name, used on reports.
`int`	*nV*	The number of variables in the problem.

Description

The routine `NLCreateProblem` allocates and initializes an NLProblem data structure. The problem as initialized has no nonlinear constraints, no bounds on the variables or objective, and no groups in the objective. Note that this is not a valid problem. Lancelot will be invoked, but will issue an error unles a group is added to the objective.

A problem should be deleted using the routine `NLFreeProblem` (page 15) when it is no longer needed. This returns all storage used by the problem.

Errors

Severity 12 errors return (NLProblem)NULL, severity 4 returns a problem with no name.

Message	Severity
"Problem name (argument 1) is NULL"	4
"Number of Variables %d (argument 2) must be positive"	12
"Out of memory, trying to allocate %d bytes"	12

NLRefProblem

Purpose

Releases storage associated with an NLProblem data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLRefProblem(P);
```

`NLProblem`	*P*	The problem.

Description

The routine `NLRefProblem` adds a reference to a problem. `NLFreeProblem` (page 15) removes one reference and releases the storage associated with the problem if the reference count is zero. This allows the user to make a copy of the problem (the NLProblem is a pointer), and not have it disappear until sometime after the user calls `NLFreeProblem`.

Errors

Errors return without changing the problem.

Message	Severity
”Problem (argument 1) is NULL”	12

NLFreeProblem

Purpose

Releases storage associated with an NLProblem data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLFreeProblem(P);
```

`NLProblem` *P* The problem.

Description

The routine `NLFreeProblem` removes one reference and releases the storage associated with the problem if the reference count is zero. The routine `NLRefProblem` (page 14) adds a reference to a problem. This allows the user to make a copy of the problem (the NLProblem is a pointer), and not have it disappear until sometime after the user calls `NLFreeProblem`.

Errors

Errors return without changing the problem.

Message	Severity
"Problem (argument 1) is NULL"	12

NLPrintProblem

Purpose

Prints a NLProblem data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLPrintProblem(fid,P);
```

`FILE`	**fid*	The output file.
`NLProblem`	*P*	The problem.

Description

The routine `NLPrintProblem` prints an NLProblem data structure. The output attempts to mimic the SIF decoders printing.

Errors

Errors return without printing the problem.

Message	Severity
”File pointer (argument 1) is NULL”	12
”Problem (argument 2) is NULL”	12

NLPrintProblemShort

Purpose

Prints a NLProblem data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLPrintProblemShort(fid,P);
```

`FILE`	**fid*	The output file.
`NLProblem`	*P*	The problem.

Description

The routine `NLPrintProblem` prints an NLProblem data structure. The output attempts to give a compact version of the problem. **Errors**
Errors return without printing the problem.

Message	Severity
"File pointer (argument 1) is NULL"	12
"Problem (argument 2) is NULL"	12

NLPGetProblemName

Purpose

Returns the name of a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
name=NLPGetProblemName(P);
```

`char`	**name*	The problem name.
`NLProblem`	*P*	The problem.

Description

This routine returns the name of a problem, which was passed to the `NLCreateProblem` (page 13) subroutine.

Note: The user should not free the returned string.

Errors

Errors return (char*)NULL.

Message	Severity
"Problem (argument 1) is NULL"	12

NLPGetNumberOfVariables

Purpose

Returns the number of variables for a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfVariables(P);
```

`int`	n	The number of variables.
`NLProblem`	P	The problem.

Description

This routine returns the number of variables for a problem. This is set when the `NLCreateProblem` (page 13) subroutine is called.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12

NLPSetVariableScale

Purpose

Sets the scale factor of a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetVariableScale(P,i,s);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*i*	The variable.
`double`	*s*	The scale factor.

Description

This routine sets the scale factor of a variable. This can be queried with the `NLPGetVariableScale` (page 21) subroutine. The default value is 1.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Variable number %d (argument 2) is illegal. Must be in range 0 to %d”	12

NLPGetVariableScale

Purpose

Returns the scale factor of a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
s=NLPGetVariableScale(P,i);
```

`double`	s	The scale factor.
`NLProblem`	P	The problem.
`int`	i	The variable.

Description

This routine returns the scale factor of a variable. This is set with the `NLPSetVariableScale` (page 20) subroutine. The default value is 1.

Errors

Errors return DBL_QNAN.

Message	Severity
"Problem (argument 1) is NULL"	12
"Variable number %d (argument 2) is illegal. Must be in range 0 to %d"	12

NLPSetVariableName

Purpose

Assigns the name of a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetVariableName(P,i,name);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*i*	The number of the variable.
`char`	**name*	The problem name.

Description

This routine sets the name of a variable. This may be queried with the `NLPSetVariableName` subroutine (page 22). If the variable has not yet been given a name, the default is "Xxxxxxxx", where 'x' is a hex digit 0-9A-F. This is create with the C-format "X

A copy of the string is made. The copy is freed when the problem is freed.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Variable number %d (argument 2) is illegal. Must be in range 0 to %d"	12
"The pointer to the variable name (argument 3) is NULL."	4
"Out of memory, trying to allocate %d bytes"	12

NLPGetVariableName

Purpose

Returns the name of a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
name=NLPGetVariableName(P,i);
```

`char`	**name*	The problem name.
`NLProblem`	*P*	The problem.
`int`	*i*	The number of the variable.

Description

This routine returns the name of a variable. If the variable has not yet been given a name, the default is “Xxxxxxxx”, where ’x’ is a hex digit 0-9A-F. This is create with the C-format “X

Note: The user should not free the returned string.

Errors

Errors return (char*)NULL.

Message	Severity
”Problem (argument 1) is NULL”	12
”Variable number %d (argument 2) is illegal. Must be in range 0 to %d”	12

NLPSetSimpleBounds

Purpose

Sets the bounds on a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetSimpleBounds(P,var,l,u);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*var*	Which variable.
`double`	*l*	The lower bound.
`double`	*u*	The upper bound.

Description

This routine sets both of the bounds on the variable. These can be queried with the `NLPGetUpperSimpleBound` (page 29) and `NLPGetLowerSimpleBound` (page 26) subroutines. The bounds can also be set one at a time using the `NLPSetLowerSimpleBound` (page 25) and `NLPSetUpperSimpleBound` (page 28) routines.

Initially the bounds are infinite. A value greater than $1.e20$ is considered to be infinity (and a value less than $-1.e20$ is minus infinity).

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Variable number %d (argument 2) is illegal. Must be in range 0 to %d”	12

NLPSetLowerSimpleBound

Purpose

Sets the lower bound on a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetLowerSimpleBound(P,var,l);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*var*	Which variable.
`double`	*l*	The lower bound.

Description

This routine sets the lower bound on the variable. This can be queried with the `NLPGetLowerSimpleBound` (page 26) subroutine. The bounds can also be set at the same time using the `NLPSetSimpleBounds` (page 24) routine.

Initially the bound is $-\infty$. (A value of $-1.e20$ is considered to be infinity.)

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Variable number %d (argument 2) is illegal. Must be in range 0 to %d"	12

NLPGetLowerSimpleBound

Purpose

Gets the lower bound on a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
l=NLPGetLowerSimpleBound(P,var);
```

`double`	*l*	The lower bound.
`NLProblem`	*P*	The problem.
`int`	*var*	Which variable.

Description

This routine returns the lower bound on the variable.

Initially the bound is $-\infty$. (A value of $-1.e20$ is considered to be infinity.)

Errors

Errors return DBL_QNAN.

Message	Severity
"Problem (argument 1) is NULL"	12
"Variable number %d (argument 2) is illegal. Must be in range 0 to %d"	12

NLPIsLowerSimpleBoundSet

Purpose

Queries whether a lower bound has been set on a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
ans=NLPIsLowerSimpleBoundSet(P,var);
```

`int`	*ans*	The answer, 1==Set, 0=Not Set.
`NLProblem`	*P*	The problem.
`int`	*var*	The index of the variable.

Description

This routines queries whether a lower bound has been set on a variable. If it still has it's default value ($-\infty$), the routine returns 0, otherwise 1.

If an error occurs ans will be -1.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Variable number %d (argument 2) is illegal. Must be in range 0 to %d”	12

NLPSetUpperSimpleBound

Purpose

Sets the upper bound on a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetUpperSimpleBound(P,var,u);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*var*	Which variable.
`double`	*u*	The upper bound.

Description

This routine sets the upper bound on the variable. This can be queried with the `NLPGetUpperSimpleBound` (page 29) subroutine. The bounds can also be set at the same time using the `NLPSetSimpleBounds` (page 24) routine.

Initially the bound is ∞. (A value of $1.e20$ is considered to be infinity.)

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Variable number %d (argument 2) is illegal. Must be in range 0 to %d"	12

NLPGetUpperSimpleBound

Purpose

Gets the upper bound on a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
u=NLPGetUpperSimpleBound(P,var);
```

`double`	u	The upper bound.
`NLProblem`	P	The problem.
`int`	var	Which variable.

Description

This routine gets the upper bound on the variable.

Initially the bound is ∞. (A value of $1.e20$ is considered to be infinity.)

Errors

Errors return DBL_QNAN.

Message	Severity
”Problem (argument 1) is NULL”	12
”Variable number %d (argument 2) is illegal. Must be in range 0 to %d”	12

NLPIsUpperSimpleBoundSet

Purpose

Queries whether a upper bound has been set on a variable.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
ans=NLPIsUpperSimpleBoundSet(P,var);
```

`int`	*ans*	The answer, 1==Set, 0=Not Set.
`NLProblem`	*P*	The problem.
`int`	*var*	The index of the variable.

Description

This routines queries whether a upper bound has been set on a variable. If it still has it's default value (∞), the routine returns 0, otherwise 1.

If an error occurs ans will be -1.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Variable number %d (argument 2) is illegal. Must be in range 0 to %d"	12

NLConvertToEqualityAndBoundsOnly

Purpose

Eliminates the inequality constraints from a Problem by introducing slack variables.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLConvertToEqualityAndBoundsOnly(P);
```

`NLProblem`	P	The problem.

Description

The routine `NLConvertToEqualityAndBoundsOnly` takes a problem and introduces slack variables (one for each inequality constraint) to convert the inequality constraints into equality constraints. That is, an inequality constraint with both bounds

$$l \leq f(\mathbf{v}) \leq u$$

is replaced by an equality constraint and simple bounds on the slack –

$$\begin{array}{l} f(\mathbf{v}) - s = 0 \\ 0 \leq s \leq u - l \end{array}$$

An inequality constraint with only an upper bound

$$f(\mathbf{v}) \leq u$$

is replaced by an equality constraint and simple bounds on the slack –

$$\begin{array}{l} f(\mathbf{v}) + s = u \\ 0 \leq s \end{array}$$

And finally, an inequality constraint with only a lower bound

$$l \leq f(\mathbf{v}) \leq u$$

is replaced by an equality constraint and simple bounds on the slack –

$$\begin{aligned} &f(\mathbf{v}) - s = l \\ &0 \leq s \end{aligned}$$

The way this is currently done requires that the inequality constraints have a single group with a trivial group function (this is true if the high level routines like NLPAddInequalityConstraint are used).

Errors

Severity 12 errors return (NLProblem)NULL, severity 4 returns a problem with no name.

Message	Severity
”Problem (argument 1) is NULL”	4
”Inequality Constraint %d has %d groups, only one is allowed currently and it must be the trivial group function”	4
”Inequality Constraint %d group must currently have the trivial group function”	4
”Out of memory, trying to allocate %d bytes”	12

NLCopyProblem

Purpose

Creates a copy of an NLProblem data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
Q=NLCopyProblem(P);
```

`NLProblem`	P	The problem.
`NLProblem`	Q	The copy.

Description

The routine `NLCopyProblem` makes a “shallow” copy of a problem. That is, the lists of constraints are duplicated, but the functions defining the objective and constraints (the group and element functions) are not.

Errors

Severity 12 errors return (NLProblem)NULL, severity 4 returns a problem with no name.

Message	Severity
”Problem (argument 1) is NULL”	4
”Out of memory, trying to allocate %d bytes”	12

NLCreateAugmentedLagrangian

Purpose

Replaces the equality constraints in a Problem with a quadratic penalty and Lagrangian terms in the objective.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLCreateAugmentedLagrangian(P,mu,l, g,b,s);
```

`NLProblem`	*P*	The problem.
`double`	*mu*	The penalty parameter μ.
`double*`	*l*	The Lagrange multipliers λ_i. Array must be at least of length `nc` (the number of equality constraints),
`int*`	*g*	The indices of the new groups in the objective (values returned). The user is responsible for allocating this array. Array must be at least of length `nc` (the number of equality constraints),
`double*`	*b*	The constant elements of the equality constraints (values returned). The user is responsible for allocating this array. Array must be at least of length `nc` (the number of equality constraints),
`double*`	*s*	The group scales of the equality constraints (values returned). The user is responsible for allocating this array. Array must be at least of length `nc` (the number of equality constraints),

Description

The routine `NLCreateAugmentedLagrangian` takes a problem and replaces the equality constraints with a quadratic penalty function and lagrangian in the objective. That is, a problem

$$\text{minimize } O(\mathbf{v})$$

$$f_i(\mathbf{v}) = 0$$

is replaced by a problem with no equality constraints and objective –

$$\text{minimize } O(\mathbf{v}) + \frac{1}{2\mu}\sum\left(f_i(\mathbf{v}) - \mu\lambda_i\right)$$

The Lagrange multipliers λ_i (on for each equality constraint) are not problem variables, but are treated as parameters in the objective (as is the penalty parameter μ).

The Lagrange multipliers and penalty parameter are given the values passed. They may be changed with the `NLSetLambdaAndMuInAugmentedLagrangian` routine (page 36). The arrays g, b and s which are filled in by this routine must be passed to that routine.

Errors

Severity 12 errors return (NLProblem)NULL, severity 4 returns a problem with no name.

Message	Severity
"Problem (argument 1) is NULL"	4
"Equality Constraint %d has %d groups. This is not supported yet."	12
"Equality Constraint %d has a nontrivial group function. This is not supported yet."	12
"Out of memory, trying to allocate %d bytes"	12

NLSetLambdaAndMuInAugmentedLagrangian

Purpose

Sets the penalty parameter and LLagrange multipliers n a Problem with a quadratic penalty and Lagrangian terms in the objective.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLSetLambdaAndMuInAugmentedLagrangian(P,mu,l, g,b,s);
```

`NLProblem`	*P*	The problem.
`double`	*mu*	The penalty parameter μ.
`double*`	*l*	The Lagrange multipliers λ_i. Array must be at least of length `nc` (the number of equality constraints),
`int*`	*g*	The indices of the new groups in the objective. This array must be the one returned by the CreateAugmentedLagrangian routine,
`double*`	*b*	The constant elements of the equality constraints. This array must be the one returned by the CreateAugmentedLagrangian routine,
`double*`	*s*	The group scales of the equality constraints. This array must be the one returned by the CreateAugmentedLagrangian routine,

Description

The routine `NLSetLambdaAndMuInAugmentedLagrangian` changes the objective in a problem with augmented lagrangian, setting new values of the penalty parameter μ and Lagrange multipliers λ_i.

The arrays g, b and s must be those rteuned by the `NLCreateAugmented-Lagrangian` routine (page 34).

Errors

Severity 12 errors return (NLProblem)NULL, severity 4 does not cahnge the problem.

Message	Severity
”Problem (argument 1) is NULL”	4

NLEliminateFixedVariables

Purpose

For each variable whose upper and lower simple bounds are identical, introduces a linear equality constraint and removes the bounds.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLEliminateFixedVariables(P);
```

`NLProblem`	P	The problem.

Description

The routine `NLEliminateFixedVariables` takes a problem and for each variable whose upper and lower simple bounds are identical, introduces a linear equality constraint and removes the bounds. That is, a problem with a variable

$$l_i \leq x_i \leq u_i$$

with $l_i = u_i$ acquires an additional equality constraint

$$x_i = u_i$$

and the bounds on x_i are removed. **Errors**
Severity 12 errors return (NLProblem)NULL, severity 4 returns a problem with no name.

Message	Severity
”Problem (argument 1) is NULL”	4

NLPSetObjective

Purpose

Sets the objective to be a function defined by a subroutine (and it's derivatives).

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLPSetObjective(P,name,nv,v,f,df,ddf,data,freeData);
```

`NLProblem` *P*	The problem.
`char*` *name*	A name given to the objective (”Obj” might be good choice.)
`int` *nv*	The dimension of the domain of the objectiv This provides some degree of sparsity.
`int*`*v*	A list of length *nv* of the problem variables th the objective depends on. When the objective evaluated the routines are passed the values these problem variables in this order.
`double` *f*`(int,double*,void*)`	The routine giving the value of the function.
`double` *df*`(int,int,double*,void*)`	The routine giving the value of the derivative the function.
`double` *ddf*`(int,int,int,double*,void*)`	The routine giving the value of the second deriv tive of the function.
`void *`*data*	A Data Block that is to be passed to the functio
`void` *freeData*`(void*)`	A routine to free the data block.

Description

The routine `NLPSetObjective` sets the objective. The routines *f*, *df*, and *ddf* define the objective function. These are scalar valued functions of a subset of the problem variables. The subset is defined by way of the *nv*, and *v* arguments. When the objective is evaluated the routine *f* will be called.

double *f*`(int` *nv*`,double *`*x*`,void *`*data*`);`

The first argument to *f* will be *nv*. The second argument is an array *x* containing values of problem variables *v*[0],...*v*[*nv*-1]. The third argument is the void pointer to *data*. This allows the user to write one subroutine which, perhaps, uses the number of variables to decide which value to return (e.g. the sum of the squares of all the variables), or to pass parameters through the *data* block. The function returns the value of the objective.

The routines *df*, and *ddf* are similar, but with additional arguments for the derivatives being requested

```
double df(int nv,int i, double *x, void *data);
```

returns $\partial f/\partial x_i$. And

```
double ddf(int nv,int i, int j, double *x, void *data);
```

returns $\partial^2 f/\partial x_i \partial x_j$.

When the problem is free'd with `NLFreeProblem`, and the reference count becomes zero, the *freedata* routine will be called to allow the user to free the data block.

Errors

Message	Severity
"Problem (first arg.) is NULL, you must supply a problem."	12
"name (second arg.) is NULL, you must supply a name for the constraint."	12
"This problem already has an objective."	12
"Out of memory, trying to allocate %d bytes"	12

NLPSetObjectiveByString

Purpose

Sets the objective to be a function defined by a string.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLPSetObjectiveByString(P,name,nv,v,varlist,expr);
```

`NLProblem` *P* The problem.

`char*` *name* A name given to the objective (”Obj” might be a good choice.)

`int` *nv* The dimension of the domain of the objective. This provides some degree of sparsity.

`int*`*v* A list of length *nv* of the problem variables that the objective depends on.

`char*` *varlist* A list of identifiers in the *expr*. When the expression is elvaluated these identifiers will be given the values of the problem variables listed in *v* (in the same order). The list is a single string, delimited by the characters “[” and “]”, and separated by commas.

`char*` *expr* An expression giving the objective.

Description

The routine `NLPSetObjectiveByString` sets the objective. The string *expr* defines the objective function. When evaluated, the identifiers listed in *varlist* are given the values of the problem variables listed in the array *v*. There should be *nv* entries in *v* and in *varlist*. For example

```
int v[3];
v[0]=1;v[1]=45;v[2]=0;

NLPSetObjective(P,"Obj",3,v, "[a,b,c]", "a**2+sqrt(cos(b))+1./c");
```

will assign `a` the value of problem variable 1, `c` the value of problem variable 45, and `c` the value of problem variable 0. The main restriction on the expression is that constants may *not* be specified using exponential notation (sorry). Elementary functions and the usual binary operations can be used. Automatic differentiation is used to evaluate the derivatives.

Errors

Message	Severity
"Problem (first arg.) is NULL, you must supply a problem."	12
"name (second arg.) is NULL, you must supply a name for the constraint."	12
"This problem already has an objective."	12
"Out of memory, trying to allocate %d bytes"	12

NLPAddGroupToObjective

Purpose

Adds a group to the objective function.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
g=NLPAddGroupToObjective(P,name,type);
```

`int`	*g*	The index of the new group.
`NLProblem`	*P*	The problem.
`char`	**name*	The name of the new group.
`char`	**type*	The type of the new group.

Description

This routine adds a group to the objective function. The *name* of the group must be unique. The *type* need not be.

A trivial group is added, with no nonlinear element, and a zero linear element. This can be added with the `NLPSetGroupFunction` (page 185), `NLPSetGroupScale` (page 194), `NLPSetGroupA` (page 188), and `NLPSetGroupB` (page 191) routines.

Errors

Message	Severity
”Problem (argument 1) is NULL”	12
”Out of memory, trying to allocate %d bytes”	12

NLPAddNonlinearElementToObjectiveGroup

Purpose

Adds an empty nonlinear element to a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
e=NLPAddNonlinearElementToObjectiveGroup(P,group,type,weight,f,variables,xfrm);
```

`int`	*e*	The index of the new nonlinear element.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`char`	**type*	The type of the new nonlinear element.
`double`	*weight*	The weight.
`NLElementFunction`	*f*	The element function or NULL.
`int`	it variables	A list of the internal variables.
`NLMatrix`	*xfrm*	The range transformation or NULL.

Description

This routine adds a nonlinear element to a group in the objective. **Errors**

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12
”Out of memory, trying to allocate %d bytes”	12

NLPSetObjectiveGroupA

Purpose

Sets the linear part of the linear element of a group in the objective.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
rc=`NLPSetObjectiveGroupA(`*P*`,`*group*`,`*a*`);`

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`NLVector`	*a*	The linear element.

Description

This routine sets the linear part of the lineear element of a group in the objective.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPSetObjectiveGroupB

Purpose

Sets the constant part of the linear element of a group in the objective.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetObjectiveGroupB(P,group,b);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`double`	*b*	The constant.

Description

This routine sets the constant part of the linear element of a group in the objective. The default value is zero.

Note: The definition uses a negative sign for the constant that might be counter intuitive.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPSetObjectiveGroupFunction

Purpose

Sets the group function of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetObjectiveGroupFunction(P,group,g);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`NLGroupFunction`	*g*	The group function.

Description

This routine sets the group function of a group in the objective. **Errors** Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12
”Can’t Set Trivial Group’s Group Function, group %d”,group	12

NLPSetObjectiveGroupScale

Purpose

Sets the group function of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetObjectiveGroupScale(P,group,s);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`double`	*s*	The group scale factor.

Description

This routine sets the group scale of a group in the objective. **Errors**
Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPGetNumberOfGroupsInObjective

Purpose

Returns the number of groups in the objective of a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfGroupsInObjective(P);
```

`int`	n	The number of groups.
`NLProblem`	P	The problem.

Description

This routine returns the current number of groups in the objective of a problem. Each time a group is added to the objective this number increases. It never decreases.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12

NLPGetObjectiveGroupNumber

Purpose

Returns the index of a group in the objective of a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
g=NLPGetObjectiveGroupNumber(P,i);
```

`int`	g	The index of the group.
`NLProblem`	P	The problem.
`int`	i	Which group.

Description

This routine returns the index of a group in the objective of a problem. Group queries use the group index. This routine can be used to query the properties of all of the groups in the Objective.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPEvaluateObjective

Purpose

Evaluates the objective function.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
o=NLPEvaluateObjective(P,x);
```

`double` *o*	The value of the objective function.
`NLProblem` *P*	The problem.
`NLVector` *x*	The point (problem variables) at which to evaluate the objective.

Description

The routine `NLPEvaluateObjective` evaluates the objective at a point x and returns the value. The user must create the vector x and give it the appropriate values.

Errors

Message	Severity
”Problem (first arg.) is NULL.”	12
”x (second arg.) is NULL.”	12

1newpage **NLPEvaluateGradientOfObjective**

Purpose

Evaluates the gradient of the objective function.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPEvaluateGradientOfObjective(P,x,g);
```

`int` *rc*	The return code, 1 indicates success.
`NLProblem` *P*	The problem.
`NLVector` *x*	The point (problem variables) at which to evaluate the objective.
`NLVector` *g*	The gradient of the objective. (The user passes the NLVector, which is given a value by the routine.

Description

The routine `NLPEvaluateGradientOfObjective` evaluates the gradient of the objective at a point *x* and returns the value. The user must create the vectors *x* and *g*, and give *x* the appropriate values. The routine sets the values in the gradient *g*. Note that if a sparse vector is passed in the gradient is returned in a sparse vector. If a dense vector is passed the result is returned in a dense vector.

Errors

Message	Severity
”Problem (first arg.) is NULL.”	12
”x (second arg.) is NULL.”	12
”g (third arg.) is NULL.”	12

NLPEvaluateHessianOfObjective

Purpose

Evaluates the Hessian of the objective function.

Library

```
libNLPAPI.a
```

C Syntax

```
#include <NLPAPI.h>
rc=NLPEvaluateHessianOfObjective(P,x,H);
```

`int` *rc*	The return code, 1 indicates success.
`NLProblem` *P*	The problem.
`NLVector` *x*	The point (problem variables) at which to evaluate the objective.
`NLMatrix` *H*	The Hessian of the objective. (The user passes the NLMatrix, which is given a value by the routine.

Description

The routine `NLPEvaluateHessianOfObjective` evaluates the Hessian of the objective at a point *x* and returns the value. The user must create the vectors *x* and *H*, and give *x* the appropriate values. The routine sets the values in the Hessian *H*. Note that the user determines the format of the Hessian when the NLMatrix is created.

Errors

Message	Severity
”Problem (first arg.) is NULL.”	12
”x (second arg.) is NULL.”	12
”H (third arg.) is NULL.”	12

NLPAddEqualityConstraint

Purpose

Adds an equality constraint defined by a subroutine (and it's derivatives) to a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
c=NLPAddEqualityConstraint(P,name,nv,v,f,df,ddf,data,freeData);
```

`int` *c*	The number assigned to the new constraint.
`NLProblem` *P*	The problem.
`char*` *name*	A name given to the constraint.
`int` *nv*	The dimension of the domain of the constrain This provides some degree of sparsity.
`int*`*v*	A list of length *nv* of the problem variables th the constraint depends on. When the constrai is evaluated the routines are passed the values these problem variables in this order.
`double` *f*`(int,double*,void*)`	The routine giving the value of the function.
`double` *df*`(int,int,double*,void*)`	The routine giving the value of the derivative the function.
`double` *ddf*`(int,int,int,double*,void*)`	The routine giving the value of the second deriv tive of the function.
`void *`*data*	A Data Block that is to be passed to the functio
`void` *freeData*`(void*)`	A routine to free the data block.

Description

The routine `NLPAddEqualityConstraint` adds an equality constraint. The routines *f*, *df*, and *ddf* define the constraint function. These are scalar valued functions of a subset of the problem variables. The subset is defined by way of the *nv*, and *v* arguments. When the constraint is evaluated the routine *f* will be called.

```
double f(int nv,double *x,void *data);
```

The first argument to *f* will be *nv*. The second argument is an array x containing values of problem variables *v*[0],...*v*[*nv*-1]. The third argument is the void pointer to *data*. This allows the user to write one subroutine which, perhaps, uses the number of variables to decide which value to return (e.g. the sum of the squares of all the variables), or to pass parameters through the *data* block. The function returns the value of the constraint.

The routines *df*, and *ddf* are similar, but with additional arguments for the derivatives being requested

double *df*(`int` *nv*,`int` *i*, `double *`*x*, `void *`*data*);

returns $\partial f/\partial x_i$. And

double *ddf*(`int` *nv*,`int` *i*, `int` *j*, `double *`*x*, `void *`*data*);

returns $\partial^2 f/\partial x_i \partial x_j$.

When the problem is free'd with `NLFreeProblem`, and the reference count becomes zero, the *freedata* routine will be called to allow the user to free the data block.

Errors

Message	Severity
"Problem (first arg.) is NULL, you must supply a problem."	12
"name (second arg.) is NULL, you must supply a name for the constraint."	12
"Out of memory, trying to allocate %d bytes"	12

NLPAddEqualityConstraintByString

Purpose

Adds an equality constraint defined by an expression in a string to a problem.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
c=`NLPAddEqualityConstraintByString(`*P*`,`*name*`,`*nv*`,`*v*`,`*varlist*`,`*expr*`);`

`int` *c*	The number assigned to the new constraint.
`NLProblem` *P*	The problem.
`char*` *name*	A name given to the constraint.
`int` *nv*	The dimension of the domain of the constraint. This provides some degree of sparsity.
`int*`*v*	A list of length *nv* of the problem variables that the constraint depends on.
`char*` *varlist*	A list of identifiers in the *expr*. When the expression is elvaluated these identifiers will be given the values of the problem variables listed in *v* (in the same order). The list is a single string, delimited by the characters "[" and "]", and separated by commas.
`char*` *expr*	An expression giving the constraint.

Description

The routine `NLPAddEqualityConstraintByString` sets the constraint. The string *expr* defines the constraint function. When evaluated, the identifiers listed in *varlist* are given the values of the problem variables listed in the array *v*. There should be *nv* entries in *v* and in *varlist*. For example

```
int v[3];
int c;

v[0]=1;v[1]=45;v[2]=0;
```

```
c=NLPAddEqualityConstraintByString(P,"Obj",3,v, "[a,b,c]", "a**2+sqrt(cos(b))+1./
```

will assign `a` the value of problem variable 1, `c` the value of problem variable 45, and `c` the value of problem variable 0. The main restriction on the expression is that constants may *not* be specified using exponential notation (sorry). Elementary functions and the usual binary operations can be used. Automatic differentiation is used to evaluate the derivatives.

Errors

Message	Severity
"Problem (first arg.) is NULL, you must supply a problem."	12
"name (second arg.) is NULL, you must supply a name for the constraint."	12
"Out of memory, trying to allocate %d bytes"	12

NLPAddNonlinearEqualityConstraint

Purpose

Adds a nonlinear equality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
g=NLPAddNonlinearEqualityConstraint(P,name);
```

`int`	*g*	The index of the new group.
`NLProblem`	*P*	The problem.
`char`	**name*	The name of the new group.

Description

This routine adds a nonlinear equality constraint. The *name* of the group must be unique. The *type* need not be.

A trivial group is added, with no nonlinear element, and a zero linear element. This can be added with the `NLPSetGroupFunction` (page 185), `NLPSetGroupScale` (page 194), `NLPSetGroupA` (page 188), and `NLPSetGroupB` (page 191) routines.

Errors

Message	Severity
”Problem (argument 1) is NULL”	12
”Out of memory, trying to allocate %d bytes”	12

NLPAddLinearEqualityConstraint

Purpose

Adds a linear equality constraint.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
g=`NLPAddLinearEqualityConstraint(`*P*`,`*name*`,`*a*`,`*b*`);`

`int`	*g*	The index of the new group.
`NLProblem`	*P*	The problem.
`char`	**name*	The name of the new group.
`double`	**a*	The linear part of the linear element.
`double`	*b*	The constant part of the linear element.

Description

This routine adds a linear equality constraint. That is, a constraint with the trivial group and no nonlinear elements. This is a convenience routine, the same as invoking the `NLPAddNonlinearEqualityConstraint` (page 58), `NLPSetGroupA` (page 188), and `NLPSetGroupB` (page 191) routines.

The *name* of the group must be unique. The *type* need not be.

A trivial group is added, with no nonlinear element, and the given linear element. The constraint can be modified using the `NLPSetGroupFunction` (page 185), `NLPSetGroupScale` (page 194), `NLPSetGroupA` (page 188), and `NLPSetGroupB` (page 191) routines.

Errors

Message	Severity
”Problem (argument 1) is NULL”	12
”Out of memory, trying to allocate %d bytes”	12

NLPAddNonlinearElementToEqualityConstraint

Purpose

Adds an empty nonlinear element to an equality constraint.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
e=`NLPAddNonlinearElementToEqualityConstraint(`*P*`,`*constraint*`,`*weight*`,`*ne*`,`*variables*`,`*xfrm*`);`
int NLPAddNonlinearElementToEqualityConstraint(NLProblem P,int constraint,double w,LNNonlinearElement E);

`int`	*e*	The index of the new nonlinear element.
`NLProblem`	*P*	The problem.
`int`	*constraint*	The index of the constraint.
`double`	*weight*	The weight.
`LNNonlinearElement`	*ne*	The nonlinear element.

Description

This routine adds a nonlinear element to an equality constraint. **Errors**

Message	Severity
"Problem (argument 1) is NULL"	12
"Constraint %d is illegal (argument 2). Must be in range 0 to %d"	12
"Out of memory, trying to allocate %d bytes"	12

NLPSetEqualityConstraintA

Purpose

Sets the linear part of the linear element of an equality constraint.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
rc=`NLPSetEqualityConstraintA(`*P*, *constraint*, *a*`);`

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*constraint*	The index of the constraint.
`NLVector`	*a*	The linear element.

Description

This routine sets the linear part of the lineear element of an equality constraint.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPSetEqualityConstraintB

Purpose

Sets the constant part of the linear element of an equality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetEqualityConstraintB(P,constraint,b);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*constraint*	The index of the constraint.
`double`	*b*	The constant.

Description

This routine sets the constant part of the linear element of an equality constraint. The default value is zero.

Note: The definition uses a negative sign for the constant that might be counter intuitive.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPGetNumberOfEqualityConstraints

Purpose

Returns the number of equality constraints in a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfEqualityConstraints(P);
```

`int`	n	The number of constraints.
`NLProblem`	P	The problem.

Description

This routine returns the current number of equality constraints in a problem.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12

NLPGetEqualityConstraintGroupNumber

Purpose

Returns the index of the group representing an equality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
group=NLPGetEqualityConstraintGroupNumber(P,c);
```

`int`	*group*	The index of the group.
`NLProblem`	*P*	The problem.
`int`	*c*	Which constraint.

Description

This routine returns the index of the group representing an equality constraint. This is the same index that is returned by the `NLPAddNonlinearEqualityConstraint` (page 58) subroutine.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Inequality constraint number %d (argument 2) is illegal. Must be in range 0 to %d”	12

NLPEvaluateEqualityConstraint

Purpose

Evaluates an equality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
o=NLPEvaluateEqualityConstraint(P,c,x);
```

`double` *o*	The value of the constraint.
`NLProblem` *P*	The problem.
`int` *c*	The number of the constraint to be evaluated.
`NLVector` *x*	The point (problem variables) at which to evaluate the constraint.

Description

The routine `NLPEvaluateEqualityConstraint` evaluates the constraint at a point x and returns the value. The user must create the vector x and give it the appropriate values.

Errors

Message	Severity
”Problem (first arg.) is NULL.”	12
”x (second arg.) is NULL.”	12

NLPEvaluateGradientOfEqualityConstraint

Purpose

Evaluates the gradient of an equality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPEvaluateGradientOfEqualityConstraint(P,c,x,g);
```

`int` *rc*	The return code, 1 if successful.
`NLProblem` *P*	The problem.
`int` *c*	The number of the constraint.
`NLVector` *x*	The point (problem variables) at which to evaluate the constraint.
`NLVector` *g*	The gradient of the objective. (The user passes the NLVector, which is given a value by the routine.

Description

The routine `NLPEvaluateGradientOfEqualityConstraint` evaluates the gradient of an equality constraint at a point *x* and returns the value. The user must create the vectors *x* and *g*, and give *x* the appropriate values. The routine sets the values in the gradient *g*. Note that if a sparse vector is passed in the gradient is returned in a sparse vector. If a dense vector is passed the result is returned in a dense vector.

Errors

Message	Severity
”Problem (first arg.) is NULL.”	12
”Constraint %d (argument 2) is Invalid, must be in [0,%d).”	12
”x (second arg.) is NULL.”	12
”g (third arg.) is NULL.”	12

NLPEvaluateHessianOfEqualityConstraint

Purpose

Evaluates the Hessian of an equality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPEvaluateHessianOfEqualityConstraint(P,c,x,H);
```

`int` *rc*	The return code, 1 indicates success.
`NLProblem` *P*	The problem.
`int` *c*	The number of the constraint.
`NLVector` *x*	The point (problem variables) at which to evaluate the objective.
`NLMatrix` *H*	The Hessian of the objective. (The user passes the NLMatrix, which is given a value by the routine.

Description

The routine `NLPEvaluateHessianOfEqualityConstraint` evaluates the Hessian of an equality constraint at a point x and returns the value. The user must create the vectors x and H, and give x the appropriate values. The routine sets the values in the Hessian H. Note that the user determines the format of the Hessian when the NLMatrix is created.

Errors

Message	Severity
"Problem (first arg.) is NULL."	12
"Constraint %d (argument 2) is Invalid, must be in [0,%d)."	12
"x (second arg.) is NULL."	12
"H (third arg.) is NULL."	12

NLPAddInequalityConstraint

Purpose

Adds an inequality constraint defined by a subroutine (and it's derivatives) to a problem.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
c=`NLPAddInequalityConstraint(`*P*`,`*name*`,`*l*`,`*u*`,`*nv*`,`*v*`,`*f*`,`*df*`,`*ddf*`,`*data*`,`*freeData*`);`

`int` *c*	The number assigned to the new constraint.
`NLProblem` *P*	The problem.
`char*` *name*	A name given to the constraint.
`double` *l*	The lower bound of the constraint. (Anything le than $-1.e20$ is taken to be $-\infty$.)
`double` *u*	The upper bound of the constraint. (Anythi greater than $1.e20$ is taken to be ∞.)
`int` *nv*	The dimension of the domain of the constrain This provides some degree of sparsity.
`int*`*v*	A list of length *nv* of the problem variables th the constraint depends on. When the constrai is evaluated the routines are passed the values these problem variables in this order.
`double` *f*`(int,double*,void*)`	The routine giving the value of the function.
`double` *df*`(int,int,double*,void*)`	The routine giving the value of the derivative the function.
`double` *ddf*`(int,int,int,double*,void*)`	The routine giving the value of the second deriv tive of the function.
`void *`*data*	A Data Block that is to be passed to the functior
`void` *freeData*`(void*)`	A routine to free the data block.

Description

The routine `NLPAddInequalityConstraint` adds an inequality constraint. The routines *f*, *df*, and *ddf* define the constraint function. These are scalar valued functions of a subset of the problem variables. The subset is defined by way of the *nv*, and *v* arguments. When the constraint is evaluated the routine *f* will be called.

```
double f(int nv,double *x,void *data);
```

The first argument to *f* will be *nv*. The second argument is an array *x* containing values of problem variables *v*[0],...*v*[*nv*-1]. The third argument is the void pointer to *data*. This allows the user to write one subroutine which, perhaps, uses the number of variables to decide which value to return (e.g. the sum of the squares of all the variables), or to pass parameters through the *data* block. The function returns the value of the constraint.

The routines *df*, and *ddf* are similar, but with additional arguments for the derivatives being requested

```
double df(int nv,int i, double *x, void *data);
```

returns $\partial f/\partial x_i$. And

```
double ddf(int nv,int i, int j, double *x, void *data);
```

returns $\partial^2 f/\partial x_i \partial x_j$.

When the problem is free'd with `NLFreeProblem`, and the reference count becomes zero, the *freedata* routine will be called to allow the user to free the data block.

Errors

Message	Severity
"Problem (first arg.) is NULL, you must supply a problem."	12
"name (second arg.) is NULL, you must supply a name for the constraint."	12
"Out of memory, trying to allocate %d bytes"	12

NLPAddInequalityConstraintByString

Purpose

Adds an inequality constraint defined by an expression in a string to a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
c=NLPAddInequalityConstraintByString(P,name,l,u,nv,v,varlist,expr);
```

`int` *c*	The number assigned to the new constraint.
`NLProblem` *P*	The problem.
`char*` *name*	A name given to the constraint.
`double` *l*	The lower bound of the constraint. (Anything less than $-1.e20$ is taken to be $-\infty$.)
`double` *u*	The upper bound of the constraint. (Anything greater than $1.e20$ is taken to be ∞.)
/ `int` *nv*	The dimension of the domain of the constraint. This provides some degree of sparsity.
`int*`*v*	A list of length *nv* of the problem variables that the constraint depends on.
`char*` *varlist*	A list of identifiers in the *expr*. When the expression is elvaluated these identifiers will be given the values of the problem variables listed in *v* (in the same order). The list is a single string, delimited by the characters "[" and "]", and separated by commas.
`char*` *expr*	An expression giving the constraint.

Description

The routine `NLPAddInequalityConstraintByString` sets the constraint. The string *expr* defines the constraint function. When evaluated, the identifiers listed in *varlist* are given the values of the problem variables listed in the array *v*. There should be *nv* entries in *v* and in *varlist*. For example

```
int v[3];
int c;

v[0]=1;v[1]=45;v[2]=0;

c=NLPAddInequalityConstraintByString(P,"Obj",-2.e20,1., 3,v, "[a,b,c]", "a**2+sqr
```

will assign a the value of problem variable 1, c the value of problem variable 45, and c the value of problem variable 0. The main restriction on the expression is that constants may *not* be specified using exponential notation (sorry). Elementary functions and the usual binary operations can be used. Automatic differentiation is used to evaluate the derivatives.

Errors

Message	Severity
"Problem (first arg.) is NULL, you must supply a problem."	12
"name (second arg.) is NULL, you must supply a name for the constraint."	12
"Out of memory, trying to allocate %d bytes"	12

NLPAddNonlinearInequalityConstraint

Purpose

Adds a nonlinear inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
g=NLPAddNonlinearInequalityConstraint(P,name);
```

`int`	*g*	The index of the new group.
`NLProblem`	*P*	The problem.
`char`	**name*	The name of the new group.

Description

This routine adds a nonlinear inequality constraint. The *name* of the group must be unique.

A trivial group is added, with no nonlinear element, and a zero linear element. This information can be added with the `NLPSetGroupFunction` (page 185), `NLPSetGroupScale` (page 194), `NLPSetGroupA` (page 188), and `NLPSetGroupB` (page 191) routines. Bounds can (and should be) set with the `NLPSetGroupFunction` (page 185), `NLPSetInequalityConstraintLowerBound` (page 78), `NLPSetInequalityConstraintUpperBound` (page 80), and `NLPSetInequalityConstraintBounds` (page 81) subroutines.

Errors

Message	Severity
”Problem (argument 1) is NULL”	12
”Out of memory, trying to allocate %d bytes”	12

NLPAddLinearInequalityConstraint

Purpose

Adds a linear inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
g=NLPAddLinearInequalityConstraint(P,name,a,b);
```

`int`	*g*	The index of the new group.
`NLProblem`	*P*	The problem.
`char`	**name*	The name of the new group.
`double`	**a*	The linear part of the linear element.
`double`	*b*	The constant part of the linear element.

Description

This routine adds a linear inequality constraint. The *name* of the group must be unique.

A trivial group is added, with no nonlinear element, and the given linear element. The constraint may be changed with the `NLPSetGroupFunction` (page 185), `NLPSetGroupScale` (page 194), `NLPSetGroupA` (page 188), and `NLPSetGroupB` (page 191) routines. Bounds can (and should be) set with the `NLPSetGroupFunction` (page 185), `NLPSetInequalityConstraintLowerBound` (page 78), `NLPSetInequalityConstraintUpperBound` (page 80), and `NLPSetInequalityConstraintBounds` (page 81) subroutines.

Errors

Message	Severity
”Problem (argument 1) is NULL”	12
”Out of memory, trying to allocate %d bytes”	12

NLPAddNonlinearElementToInequalityConstraint

Purpose

Adds an empty nonlinear element to an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
e=NLPAddNonlinearElementToInequalityConstraint(P,constraint,weight,ne,variables,xfrm);
```

int NLPAddNonlinearElementToInequalityConstraint(NLProblem P,int constraint,double w,LNNonlinearElement E);

`int`	*e*	The index of the new nonlinear element.
`NLProblem`	*P*	The problem.
`int`	*constraint*	The index of the constraint.
`double`	*weight*	The weight.
`LNNonlinearElement`	*ne*	The nonlinear element.

Description

This routine adds a nonlinear element to an inequality constraint. **Errors**

Message	Severity
"Problem (argument 1) is NULL"	12
"Constraint %d is illegal (argument 2). Must be in range 0 to %d"	12
"Out of memory, trying to allocate %d bytes"	12

NLPSetInequalityConstraintA

Purpose

Sets the linear part of the linear element of an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetInequalityConstraintA(P,constraint,a);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*constraint*	The index of the constraint.
`NLVector`	*a*	The linear element.

Description

This routine sets the linear part of the lineear element of an inequality constraint.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPSetInequalityConstraintB

Purpose

Sets the constant part of the linear element of an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetInequalityConstraintB(P,constraint,b);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*constraint*	The index of the constraint.
`double`	*b*	The constant.

Description

This routine sets the constant part of the linear element of an inequality constraint. The default value is zero.

Note: The definition uses a negative sign for the constant that might be counter intuitive.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPGetInequalityConstraintLowerBound

Purpose

Gets the lower bound for an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
l=NLPGetInequalityConstraintLowerBound(P,c);
```

`double`	l	The lower bound.
`NLProblem`	P	The problem.
`int`	c	Which constraint.

Description

This routine returns the lower bound for the inequality constraint.

Initially the bound is $-\infty$. (A value of $-1.e20$ is considered by Lancelot to be infinity.)

Errors

Errors return DBL_QNAN.

Message	Severity
"Problem (argument 1) is NULL"	12
"Inequality constraint number %d (argument 2) is illegal. Must be in range 0 to %d"	12

NLPSetInequalityConstraintLowerBound

Purpose

Sets the lower bound on an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetInequalityConstraintLowerBound(P,c,l);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*c*	Which constraint.
`double`	*l*	The lower bound.

Description

This routine sets the lower bound on the inequality constraint. This can be queried with the `NLPGetInequalityConstraintLowerBound` (page 77) subroutine. The bounds can also be set at the same time using the `NLPSetInequalityConstraintBounds` (page 81) routine.

Initially the bound is $-\infty$. (A value of $-1.e20$ is considered by Lancelot to be infinity.)

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Inequality constraint number %d (argument 2) is illegal. Must be in range 0 to %d"	12

NLPGetInequalityConstraintUpperBound

Purpose

Gets the upper bound for an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
u=NLPGetInequalityConstraintUpperBound(P,c);
```

`double`	u	The upper bound.
`NLProblem`	P	The problem.
`int`	c	Which constraint.

Description

This routine gets the upper bound for an inequality constraint.

Initially the bound is ∞. (A value of $1.e20$ is considered by Lancelot to be infinity.)

Errors

Errors return DBL_QNAN.

Message	Severity
"Problem (argument 1) is NULL"	12
"Inequality constraint number %d (argument 2) is illegal. Must be in range 0 to %d"	12

NLPSetInequalityConstraintUpperBound

Purpose

Sets the upper bound on an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetInequalityConstraintUpperBound(P,c,u);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*c*	Which constraint.
`double`	*u*	The upper bound.

Description

This routine sets the upper bound on the inequality constraint. This can be queried with the `NLPGetInequalityConstraintUpperBound` (page 79) subroutine. The bounds can also be set at the same time using the `NLPSetInequalityConstraintBounds` (page 81) routine.

Initially the bound is ∞. (A value of $1.e20$ is considered by Lancelot to be infinity.)

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Inequality constraint number %d (argument 2) is illegal. Must be in range 0 to %d”	12

NLPSetInequalityConstraintBounds

Purpose

Sets the bounds on an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
rc=`NLPSetInequalityConstraintBounds(`P, c, l, u`);`

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*c*	Which constraint.
`double`	*l*	The lower bound.
`double`	*u*	The upper bound.

Description

This routine sets the bounds on the inequality constraint. This can be queried with the `NLPGetInequalityConstraintUpperBound` (page 79) and `NLPGetInequalityConstraintLowerBound` (page 77) subroutine. The bounds can also be set one at a time using the `NLPSetInequalityConstraintUpperBound` (page 80) and `NLPSetInequalityConstraintLowerBound` (page 78) routines.

Initially the bounds are $-\infty$ to ∞. (A value of $-1.e20$ is considered by Lancelot to be infinity.) Obviously this is no constraint at all unless the bounds are set.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Inequality constraint number %d (argument 2) is illegal. Must be in range 0 to %d”	12

NLPGetNumberOfInequalityConstraints

Purpose

Returns the number of inequality constraints in a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfInequalityConstraints(P);
```

`int`	n	The number of constraints.
`NLProblem`	P	The problem.

Description

This routine returns the current number of inequality constraints in a problem.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12

NLPGetInequalityConstraintGroupNumber

Purpose

Returns the index of the group representing an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
group=NLPGetInequalityConstraintGroupNumber(P,c);
```

`int`	*group*	The index of the group.
`NLProblem`	*P*	The problem.
`int`	*c*	Which constraint.

Description

This routine returns the index of the group representing an inequality constraint. This is the same index that is returned by the `NLPAddNonlinearInequalityConstraint` (page 72) subroutine.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Inequality constraint number %d (argument 2) is illegal. Must be in range 0 to %d”	12

NLPEvaluateInequalityConstraint

Purpose

Evaluates an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
o=NLPEvaluateInequalityConstraint(P,c,x);
```

`double` *o*	The value of the constraint.
`NLProblem` *P*	The problem.
`int` *c*	The number of the constraint to be evaluated.
`NLVector` *x*	The point (problem variables) at which to evaluate the constraint.

Description

The routine `NLPEvaluateInequalityConstraint` evaluates the constraint at a point *x* and returns the value. The user must create the vector *x* and give it the appropriate values.

Errors

Message	Severity
”Problem (first arg.) is NULL.”	12
”x (second arg.) is NULL.”	12

NLPEvaluateGradientOfInequalityConstraint

Purpose

Evaluates the gradient of an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPEvaluateGradientOfInequalityConstraint(P,c,x,g);
```

`int` *rc*	The return code, 1 if successful.
`NLProblem` *P*	The problem.
`int` *c*	The number of the constraint.
`NLVector` *x*	The point (problem variables) at which to evaluate the constraint.
`NLVector` *g*	The gradient of the objective. (The user passes the NLVector, which is given a value by the routine.

Description

The routine `NLPEvaluateGradientOfInequalityConstraint` evaluates the gradient of an inequality constraint at a point x and returns the value. The user must create the vectors x and g, and give x the appropriate values. The routine sets the values in the gradient g. Note that if a sparse vector is passed in the gradient is returned in a sparse vector. If a dense vector is passed the result is returned in a dense vector.

Errors

Message	Severity
”Problem (first arg.) is NULL.”	12
”Constraint %d (argument 2) is Invalid, must be in [0,%d).”	12
”x (second arg.) is NULL.”	12
”g (third arg.) is NULL.”	12

NLPEvaluateHessianOfInequalityConstraint

Purpose

Evaluates the Hessian of an inequality constraint.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPEvaluateHessianOfInequalityConstraint(P,c,x,H);
```

`int` *rc*	The return code, 1 indicates success.
`NLProblem` *P*	The problem.
`int` *c*	The number of the constraint.
`NLVector` *x*	The point (problem variables) at which to evaluate the objective.
`NLMatrix` *H*	The Hessian of the objective. (The user passes the NLMatrix, which is given a value by the routine.

Description

The routine `NLPEvaluateHessianOfInequalityConstraint` evaluates the Hessian of an inequality constraint at a point x and returns the value. The user must create the vectors x and H, and give x the appropriate values. The routine sets the values in the Hessian H. Note that the user determines the format of the Hessian when the NLMatrix is created.

Errors

Message	Severity
"Problem (first arg.) is NULL."	12
"Constraint %d (argument 2) is Invalid, must be in [0,%d)."	12
"x (second arg.) is NULL."	12
"H (third arg.) is NULL."	12

NLError

Purpose

Queries whether an error condition exists.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLError();
```

`int` *rc* The return code.

Description

This routine checks to see if any routine has set an error condition.

NLGetNErrors

Purpose

Returns the number of errors that have been flagged.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLGetNErrors();
```

`int`	*n*	The number of errors.

Description

This routine returns the number of errors that have been set.

NLGetErrorSev

Purpose

Returns the severity of an error.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
sev=NLGetErrorSev(i);
```

`int`	*sev*	The severity.
`int`	*i*	Which error.

Description

This routine returns the severity of the ith error.

NLGetErrorRoutine

Purpose

Returns the name of the routine that issued an error.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
routine=NLGetErrorRoutine(i);
```

`char`	**routine*	The name of the routine.
`int`	*i*	Which error.

Description

This routine returns the name of the routine which issued the *i*th error.

NLGetErrorMsg

Purpose

Returns the message associated with an error.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
msg=NLGetErrorMsg(i);
```

`char`	**msg*	The message text.
`int`	*i*	Which error.

Description

This routine returns the message associated with the *i*th error.

NLGetErrorLine

Purpose

Returns the statement at which an error occurred.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
stmt=NLGetErrorLine(i);
```

`int`	*stmt*	The statement.
`int`	*i*	Which error.

Description

This routine returns the statement at which the *i*th error occured.

NLGetErrorFile

Purpose

Returns the file containing the source code from which an error was issued.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
file=NLGetErrorFile(i);
```

`char`	**file*	The file.
`int`	*i*	Which error.

Description

This routine returns the file containing the source code from which the *i*th error was issued.

NLClearErrors

Purpose

Clears all errors.

Library

```
libNLPAPI.a
```

C Syntax

```
#include <NLPAPI.h>
void NLClearErrors();
```

Description

This routine clears the error stack.

NLCreateVector

Purpose

Allocates and initializes an NLVector data structure of a given length.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
v=NLCreateVector(n);
```

`NLVector`	v	The vector.
`int`	n	The length of the vector.

Description

The routine `NLCreateVector` allocates an NLVector data structure and initializes it to a vector of given length with no non-zero coordinates. The coordinates may be changed with the `NLVSetC` routine (page 106). Vectors with supplied coordinate values can be created with the `NLCreateVectorWithSparseData` and `NLCreateVectorWithFullData` subroutines (pages 96 and 98).

The NLVector data structure uses reference counting. The data structure should be deleted using the `NLFreeVector` subroutine (page 101). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefVector` subroutine (page 102).

Errors

Severity 12 errors return (NLVector)NULL.

Message	Severity
”Length of Vector %d (argument 1) is Illegal. Must be positive.”	12
”Out of memory, trying to allocate %d bytes”	12

NLCreateVectorWithSparseData

Purpose

Allocates and initializes an NLVector data structure of a given length.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
v=NLCreateVectorWithSparseData(n,nz,el,vl);
```

`NLVector`	*v*	The vector.
`int`	*n*	The length of the vector.
`int`	*nz*	The number of non-zeros in the vector.
`int`	**el*	The indices of the non-zero coordinates.
`double`	**vl*	The values of the non-zero coordinates.

Description

This routine, `NLCreateVectorWithSparseData` allocates and initializes an NLVector data structure of a given length and coordinates. The vector returned has *nz* non-zero coordinates, given in the list *el* and with values from the array *vl*. The coordinates may be changed using the `NLVSetC` routine (page 106). Zero vectors and vectors with no non-zero coordinates can be created with the `NLCreateVector` (page 96) and `NLCreateVectorWithFullData` (page 98) subroutines.

Note that the coordinates and values are copied out of the arrays, so subsequent changes to the *el* and *vl* arrays do not effect the vector.

The NLVector data structure uses reference counting. The data structure should be deleted using the `NLFreeVector` subroutine (page 101). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefVector` subroutine (page 102).

Errors

Severity 4 errors return a vector with no nonzero coordinates. Severity 12 errors return (NLVector)NULL.

Message	Severity
"Length of Vector %d (argument 1) is Illegal. Must be positive."	12
"Number of nonzeros in vector %d (argument 2) is Illegal. Must be nonnegative."	12
"The pointer to the array of nonZeros (argument 3) is NULL"	4
"The pointer to the array of coordinates (argument 4) is NULL"	4
"Out of memory, trying to allocate %d bytes"	12

NLCreateVectorWithFullData

Purpose

Allocates and initializes an NLVector data structure of a given length.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
v=NLCreateVectorWithFullData(n,vl);
```

`NLVector`	*v*	The vector.
`int`	*n*	The length of the vector.
`double`	**vl*	The values of the coordinates.

Description

The routine This routine, `NLCreateVectorWithFullData` returns an NLVector data structure of a given length and coordinates. The vector returned has all coordinates marked as non-zero.

The coordinates may be changed using the `NLVSetC` routine (page 106). Zero vectors and sparse vectors can be created with the `NLCreateVector` (page 95) and `NLCreateVectorWithSparseData` (page 96) subroutines.

Note that the coordinates and values are copied out of the *vl* array, so subsequent changes to it do not effect the vector.

The NLVector data structure uses reference counting. The data structure should be deleted using the `NLFreeVector` subroutine (page 101). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefVector` subroutine (page 102).

Errors

Severity 4 errors return a vector with no nonzero coordinates. Severity 12 errors return (NLVector)NULL.

Message	Severity
"Length of Vector %d (argument 1) is Illegal. Must be positive."	12
"The pointer to the array of coordinates (argument 2) is NULL"	4
"Out of memory, trying to allocate %d bytes"	12

NLVector NLCreateDenseWrappedVector(int n,double *data)

NLCreateDenseWrappedVector

Purpose

Allocates and initializes an NLVector data structure of a given length with data at a given storage location.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
v=NLCreateDenseWrappedVector(n, data);
```

`NLVector`	*v*	The vector.
`int`	*n*	The length of the vector.
`double*`	*data*	The buffer for the coordinates of the vector.

Description

The routine `NLCreateDenseWrappedVector` allocates an NLVector data structure and sets the coordinates to reference a given buffer. The coordinates may be changed with the `NLVSetC` routine (page 106) or by changing the buffer.

The NLVector data structure uses reference counting. The data structure should be deleted using the `NLFreeVector` subroutine (page 101). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefVector` subroutine (page 102).

Errors

Severity 12 errors return (NLVector)NULL.

Message	Severity
”Length of Vector %d (argument 1) is Illegal. Must be positive.”	12
”Out of memory, trying to allocate %d bytes”	12

NLFreeVector

Purpose

Frees the storage associated with an NLVector data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLFreeVector(v);
```

`NLVector`	*v*	The vector.

Description

The NLVector data structure uses reference counting. This routine should be used to indicate that a vector is no longer needed. It will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefVector` subroutine (page 102).

Errors

Severity 4 errors return without changing the vector.

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLRefVector

Purpose

Registers a reference to an NLVector data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLRefVector(v);
```

`NLVector`	*v*	The vector.

Description

The NLVector data structure uses reference counting. This routine should be used to indicate that a vector is needed by another data structure. The vector will not be deleted until the same data structure indicates that it no longer needed (for example, when the data structure itself is deleted). This works as long as the `NLFreeVector` subroutine (page 101) is used to delete the vector, and is only used once per added reference.

Errors

Severity 4 errors return without changing the vector.

Message	Severity
"Pointer to Vector (argument 1) is NULL"	4

NLPrintVector

Purpose

Prints an NLVector.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLPrintVector(fid, v);
```

`FILE`	**fid*	The output file.
`NLVector`	*v*	The vector.

Description

The routine `NLPrintVector` prints an NLVector. The output presents the vector as a dense vector. Long vectors print the first and last few coordinates.

Errors

Errors return without printing the vector.

Message	Severity
”File pointer (argument 1) is NULL”	12
”Vector (argument 2) is NULL”	12

NLVGetNC

Purpose

Returns the dimension (the number of coordinates) of a vector.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLVGetNC(v);
```

`int`	n	The number of non-zero coordinates.
`NLVector`	v	The vector.

Description

This routine returns the dimension, length or number of coordinates of a vector. This is set when the vector is created.

Errors

Severity 4 errors return -1.

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLVGetC

Purpose

Returns a coordinate of a vector.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
c=NLVGetC(v,i);
```

`double`	c	The value of the coordinate.
`NLVector`	v	The vector.
`int`	i	Which coordinate to return.

Description

This routine returns the value of a coordinate of a vector. The index i must be nonnegative and less than the number of coordinates (`NLVGetNC`).

Errors

Severity 4 errors return a DBL_QNAN.

Message	Severity
"Pointer to Vector (argument 1) is NULL"	4
"Coordinate %d (argument 2) is illegal. Must be in 0 to %d"	4

NLVSetC

Purpose

Sets the specified coordinate of a vector.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLVSetC(v,i,c);
```

`int`	*rc*	The return code.
`NLVector`	*v*	The vector.
`int`	*i*	Which coordinate to return.
`double`	*c*	The value of the coordinate.

Description

This routine changes the value of a coordinate of a vector. The index *i* must be nonnegative and less than the number of coordinates (`NLVGetNC`). If the coordinate is not currently set, it is added to the list of coordinates.

Errors

Errors return a 0, normal execution returns 1. Errors return without changing the vector.

Message	Severity
"Pointer to Vector (argument 1) is NULL"	4
"Coordinate %d (argument 2) is illegal. Must be in 0 to %d"	4
"Out of memory, trying to allocate %d bytes"	12

NLCopyVector

Purpose

Returns a copy of a vector.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
v=NLCopyVector(u);
```

`NLVector`	*v*	The copy.
`NLVector`	*u*	The vector.

Description

This routine returns a copy of a vector. The copy is of the same type (dense/sparse) as the original, and has the same value. **Errors**
Severity 4 errors return (NLVector)NULL.

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLVSetToZero

Purpose

Sets a vector to zero.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLVSetToZero(u);
```

`NLVector`	u	The vector.

Description

This routine sets all coordinates of a vector to zero. **Errors**
Severity 4 errors return (NLVector)NULL.

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLVIncrementC

Purpose

Increments a coordinate of a vector.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLVIncrementC(u,i,vl);
```

`int`	*rc*	The return code. 1 indicates success.
`NLVector`	*u*	The vector.
`int`	*i*	The coordinate to increment.
`double`	*vl*	The amount to add to the coodinate.

Description

This routine adds a value to a coordinate of a vector. **Errors** Severity 4 errors return (NLVector)NULL.

Message	Severity
"Pointer to Vector (argument 1) is NULL"	4
"Coordinate %d (argument 2) is illegal. Must be in 0 to %d"	4
"Out of memory, trying to allocate %d bytes"	12

NLVInnerProd

Purpose

Returns the inner product (Euclidean) of two vectors.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
p=NLVInnerProd(u,v);
```

`double`	p	The inner product.
`NLVector`	u	The first vector.
`NLVector`	v	The second vector.

Description

This routine returns the Euclidean inner product of two vectors. **Errors** Severity 4 errors return (NLVector)NULL.

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4
”Pointer to Vector (argument 2) is NULL”	4

NLVPlusV

Purpose

Returns the sum of two vectors (actually the daxpy).

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
p=NLVPlusV(u,v,a);
```

`NLVector`	u	The first vector.
`NLVector`	v	The second vector.
`double`	a	The multiplication factor.

Description

This routine sets $u = u{+}a{*}v$. **Errors**

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4
”Pointer to Vector (argument 2) is NULL”	4

NLNegateVector

Purpose

Sets a vector to it's product with -1.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLNegateVector(u);
```

`NLVector`	*u*	The vector.

Description

This sets a vector to its negative. **Errors**

Message	Severity
"Pointer to Vector (argument 1) is NULL"	4

NLVSparse

Purpose

Queries if a vector is sparse.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
flag=NLVSparse(u);
```

`int`	*flag*	The answer. 1 indicates sparse, 0 dense
`NLVector`	*u*	The vector.

Description

This routine returns 1 if the vector is a sparse vector (otherwise returns 0).

Errors

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLVnNonZeros

Purpose

For a sparse vector returns a pointer to the array containing the list of nonzeros.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
c=NLVnNonZeros(u);
```

`int*`	c	The list of which coordinates are nonzero.
`NLVector`	u	The vector.

Description

This routine returns a pointer to the array of which coordinates of a vector are nonzero if the vector is a sparse vector (otherwise returns (int*)NULL). Note that this array may be reallocated when a coordinate becomes nonzero. In that case the pointer is no longer valid and should not be used (get a new pointer!). **Errors**

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLVnonZero

Purpose

For a sparse vector returns a pointer to the array containing the list of nonzeros.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
c=NLVnonZero(u);
```

`int*`	c	The list of which coordinates are nonzero.
`NLVector`	u	The vector.

Description

This routine returns a pointer to the array of which coordinates of a vector are nonzero if the vector is a sparse vector (otherwise returns (int*)NULL). Note that this array may be reallocated when a coordinate becomes nonzero. In that case the pointer is no longer valid and should not be used (get a new pointer!). **Errors**

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLVGetNumberOfNonZeros

Purpose

Returns the number of nonzeros (if sparse) or the number of coordinates (if dense).

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLVGetNumberOfNonZeros(u);
```

`int`	n	The answer.
`NLVector`	u	The vector.

Description

This routine returns the number of nonzeros if the vector is a sparse vector (otherwise returns the total umber of coordinates). **Errors**

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLVGetNonZeroCoord

Purpose

Returns the requested nonZero (counting only nonzeros for sparse vectors).

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
c=NLVGetNonZeroCoord(u,i);
```

`double`	*c*	The value of the coordinate.
`NLVector`	*u*	The vector.
`int`	*i*	The answer.

Description

This routine returns the ith nonzero if the vector is a sparse vector (otherwise returns the ith coordinate). **Errors**

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4
”NonZero Coordinate %d (argument 2) is illegal. Must be in 0 to %d”	12

NLVGetNonZero

Purpose

Returns the coordinate index of a non-zero coordinate in a vector.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
c=NLVGetNonZero(v,i);
```

`double`	c	The value of the th non-zero coordinate.
`NLVector`	v	The vector.
`int`	i	Which non-zero coordinate.

Description

This routine returns the value of a non-zero coordinate. The argument i must be nonnegative and less than the number of non-zero coordinates `NLVGetNumber of NonZeros`.

Errors

Errors return a DBL_QNAN.

Message	Severity
"Pointer to Vector (argument 1) is NULL"	4
"NonZero Coordinate %d (argument 2) is illegal. Must be in 0 to %d"	12

NLVWrapped

Purpose

Queries if a vector is wrapped.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
flag=NLVWrapped(u);
```

`int`	*flag*	The answer. 1 indicates wrapped, 0 dense
`NLVector`	*u*	The vector.

Description

This routine returns 1 if the vector is a wrapped vector (otherwise returns 0).

Errors

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLVData

Purpose

Returns a pointer to the data array of a vector.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
data=NLVData(u);
```

`int`	*data*	The data array.
`NLVector`	*u*	The vector.

Description

This routine returns the data array of a vector. If the vector is sparse this contains the packed nonzeros. If dense it contains the coordinates.

Errors

Message	Severity
”Pointer to Vector (argument 1) is NULL”	4

NLCreateMatrix

Purpose

Allocates and initializes an NLMatrix data structure of a given size.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
A=NLCreateMatrix(n,m);
```

`NLMatrix`	A	The matrix.
`int`	n	The number of rows in the matrix.
`int`	m	The number of columns in the matrix.

Description

The routine `NLCreateMatrix` allocates and initializes an NLMatrix data structure of a given size. The matrix returned has all elements zero, until they are set with the `NLMSetElement` routine (page 133). The `NLCreateMatrixWithData` subroutine (page 122) can be used to created Matrices with supplied elements.

The NLMatrix data structure uses reference counting. The data structure should be deleted using the `NLFreeMatrix` subroutine (page 129). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefMatrix` subroutine (page 128).

Errors

Errors return (NLMatrix)NULL.

Message	Severity
"Number of rows %d (argument 1) is negative."	12
"Number of columns %d (argument 2) is negative."	12
"Out of memory, trying to allocate %d bytes"	12
"Out of memory, trying to allocate %dx%d matrix (%d bytes)"	12

NLCreateMatrixWithData

Purpose

Allocates and initializes an NLMatrix data structure of a given size with given elements.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
A=NLCreateMatrixWithData(n,m,aij);
```

`NLMatrix`	A	The matrix.
`int`	n	The number of rows in the matrix.
`int`	m	The number of columns in the matrix.
`double`	$*aij$	The entries of the matrix.

Description

The routine `NLCreateMatrix` allocates and initializes an NLMatrix data structure of a given size with given elements. The elements may be changed later with the `NLMSetElement` routine (page 133). Zero Matrices can be created with the `NLCreateMatrix` subroutine (page 121).

The NLMatrix data structure uses reference counting. The data structure should be deleted using the `NLFreeMatrix` subroutine (page 129). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefMatrix` subroutine (page 128).

Errors

Severity 12 errors return (NLMatrix)NULL. Severity 4 errors return a matrix with all entries zero.

Message	Severity
”Number of rows (argument 1) is negative %d”	12
”Number of columns (argument 2) is negative %d”	12
”Pointer to data (argument 3) is NULL”	4
”Out of memory, trying to allocate %d bytes”	12
”Out of memory, trying to allocate %dx%d matrix (%d bytes)”	12

NLCreateSparseMatrix

Purpose

Allocates and initializes an NLMatrix data structure of a given size.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
A=NLCreateSparseMatrix(n,m);
```

`NLMatrix`	*A*	The matrix.
`int`	*n*	The number of rows in the matrix.
`int`	*m*	The number of columns in the matrix.

Description

The routine `NLCreateSparseMatrix` allocates and initializes a sparse NLMatrix data structure of a given size. Only the non-zeros are stored. The matrix returned has all elements zero, until they are set with the `NLMSetElement` routine (page 133), or incremented by the `LNMIncrementElement` routine (page 134).

The NLMatrix data structure uses reference counting. The data structure should be deleted using the `NLFreeMatrix` subroutine (page 129). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefMatrix` subroutine (page 128).

Errors

Errors return (NLMatrix)NULL.

Message	Severity
"Number of rows %d (argument 1) is negative."	12
"Number of columns %d (argument 2) is negative."	12
"Out of memory, trying to allocate %d bytes"	12
"Out of memory, trying to allocate %dx%d matrix (%d bytes)"	12

NLCreateWSMPSparseMatrix

Purpose

Allocates and initializes a square NLMatrix data structure of a given size. The matrix is stored in sparse format where a list of the column for each nonzero is stored, contiguous by rows, and ordered from first row to last. An array is also stored with the index of the beginning of each row in the column array.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
A=NLCreateWSMPSparseMatrix(n);
```

`NLMatrix`	*A*	The matrix.
`int`	*n*	The number of rows and columns in the matrix.

Description

The routine `NLCreateMatrix` allocates and initializes a square NLMatrix data structure of a given size. The matrix is stored in sparse format where a list of the column for each nonzero is stored, contiguous by rows, and ordered from first row to last. An array is also stored with the index of the beginning of each row in the column array. Initially there are no nonzeros. As elements are set with the `NLMSetElement` routine (page 133), new nonzeros are added to the matrix.

The NLMatrix data structure uses reference counting. The data structure should be deleted using the `NLFreeMatrix` subroutine (page 129). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefMatrix` subroutine (page 128).

Errors

Errors return (NLMatrix)NULL.

Message	Severity
"Number of rows %d (argument 1) is negative."	12
"Out of memory, trying to allocate %d bytes"	12
"Out of memory, trying to allocate %dx%d matrix (%d bytes)"	12

NLCreateDenseWrappedMatrix

Purpose

Allocates and initializes a dense NLMatrix data structure of a given size, with a data array provided. If the user later changes the array the NLMatrix will see the changes.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
A=NLCreateDenseWrappedMatrix(n,m,data);
```

`NLMatrix`	*A*	The matrix.
`int`	*n*	The number of rows in the matrix.
`int`	*m*	The number of columns in the matrix.
`double*`	*data*	The data array.

Description

The routine `NLCreateDenseWrappedMatrix` allocates and initializes an NLMatrix data structure of a given size. The matrix returned has the data array specified.

The NLMatrix data structure uses reference counting. The data structure should be deleted using the `NLFreeMatrix` subroutine (page 129). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefMatrix` subroutine (page 128).

Errors

Errors return (NLMatrix)NULL.

Message	Severity
”Number of rows %d (argument 1) is negative.”	12
”Number of columns %d (argument 2) is negative.”	12

NLRefMatrix

Purpose

Registers a reference to an NLMatrix data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLRefMatrix(A);
```

`NLMatrix` *A* The matrix.

Description

The NLMatrix data structure uses reference counting. This routine should be used to indicate that a vector is needed by another data structure. The vector will not be deleted until the same data structure indicates that it no longer needed (for example, when the data structure itself is deleted). This works as long as the `NLFreeMatrix` subroutine (page 129) is used to delete the vector, and is only used once per added reference.

Errors

Error returns without changing the matrix.

Message	Severity
"Matrix (argument 1) is NULL"	4

NLFreeMatrix

Purpose

Frees the storage associated with an NLMatrix data structure.

Library

```
libNLPAPI.a
```

C Syntax

```
#include <NLPAPI.h>
void NLFreeMatrix(A);
```

`NLMatrix`	*A*	The matrix.

Description

The NLMatrix data structure uses reference counting. This routine should be used to indicate that a vector is no longer needed. It will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefMatrix` subroutine (page 128).

Errors

Error returns without changing the matrix.

Message	Severity
"Matrix (argument 1) is NULL"	4

NLMGetNumberOfRows

Purpose

Returns the number of rows in an NLMatrix.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLMGetNumberOfRows(A);
```

`int`	n	The number of rows.
`NLMatrix`	A	The matrix.

Description

This routine returns the number of rows in the matrix. This is set when the matrix is created.

Errors

Errors return -1.

Message	Severity
”Matrix (argument 1) is NULL”	12

NLMGetNumberOfCols

Purpose

Returns the number of columns in an NLMatrix.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
m=NLMGetNumberOfCols(A);
```

`int`	m	The number of columns.
`NLMatrix`	A	The matrix.

Description

This routine returns the number of columns in the matrix. This is set when the matrix is created.

Errors

Errors return -1.

Message	Severity
”Matrix (argument 1) is NULL”	12

NLMGetElement

Purpose

Returns an element of an NLMatrix.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
aij=NLMGetElement(A,i,j);
```

`double`	*aij*	The element of the matrix.
`NLMatrix`	*A*	The matrix.
`int`	*i*	The row index of the element.
`int`	*j*	The column index of the element.

Description

This routine returns the specified element of the matrix. This is set when the matrix is created, or with the `NLMSetElement` routine (page 133).

Errors

Errors return DBL_QNAN.

Message	Severity
"Matrix (argument 1) is NULL"	12
"Row index %d (argument 2) is negative."	12
"Row index %d (argument 2) is too large. Must be less than %d"	12
"Column index %d (argument 3) is negative".	12
"Column index %d (argument 3) is too large. Must be less than %d"	12

NLMSetElement

Purpose

Changes the value of an element of an NLMatrix.

Library

```
libNLPAPI.a
```

C Syntax

```
#include <NLPAPI.h>
rc=NLMSetElement(A,i,j,aij);
```

`int`	*rc*	The return code.
`NLMatrix`	*A*	The matrix.
`int`	*i*	The row index of the element.
`int`	*j*	The column index of the element.
`double`	*aij*	The element of the matrix.

Description

This routine changes the specified element of the matrix.

Errors

Errors return 0, with no changes to the matrix. Normal execution returns 1.

Message	Severity
"Matrix (argument 1) is NULL"	12
"Row index %d (argument 2) is negative."	12
"Row index %d (argument 2) is too large. Must be less than %d"	12
"Column index %d (argument 3) is negative".	12
"Column index %d (argument 3) is too large. Must be less than %d"	12

NLMIncrementElement

Purpose

Increments the value of an element of an NLMatrix.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLMIncrementElement(A,i,j,aij);
```

`int`	*rc*	The return code.
`NLMatrix`	*A*	The matrix.
`int`	*i*	The row index of the element.
`int`	*j*	The column index of the element.
`double`	*aij*	The increment element of the matrix.

Description

This routine changes the specified element of the matrix, by adding the specified increment. If the matrix is sparse, and the element does not have a value, the value is set to the increment.

Errors

Errors return 0, with no changes to the matrix. Normal execution returns 1.

Message	Severity
"Matrix (argument 1) is NULL"	12
"Row index %d (argument 2) is negative."	12
"Row index %d (argument 2) is too large. Must be less than %d"	12
"Column index %d (argument 3) is negative".	12
"Column index %d (argument 3) is too large. Must be less than %d"	12

NLMatrixDoubleProduct

Purpose

Computes the product $u^T Av$.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
p=NLMatrixDoubleProduct(u,A,v);
```

`double`	p	The return code.
`NLVector`	u	The vector operating on the left.
`NLMatrix`	A	The matrix.
`NLVector`	v	The vector operating on the right.

Description

This routine returns the product $u^T Av$. If A is an $n \times m$ matrix v must be an m-vector and u must be an n-vector. **Errors**

Message	Severity
"left vector (argument 1) is NULL"	12
"Matrix (argument 2) is NULL"	12
"right vector (argument 1) is NULL"	12
"Cannot find $u^T Av$ for a %d vector a %dx%d matrix and a %d vector."	12

NLMVMult

Purpose

Computes the product $b = Ax$.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLMVMult(A,x,b);
```

`NLMatrix`	A	The matrix.
`double*`	x	An array containing the coordinates of the vector x.
`double*`	b	An array for the result.

Description

This routine returns the product $b = Ax$. If A is an $n \times m$ matrix x must be an array of length m, and b must be an array of length n. **Errors**

Message	Severity
”A (first argument) is NULL”	12
”x (second argument) is NULL”	12
”b (third argument) is NULL”	12

NLMVMultT

Purpose

Computes the product $b = A^T x$.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLMVMultT(A,x,b);
```

`NLMatrix`	A	The matrix.
`double*`	x	An array containing the coordinates of the vector x.
`double*`	b	An array for the result.

Description

This routine returns the product $b = A^T x$. If A is an $n \times m$ matrix x must be an array of length n, and b must be an array of length m.

Errors

Message	Severity
"A (first argument) is NULL"	12
"x (second argument) is NULL"	12
"b (third argument) is NULL"	12

NLMSetToZero

Purpose

Sets all elements of an NLMatrix to zero.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLMSetToZero(A,i,j,aij);
```

`int`	*rc*	The return code. (1 indicates success)
`NLMatrix`	*A*	The matrix.

Description

This routine sets all elements of a matrix to zero. For sparse matrices the nonzero structure is not changed, even though the element values are set to zero. **Errors**
Errors return 0, with no changes to the matrix. Normal execution returns 1.

Message	Severity
”Matrix (argument 1) is NULL”	12

NLMatrixClone

Purpose

Creates a matrix of the same type and size of another, with the same element values.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
B=NLMatrixClone(A);
```

`NLMatrix`	B	The clone.
`NLMatrix`	A	The matrix.

Description

This routine creates a new matrix with a deep copy (i.e. new element arrays are allocated and the values copied). The result is a matrix of the same type as the original. **Errors**
Errors return (NLMatrix)NULL.

Message	Severity
”Matrix (argument 1) is NULL”	12

NLGetGershgorinBounds

Purpose

Computes bounds (using Gershgorin disks) of the leftmost eigenvalue of a matrix (with an optional diagonal scaling).

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLGetGershgorinBounds(A,M,L,U);
```

`NLMatrix`	A	The matrix.
`double*`	M	The vector giving the diagonal scaling.
`double*`	L	A double to hold the lower bound.
`double*`	U	A double to hold the upper bound.

Description

This routine provides bounds for the leftmost eigenvalue of the matrix. If the diagonal scaling M is not null the matrix is scaled by $\mathrm{diag}(1./sqrtM)A\mathrm{diag}(1./sqrtM)$. The method is simply to use the Gershgorin bounds. **Errors**

Message	Severity
"Matrix (first argument), is NULL"	12
"Address for the lower bound (third argument), is NULL"	12
"Address for the upper bound (fourth argument), is NULL"	12

NLMatrixOneNorm

Purpose

Computes the 1-norm of a matrix (with an optional diagonal scaling).

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
L1=NLMatrixOneNorm(A,M);
```

`double`	L_1	The norm.
`NLMatrix`	A	The matrix.
`double*`	M	The vector giving the diagonal scaling.

Description

This routine returns the one norm of the matrix. If the diagonal scaling M is not null the matrix is scaled by diag$(1./sqrtM)A$diag$(1./sqrtM)$.

Errors

Message	Severity
"Matrix (first argument), is NULL"	12

NLMSumSubMatrixInto

Purpose

Adds a sub-matrix with a symmetric non-zero structure into a matrix $A = A + s * a$.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLMSumSubMatrixInto(A,s,n,r,a);
```

`NLMatrix`	A	The matrix.
`double`	s	A scale factor.
`int`	n	The number of rows/columns of the submatrix.
`int*`	r	A vector of length n giving the rows/columns of the submatrix.
`double*`	a	An $n \times n$ matrix giving the elements of the submatrix (stored by columne – $a_{ij} = a[i + n * j]$).

Description

This routine adds a submatrix (with symmetric non-zero structure) into a matrix. That is,

$$A_{r[i],r[j]} = A_{r[i],r[j]} + s * a_{i,j}$$

Errors

Message	Severity
"Matrix (first argument), is NULL"	12
"Array of row indices (fourth argument), is NULL"	12
"Array of submatrix elements (fifth argument), is NULL"	12

NLMSumRankOneInto

Purpose

Adds a rank one matrix into a matrix $A = A + s * vv^T$.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLMSumRankOneInto(A,s,v);
```

`NLMatrix`	A	The matrix.
`double`	s	A scale factor.
`double*`	v	An array of length n giving the elements of the vector.

Description

This routine adds a submatrix (with symmetric non-zero structure) into a matrix. That is,

$$A_{i,j} = A_{i,j} + s * v_i v_j$$

Errors

Message	Severity
"Matrix (first argument), is NULL"	12
"Vector (third argument), is NULL"	12

NLMMMMProd

Purpose

Computes the matrix-matrix-matrix product $B = M^T AM$.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLMMMMProd(A,M,B);
```

`NLMatrix`	A	An $n \times n$ matrix.
`double*`	M	An array containing the dense matrix M (stored by column – $M_{ij} = M[i + n * j]$).
`double*`	v	An array of length $n * n$ for the result (will be stored by column – $B_{ij} = B[i + n * j]$).

Description

This routine computes the matrix-matrix-matrix product

$$B_{i,j} = \sum_k \sum_l M_{i,k} A_{k,l} M_{l,j}$$

Errors

Message	Severity
”Matrix (first argument), is NULL”	12
”Matrix (second argument), is NULL”	12
”Result (third argument), is NULL”	12

NLMSparse

Purpose

Queries if a matrix is sparse.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
flag`=NLMSparse(`*A*`);`

`int`	*flag*	The answer. 0 indicates dense, 1 sparse stored by row and column index, 2 WSMP format.
`NLMatrix`	*A*	The matrix.

Description

This routine returns 0 if the matrix is a dense matrix. If it is stored by row and column index a ”1” is returned and the matrix is represented by arrays *int *row*, *int *col* and *double *data* so that

$$A_{row[i],col[i]} = data[i]$$

If the matrix is stored in “WSMP format”, i.e. as a set of sparse rows, this routine returns a ”2”. In this case the same three arraies are used, but each non-zero has an entry in the *col* array, and the *row* array indicates where in each row begins in the *col* array. That is, if $0 \leq l < n$ is such that

$$row[l] \leq i < row[l+1]$$

then

$$A_{row[l],col[i]} = data[i]$$

Errors

Message	Severity
”Matrix (argument 1) is NULL”	4

NLMDetermineHessianSparsityStructure

Purpose

Updates the sparsity structure of a matrix to accomodate the nonzeros in the Hessian of the objective or a constraint of a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NLMDetermineHessianSparsityStructure(P,type,c,H);
```

`NLProblem`	*P*	The problem.
`char`	*type*	Which Hessian to update – 'O' objective, 'I' inequality constraint, 'E' equality constraint.
`int`	*i*	Which constraint (if type is not 'O').
`NLMatrix`	*A*	The Hessian (the user is responible for allocating this).

Description

This routine updates the sparsity structure of a matrix to accomodate the nonzeros in the Hessian of the objective or a constraint of a problem. **Errors**

Message	Severity
"Problem (first argument), is NULL"	12
"type %c (second argument), is not valid. Must be 'O', 'I', 'E', or 'M'"	12
"Hessian (fourth argument), is NULL."	12

NLMData

Purpose

Returns a pointer to the data array of the matrix.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
data=NLMData(A);
```

`double*`	*data*	The data array.
`NLMatrix`	*A*	The matrix.

Description

This routine returns a pointer to the internal data array. The contents of the array varies according to the sparsity structure (see the NLMSparse routine on page 145 for a description of the uses of the array). **Errors**

Message	Severity
”Matrix (argument 1) is NULL”	4

NLMnE

Purpose

Returns the number of “nonzero” entries in a matrix.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
nE=NLMnE(A);
```

`double*`	nE	The number of nonzeros.
`NLMatrix`	A	The matrix.

Description

Thiis routine returns the number of “nonzero” entries in a matrix. Note that this is the number of possible nonzeros, not the number of actual nonzeros. So for an $n{\times}m$ matrix stored as a dense matrix the result is always $n{*}m$. For matrices stored in one of the sparse formats it is the number of allocated nonzeros. **Errors**

Message	Severity
”Matrix (argument 1) is NULL”	4

NLMRow

Purpose

Returns a pointer to the “row” array of the matrix.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
row=NLMRow(A);
```

`double*`	*row*	The row array.
`NLMatrix`	*A*	The matrix.

Description

This routine returns a pointer to the internal “row” array. The contents of the array varies according to the sparsity structure (see the NLMSparse routine on page 145 for a description of the uses of the array). **Errors**

Message	Severity
”Matrix (argument 1) is NULL”	4

NLMCol

Purpose

Returns a pointer to the “col” array of the matrix.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
col=NLMCol(A);
```

`double*`	*col*	The col array.
`NLMatrix`	*A*	The matrix.

Description

This routine returns a pointer to the internal “col” array. The contents of the array varies according to the sparsity structure (see the NLMSparse routine on page 145 for a description of the uses of the array). **Errors**

Message	Severity
”Matrix (argument 1) is NULL”	4

NLPrintMatrix

Purpose

Prints an NLMatrix.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLPrintMatrix(fid, v);
```

`FILE`	**fid*	The output file.
`NLMatrix`	*v*	The matrix.

Description

The routine `NLPrintMatrix` prints an NLMatrix. The output presents the matrix as a dense matrix. Large matrices print the whole thing, so watch out!

Errors

Errors return without printing the matrix.

Message	Severity
"File pointer (argument 1) is NULL"	12
"Matrix (argument 2) is NULL"	12

NLCreateGroupFunctionByString

Purpose

Allocates and initializes an NLGroupFunction data structure by way of an expression.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
G=`NLCreateGroupFunctionByString(`*P*`,`*type*`,`*var*`,`*expr*`);`

`NLGroupFunction` *G*	The group function.
`NLProblem` *P*	The problem to which the group function belongs.
`char *`*type*	A name associated to the group function.
`char *`*var*	The identifer used in the expression for the argument of the group function.
`char *`*expr*	An expression for the group function.

Description

The routine `NLCreateGroupFunctionByString` allocates and initializes an NLGroupFunction data structure. The *var* string (e.g. "s" or "[s]") gives the identifier used in the expression string (e.g. "sin(s)") for the argument of the group function.

The NLGroupFunction data structure uses reference counting. The data structure should be deleted using the `NLFreeGroupFunction` subroutine (page 157). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefGroupFunction` subroutine (page 156).

Errors

Errors return (NLGroupFunction)NULL.

Message	Severity
”Problem (argument 1) is NULL”	12
”type (argument 2) is NULL”	12
”var (argument 3) is NULL”	12
”expr (argument 4) is NULL”	12
”Out of memory, trying to allocate %d bytes”	12

NLCreateGroupFunction

Purpose

Allocates and initializes an NLGroupFunction data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
G=NLCreateGroupFunction(P,type,g,dg,ddg,data,freeData);
```

`NLGroupFunction` *G*	The group function.
`NLProblem` *P*	The problem to which the group function belongs.
`char *`*type*	A name associated to the group function.
`double` *g*`(double,void*)`	The routine giving the value of the function.
`double` *dg*`(double,void*)`	The routine giving the value of the derivative of the function.
`double` *ddg*`(double,void*)`	The routine giving the value of the second derivative of the function.
`void` **data*	A pointer to user specific data which is to be provided to the function.
`void` *(*freeData)(void*)*	A pointer to a routine to free user specific data, or (void (*)(void*)NULL). Will be called when the element function is deleted.

Description

The routine `NLCreateGroupFunction` allocates and initializes an NLGroupFunction data structure.

The NLGroupFunction data structure uses reference counting. The data structure should be deleted using the `NLFreeGroupFunction` subroutine (page 157). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefGroupFunction` subroutine (page 156).

Errors

Errors return (NLGroupFunction)NULL.

Message	Severity
”Out of memory, trying to allocate %d bytes”	12

NLRefGroupFunction

Purpose

Registers a reference to an NLGroupFunction data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLRefGroupFunction(G);
```

`NLGroupFunction`	G	The group function.

Description

The NLGroupFunction data structure uses reference counting. This routine should be used to indicate that a vector is needed by another data structure. The vector will not be deleted until the same data structure indicates that it no longer needed (for example, when the data structure itself is deleted). This works as long as the `NLFreeGroupFunction` subroutine (page 157) is used to delete the vector, and is only used once per added reference.

Errors

Errors return without changing the group function.

Message	Severity
"Group Function (argument 1) is NULL"	12

NLFreeGroupFunction

Purpose

Frees the storage associated with an NLGroupFunction data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLFreeGroupFunction(G);
```

`NLGroupFunction`	*G*	The group function.

Description

The NLGroupFunction data structure uses reference counting. This routine should be used to indicate that a vector is no longer needed. It will decrement the reference count and free the storage if the count goes to zero. The `NLRefGroupFunction` subroutine (page 156) may be used to add references.

Errors

Errors return with changing the group function.

Message	Severity
"Group Function (argument 1) is NULL"	4

NLGEval

Purpose

Evaluates an NLGroupFunction.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
g=NLGEval(G,x);
```

`double`	g	The value of the group function.
`NLGroupFunction`	G	The group function.
`double`	x	The point.

Description

This routine returns the value of a group function $g(x)$.

Errors

Errors return DBL_QNAN.

Message	Severity
”Group Function (argument 1) is NULL”	12
”Group Function function is NULL”	12

NLGEvalDer

Purpose

Evaluates the derivative of an NLGroupFunction.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
g=NLGEvalDer(G,x);
```

`double`	g	The value of the derivative.
`NLGroupFunction`	G	The group function.
`double`	x	The point.

Description

This routine returns the value of the derivative of a group function d$g(x)$/dx.

Errors

Errors return DBL_QNAN.

Message	Severity
"Group Function (argument 1) is NULL"	12
"Group Function Derivative function is NULL"	12

NLGEvalSecDer

Purpose

Evaluates the second derivative of an NLGroupFunction.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
g=NLGEvalSecDer(G,x);
```

`double`	g	The value of the second derivative.
`NLGroupFunction`	G	The group function.
`double`	x	The point.

Description

This routine returns the value of the second derivative of a group function d$g(x)$/dx.

Errors

Errors return DBL_QNAN.

Message	Severity
"Group Function (argument 1) is NULL"	12
"Group Function Second Derivative function is NULL"	12

NLCreateElementFunctionByString

Purpose

Allocates and initializes an NLElementFunction data structure by means of an expression.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
F=`NLCreateElementFunctionByString`(*P*, *type*, *n*, *R*, *vars*, *expr*);

`NLElementFunction` *F*	The element function.
`NLProblem` *P*	The problem.
`char *`*type*	The type given to the new element function.
`int` *n*	The number of element variables.
`NLMatrix` *R*	The range transformation, or (NLMatrix)NULL if the identity.
`char *`*vars*	A “[]“ delimited, comma separated list of the identifiers used in the expression for the internal variables of the element function.
`char *`*expr*	An expression giving the element function.

Description

The routine `NLCreateElementFunctionByString` allocates and initializes an NLElementFunction data structure. The *vars* string (e.g. ”[x,y,z]”) gives a list of the identifiers used in the expression for the internal variables of the element. The range transformation *R* is used to map from element variables (a subset of the problem variables) to internal variables. The order in the *vars* string is the order in the vector produced by the range transformation.

The NLElementFunction data structure uses reference counting. The data structure should be deleted using the `NLFreeElementFunction` subroutine (page 166). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefElementFunction` subroutine (page 165).

Errors

Errors return (NLElementFunction)NULL.

Message	Severity
”Problem (argument 1) is NULL”	12
”type (argument 2) is NULL”	12
”vars (argument 5) is NULL”	12
”expr (argument 6) is NULL”	12
”Out of memory, trying to allocate %d bytes”	12
”Number of coordinates is not positive %d”	12

NLCreateElementFunction

Purpose

Allocates and initializes an NLElementFunction data structure.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
NLElementFunction NLCreateElementFunction(NLProblem P,char *type,int n,NLMatrix R,double (*f)(int,double*,void*),double (*df)(int,int,double*,void*),double (*dd f)(int,int,int,double*,void*),void *data,void (*freedata)(void*));
F=`NLCreateElementFunction(`*P*`,` *type*`,` *n*`,` *R*`,` *f*`,` *df*`,` *ddf*`,` *datafreeData*`);`

`NLElementFunction` *F*	The element function.
`NLProblem` *P*	The problem.
`char *`*type*	The type given to the new element function.
`int` *n*	The number of element variables.
`NLMatrix` *R*	The range transformation, or (NLMatrix)NULL the identity.
`double` *f*`(int,double*,void*)`	The routine giving the value of the function.
`double` *df*`(int,int,double*,void*)`	The routine giving the value of the derivative the function.
`double` *ddf*`(int,int,int,double*,void*)`	The routine giving the value of the second derivtive of the function.
`void` **data*	A pointer to user specific data which will passed to the function.
`void` *(*freeData)(void*)*	A pointer to a routine to free user specific data, (void (*)(void*)NULL). Will be called when t element function is deleted.

Description

The routine `NLCreateElementFunction` allocates and initializes an NLElementFunction data structure.

The NLElementFunction data structure uses reference counting. The data structure should be deleted using the `NLFreeElementFunction` subroutine (page 166). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefElementFunction` subroutine (page 165).

Errors

Errors return (NLElementFunction)NULL.

Message	Severity
"Out of memory, trying to allocate %d bytes"	12
"Number of coordinates is not positive %d"	12

NLRefElementFunction

Purpose

Registers a reference to an NLElementFunction data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLRefElementFunction(F);
```

`NLElementFunction`	F	The element function.

Description

The NLElementFunction data structure uses reference counting. This routine should be used to indicate that a vector is needed by another data structure. The vector will not be deleted until the same data structure indicates that it no longer needed (for example, when the data structure itself is deleted). This works as long as the `NLFreeElementFunction` subroutine (page 166) is used to delete the vector, and is only used once per added reference.

Errors

Errors return without chaning the element function.

Message	Severity
"Element Function (argument 1) is NULL"	12

NLFreeElementFunction

Purpose

Frees the storage associated with an NLElementFunction data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLFreeElementFunction(F);
```

`NLElementFunction`	*F*	The element function.

Description

The NLElementFunction data structure uses reference counting. This routine should be used to indicate that a vector is no longer needed. It will decrement the reference count and free the storage if the count goes to zero. The `NLRefElementFunction` subroutine (page 165) may be used to add References.

Errors

Errors return without chaning the element function.

Message	Severity
”Element Function (argument 1) is NULL”	12

NLEGetDimension

Purpose

Returns the number of unknowns (element internal variables) for an NLElementFunction.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLEGetDimension(F);
```

`int`	n	The number of unknowns.
`NLElementFunction`	F	The element function.

Description

This routine returns the number of unknowns for an element function.

Errors

Errors return -1.

Message	Severity
"Element Function (argument 1) is NULL"	12

NLEEval

Purpose

Evaluates an NLElementFunction.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
f=NLEEval(F,n,x);
```

`double`	f	The value of the element function.
`NLElementFunction`	F	The element function.
`int`	n	The number of coordinates.
`double`	$*x$	The point.

Description

This routine returns the value of a element function $f(x)$.

Errors

Errors return DBL_QNAN.

Message	Severity
"Element Function (argument 1) is NULL"	12
"Number of arguments to Element Function %d is illegal (argument 2). Must be %d. Argument 1 is %8.8x"	12
"Pointer to x (argument 3) is NULL. F is %8.8x"	12

NLEEvalDer

Purpose

Evaluates the derivative of an NLElementFunction.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
f=NLEEvalDer(F,i,n,x);
```

`double`	f	The value of the derivative.
`NLElementFunction`	F	The element function.
`int`	i	The first variable.
`int`	n	The number of coordinates in the point.
`double`	$*x$	The point.

Description

This routine returns the value of the derivative of a element function $\mathrm{d}f(x)/\mathrm{dx}_\mathrm{i}$.

Errors

Errors return DBL_QNAN.

Message	Severity
”Element Function (argument 1) is NULL”	12
”Direction %d (argument 2) is illegal. Must be in range 0 to %d Argument 1 is %8.8x”	12
”Number of arguments to Element Function %d is illegal (argument 3). Must be %d. Argument 1 is %8.8x”	12
”Pointer to x (argument 4) is NULL. F is %8.8x”	12

NLEEvalSecDer

Purpose

Evaluates the second derivative of an NLElementFunction.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
f=NLEEvalSecDer(F,i,j,n,x);
```

`double`	f	The value of the second derivative.
`NLElementFunction`	F	The element function.
`int`	i	The first variable.
`int`	j	The second variable.
`int`	n	The number of coordinates in the point.
`double`	$*x$	The point.

Description

This routine returns the value of the second derivative of a element function $d^2 f(x)/dx_i/dx_j$.

Errors

Errors return DBL_QNAN.

Message	Severity
"Element Function (argument 1) is NULL"	12
"Direction %d (argument 2) is illegal. Must be in range 0 to %d Argument 1 is %8.8x"	12
"Direction %d (argument 3) is illegal. Must be in range 0 to %d Argument 1 is %8.8x"	12
"Number of arguments to Element Function %d is illegal (argument 4). Must be %d. Argument 1 is %8.8x"	12
"Pointer to x (argument 5) is NULL. F is %8.8x"	12

NLCreateNonlinearElement

Purpose

Allocates and initializes an NLElementFunction data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
NE=NLCreateNonlinearElement(P,type,fn,vars);
```

`NLNonlinearElement`*NE*	The new nonlinear element.
`NLProblem` *P*	The problem.
`char *`*type*	The type given to the new nonlinear element.
`NLElementFuntion` *fn*	The element function for the new nonlinear element.
`int *`*vars*	A list of the element variables for the new nonlinear element.

Description

The routine `NLCreateNonlinearElement` allocates and initializes an NLNonlinearElement data structure.

The NLNonlinearElement data structure uses reference counting. The data structure should be deleted using the `NLFreeNonlinearElement` subroutine (page 173). This will decrement the reference count and free the storage if the count goes to zero. References may be added using the `NLRefNonlinearElement` subroutine (page 172).

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL."	12
"Element Function (argument 3) is NULL."	12
"Out of memory, trying to allocate %d bytes"	12

NLRefNonlinearElement

Purpose

Registers a reference to an NLNonlinearElement data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLRefNonlinearElement(P,F);
```

`NLProblem`	P	The problem.
`NLNonlinearElement`	F	The element function.

Description

The NLNonlinearElement data structure uses reference counting. This routine should be used to indicate that a vector is needed by another data structure. The vector will not be deleted until the same data structure indicates that it no longer needed (for example, when the data structure itself is deleted). This works as long as the `NLFreeNonlinearElement` subroutine (page 173) is used to delete the vector, and is only used once per added reference.

Errors

Errors return without chaning the element function.

Message	Severity
”Problem (argument 1) is NULL”	12
”Element Function (argument 2) is invalid”	12

NLFreeNonlinearElement

Purpose

Frees the storage associated with an NLNonlinearElement data structure.

Library

```
libNLPAPI.a
```

C Syntax

```
#include <NLPAPI.h>
void NLFreeNonlinearElement(P,F);
```

`NLProblem`	*P*	The problem.
`NLNonlinearElement`	*F*	The element function.

Description

The NLNonlinearElement data structure uses reference counting. This routine should be used to indicate that a vector is no longer needed. It will decrement the reference count and free the storage if the count goes to zero. The `NLRefNonlinearElement` subroutine (page 172) may be used to add References.

Errors

Errors return without chaning the element function.

Message	Severity
"Problem (argument 1) is NULL"	12
"Nonlinear Element (argument 2) is invalid, "	12

NLNEGetName

Purpose

Returns the name of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
name=`NLNEGetName(`*P*`,`*ne*`);`

`char *`	*name*	The name.
`NLProblem`	*P*	The problem.
`NLNonlinearElement`	*ne*	The nonlinear element.

Description

This routine returns the name of a nonlinear element.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Nonlinear Element (argument 2) is invalid”	12

NLNEGetElementDimension

Purpose

Returns the number of element variables for a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLNEGetElementDimension(P,ne);
```

`int`	*n*	The number of element variables.
`NLProblem`	*P*	The problem.
`NLNonlinearElement`	*ne*	The nonlinear element.

Description

This routine returns the number of element variables for a nonlinear element.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Nonlinear Element (argument 2) is invalid"	12

NLNEGetInternalDimension

Purpose

Returns the number of internal variables for a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLNEGetInternalDimension(P,ne);
```

`int`	*n*	The number of internal variables.
`NLProblem`	*P*	The problem.
`NLNonlinearElement`	*ne*	The nonlinear element.

Description

This routine returns the number of internal variables for a nonlinear element.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Nonlinear Element (argument 2) is invalid”	12

NLNEGetElementFunction

Purpose

Returns the element function of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
f=NLNEGetElementFunction(P,ne);
```

`NLElementFunction`	*f*	The element function.
`NLProblem`	*P*	The problem.
`NLNonlinearElement`	*ne*	The nonlinear element.

Description

This routine returns the element function of a nonlinear element.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Nonlinear Element (argument 2) is invalid”	12

NLNEGetIndex

Purpose

Returns the index of an element variable of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
var=NLNEGetIndex(P,ne,i);
```

`int`	*var*	The variable.
`NLProblem`	*P*	The problem.
`NLNonlinearElement`	*ne*	The nonlinear element.
`int`	*i*	The index of the element variable.

Description

This routine returns the index of an element variable of a nonlinear element.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Nonlinear Element (argument 2) is invalid”	12
”Variable (argument 3) is invalid”	12

NLNEGetRangeXForm

Purpose

Returns the range transformation of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
f=NLNEGetRangeXForm(P,ne);
```

`LNRangeXForm`	*f*	The range transformation (NULL if the identity).
`NLProblem`	*P*	The problem.
`NLNonlinearElement`	*ne*	The nonlinear element.

Description

This routine returns the range transformation of a nonlinear element.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Nonlinear Element (argument 2) is invalid"	12

NLPGetNumberOfNonlinearElements

Purpose

Returns the number of nonlinear elements.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfNonlinearElements(P);
```

`int` n	The number of nonlinear elements.
`NLProblem` P	The problem.

Description

The routine `NLPGetNumberOfNonlinearElements` returns the number of nonlinear elements.

Errors

Message	Severity
”Problem (argument 1) is NULL”	12

NLPGetNumberOfGroups

Purpose

Returns the number of groups in a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfGroups(P);
```

`int`	n	The number of groups.
`NLProblem`	P	The problem.

Description

This routine returns the current number of groups in a problem. Each time a group is added to the objective, or a nonlinear constraint is added this number increases. It never decreases.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12

NLPGetTypeOfGroup

Purpose

Returns the type of a group.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
type=`NLPGetTypeOfGroup(`P`,`i`);`

`char`	$*name$	The type of the group.
`NLProblem`	P	The problem.
`int`	i	The number of the group type.

Description

This routine returns the type of a group. Group types are assigned with the `NLPAddGroupToObjective` (page 43) subroutine, as the last argument. A new group is assigned a number, and the name is stored. The type name "TRIVIAL GROUP" is always defined, and is type number 0.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPGetGroupTypeName

Purpose

Returns the name of a type of group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
name=NLPGetGroupTypeName(P,i);
```

`char`	**name*	The name of the group type.
`NLProblem`	*P*	The problem.
`int`	*i*	The number of the group type.

Description

This routine returns the name of a type of group. Group types are assigned with the `NLPAddGroupToObjective` isubroutine (page 43) and similar routines for the constraints. A new group is assigned a number, and the name is stored. The type name “TRIVIAL GROUP” is always defined, and is type number 0.

Note: The user should not free the returned string.

Errors

Errors return (char*)NULL.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPGetGroupName

Purpose

Returns the name of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
name=NLPGetGroupName(P,i);
```

`char`	**name*	The name of the group.
`NLProblem`	*P*	The problem.
`int`	*i*	The number of the group.

Description

This routine returns the name of a group. Group names are assigned with the `NLPAddGroupToObjective` (page 43) subroutine (and the routines for ading groups to the constraints). Group names need to be unique.

Note: The user should not free the returned string.

Errors

Errors return (char*)NULL.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPSetGroupFunction

Purpose

Sets the group function of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetGroupFunction(P,group,g);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`NLGroupFunction`	*g*	The group function.

Description

This routine sets the group function of a group. This can be queried with the `NLPGetGroupFunction` (page 186) subroutine. The default value is the identity (the trivial group).

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Can't Set Trivial Group's Group Function, group %d",group	12

NLPGetGroupFunction

Purpose

Gets the group function of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
g=NLPGetGroupFunction(P,group);
```

`NLGroupFunction`	*g*	The group function.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.

Description

This routine returns the group function of a group. This can be set with the `NLPSetGroupFunction` (page 185) subroutine. The default value is the identity (the trivial group).

Errors

Errors return (NLGroupFunction)NULL.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPIsGroupFunctionSet

Purpose

Queries whether the group function of a group has been set.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
ans=NLPIsGroupFunctionSet(P,group);
```

`int`	*ans*	The answer, 1==Set, 0=Not Set.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.

Description

This routine queries the group function of a group. If it has it's default value, the trivial group, 0 is returned, otherwise 1.

If an error occurs ans will be -1.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPSetGroupA

Purpose

Sets the linear part of the linear element of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetGroupA(P,group,a);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`NLVector`	*a*	The linear element.

Description

This routine sets the linear part of the lineear element of a group. This can be queried with the `NLPGetGroupA` (page 189) subroutine.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPGetGroupA

Purpose

Gets the linear part of the linear element of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
a=NLPGetGroupA(P,group);
```

`NLVector`	*a*	The linear element.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.

Description

This routine sets the linear part of the lineear element of a group. This can be queried with the `NLPGetGroupA` (page 189) subroutine. The default value is the zero vector.

The vector must have as many entires as the number of variables in the problem. See the `NLCreateVector` (page 95), the `NLCreateVectorWithFullData` (page 98), and the `NLCreateVectorWithSparseData` (page 96) subroutines.

Errors

Errors return (NLVector)NULL.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPIsGroupASet

Purpose

Queries whether the linear part of the linear element of a group has been set.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
ans=NLPIsGroupASet(P,group);
```

`int`	*ans*	The answer, 1==Set, 0=Not Set.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.

Description

This routines queries whether the linear part of the linear element of a group has been set. If it has it's default value, the zero vector, the routine returns 0, otherwise 1.

If an error occurs ans will be -1.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPSetGroupB

Purpose

Sets the constant part of the linear element of a group.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
rc=`NLPSetGroupB(`*P*`,`*group*`,`*b*`);`

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`double`	*b*	The constant.

Description

This routine sets the constant part of the lineear element of a group. The default value is zero.

Note: The definition uses a negative sign for the constant that might be counter intuitive.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPGetGroupB

Purpose

Gets the constant part of the linear element of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
b=NLPGetGroupB(P,group);
```

`NLVector`	*b*	The constant.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.

Description

This routine returns the constant part of the lineear element of a group. This can be queried with the `NLPGetGroupB` (page 192) subroutine.

Errors

Errors return DBL_QNAN.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPIsGroupBSet

Purpose

Queries whether the constant part of the linear element of a group has been set.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
ans=NLPIsGroupBSet(P,group);
```

`int`	*ans*	The answer, 1==Set, 0=Not Set.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.

Description

This routines queries whether the constant part of the linear element of a group has been set. If it has it's default value, zero, the routine returns 0, otherwise 1.

Note: setting the constant part to 0 with `NLPSetGroupB` will not result in an "unset" result.

If an error occurs ans will be -1.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPSetGroupScale

Purpose

Sets the scale factor of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetGroupScale(P,group,s);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`double`	*s*	The scale factor.

Description

This routine sets the scale factor of a group. This can be queried with the `NLPGetGroupScale` (page 195) subroutine.

Note: The definition uses $1/s$ to multiply the group function.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPGetGroupScale

Purpose

Gets the scale factor of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
s=NLPGetGroupScale(P,group);
```

`double`	*s*	The scale factor.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.

Description

This routine returns the scale factor of a group. This can be queried with the `NLPGetGroupScale` (page 195) subroutine.

Note: The definition uses $1/s$ to multiply the group function.

Errors

Errors return DBL_QNAN.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPAddNonlinearElementToGroup

Purpose

Adds an empty nonlinear element to a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
e=NLPAddNonlinearElementToGroup(P,group,type,weight,f,variables,xfrm);
```

`int`	*e*	The index of the new nonlinear element.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`char`	**type*	The type of the new nonlinear element.
`double`	*weight*	The weight.
`NLElementFunction`	*f*	The element function or NULL.
`int`	it variables	A list of the internal variables.
`NLMatrix`	*xfrm*	The range transformation or NULL.

Description

This routine adds a nonlinear element to a group. The group may be either a nonlinear constraint or an objective group.

If the element function passed is `(NLElementFunction)NULL`, it is not set. If the range transformation passed is `(NLMatrix)NULL`, it is not set. If a range transformation is given, it must have as many columns as there are internal variables, and as many rows as the element function has unknowns.

Errors

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12
”Out of memory, trying to allocate %d bytes”	12

NLPGetElementWeight

Purpose

Returns the weight of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
weight=NLPGetElementWeight(P,group,element);
```

`double`	*weight*	The weight.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the nonlinear element.

Description

This routine returns the weight of a nonlinear element in a group. The group may be either a nonlinear constraint or an objective group.

Errors

Errors return DBL_QNAN.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12

NLPSetElementWeight

Purpose

Changes the weight of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetElementWeight(P,group,element,weight);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the nonlinear element.
`double`	*weight*	The weight.

Description

This routine changes the weight of a nonlinear element in a group. The group may be either a nonlinear constraint or an objective group.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12

NLPIsElementWeightSet

Purpose

Queries whether the weight of a nonlinear element has been set.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
ans=NLPIsElementWeightSet(P,group,element);
```

`int`	*ans*	The answer, 1==Set, 0=Not Set.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the element.

Description

This routines queries whether the weight of a nonlinear element of a group has been set. If it has not the routine returns 0, otherwise 1.

If an error occurs ans will be -1.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12

NLPGetElementFunctionOfGroup

Purpose

Returns the nonlinear element function of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
f=NLPGetElementFunction(P,group,element);
```

`NLElementFunction`	*f*	The element function.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the nonlinear element.

Description

This routine returns the nonlinear element function of an element of a group. Note that a global index may also be used, with the `NLPGetElementFunction` (page 202) routine.

Errors

Errors return (NLElementFunction)NULL.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12

NLPGetGroupNonlinearElement

Purpose

Returns a nonlinear element of a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
ne=NLPGetGroupNonlinearElement(P,group,i);
```

`NLNonlinearElement` *ne*	The nonlinear element.
`NLProblem` *P*	The problem.
`int` *group*	The group.
`int` *i*	Which element.

Description

The routine `NLPGetGroupNonlinearElement` returns a nonlinear element of a group.

Errors

Message	Severity
"Problem (argument 1) is NULL"	12
"Group (argument 2) is invalid"	12
"Element (argument 3) is invalid"	12

NLPGetElementFunction

Purpose

Returns the nonlinear element function of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
f=NLPGetElementFunction(P,element);
```

`NLElementFunction`	*f*	The element function.
`NLProblem`	*P*	The problem.
`int`	*element*	The index of the nonlinear element.

Description

This routine returns the nonlinear element function of an element. Note that a global index is used, not the numebr of the element in a group. That method is used by the `NLPGetElementFunctionOfGroup` (page 200) routine.

Errors

Errors return (NLElementFunction)NULL.

Message	Severity
"Problem (argument 1) is NULL"	12
"Element %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPSetElementFunction

Purpose

Changes the nonlinear element function of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
rc=`NLPSetElementFunction(`*P*, *group*, *element*,*f*, *variables*`);`

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the nonlinear element.
`NLElementFunction`	*f*	The element function.
`int`	it variables	A list of the internal variables.

Description

This routine changes the nonlinear element function of an element of a group. There must be as many entries in the list of internal variables as the element function has unknowns.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12
”Element %d is illegal (argument 3). Must be in range 0 to %d”	12
”Out of memory, trying to allocate %d bytes”	12

NLPSetElementFunctionWithRange

Purpose

Changes the nonlinear element function of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=NLPSetElementFunctionWithRange(P,group,element,f,variables,xfrm);
```

`int`	*rc*	The return code.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the nonlinear element.
`NLElementFunction`	*f*	The element function.
`int`	it variables	A list of the internal variables.
`NLMatrix`	*xfrm*	The range transformation.

Description

This routine changes the nonlinear element function of an element of a group. There must be as many entries in the list of internal variables as the element function has unknowns.

The range transformation must have as many columns as there are internal variables, and as many rows as the element function has unknowns.

Errors

Errors return 0 and make no changes to the problem. Normal execution returns 1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12
"Out of memory, trying to allocate %d bytes",n*sizeof(int)	12

NLPIsElementFunctionSet

Purpose

Queries whether the weight of a nonlinear element has been set.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
ans=`NLPIsElementFunctionSet(`*P*`,`*group*`,`*element*`);`

`int`	*ans*	The answer, 1==Set, 0=Not Set.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the element.

Description

This routines queries whether the element function of a nonlinear element of a group has been set. If it has not the routine returns 0, otherwise 1.

If an error occurs ans will be -1.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12

NLPGetElementRangeTransformationOfGroup

Purpose

Returns the range transformation of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
f=`NLPGetElementRangeTransformationOfGroup(`*P*`,`*group*`,`*element*`);`

`NLMatrix`	*f*	The range transformation.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the nonlinear element.

Description

This routine returns the range transformation of an element of a group. Note that a global index may also be used, with the `NLPGetElementRangeTransformation` (page 207) routine.

Errors

Errors return (NLMatrix)NULL.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12

NLPGetElementRangeTransformation

Purpose

Returns the range transformation of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
xfrm=NLPGetElementRangeTransformation(P,element);
```

`NLMatrix`	*xfrm*	The range transformation.
`NLProblem`	*P*	The problem.
`int`	*element*	The index of the nonlinear element.

Description

This routine returns the range transformation of a nonlinear element in a group. Note that a global index is used, not the numebr of the element in a group. That method is used by the `NLPGetElementRangeTransformationOfGroup` (page 206) routine.

Errors

Errors return (NLMatrix)NULL.

Message	Severity
"Problem (argument 1) is NULL"	12
"The global element index %d (argument 2) is illegal, must be in range 0 to %d."	4

NLPGetNumberOfInternalVariablesInElement

Purpose

Returns the number of internal variables of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetElementNumberOfInternalVariablesInElement(P,group,element);
```

`int`	*n*	The number of internal variables.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the nonlinear element.

Description

This routine returns the number of internal variables of a nonlinear element in a group. The group may be either a nonlinear constraint or an objective group.

Note: this is not the number of unknowns of an element function, since the range transformation may be applied before the element function is evaluated.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12

NLPGetElementIndexIntoWhole

Purpose

Returns the number of internal variables of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
var=NLPGetElementIndexIntoWhole(P,group,element,int i);
```

`int`	*var*	The index of the internal variable.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the nonlinear element.
`int`	*i*	Which internal variable.

Description

This routine returns the index of an internal variable of a nonlinear element in a group. The group may be either a nonlinear constraint or an objective group.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12
"Element Internal Variable index %d is illegal (argument 4). Must be in range 0 to %d"	12

NLPGetElementNumberOfUnknowns

Purpose

Returns the number of unknowns of a nonlinear element function.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetElementNumberOfUnknowns(P,group,element);
```

`int`	*n*	The number of unknowns.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the nonlinear element.

Description

This routine returns the number of unknowns of a nonlinear element function in a group. The group may be either a nonlinear constraint or an objective group.

Note: this is not the number of internal variables of an element, since the range transformation may be applied before the element function is evaluated.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12

NLPGetNumberOfElementsInGroup

Purpose

Returns the total number of nonlinear elements in a group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfElementsInGroup(P,group);
```

`int`	*n*	The number of elements.
`NLProblem`	*P*	The problem.
`int`	*group*	Which group.

Description

This routine returns the total number of nonlinear elements in a group.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12

NLPGetNumberOfElements

Purpose

Returns the total number of nonlinear elements for a problem.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfElements(P);
```

`int`	*n*	The number of elements.
`NLProblem`	*P*	The problem.

Description

This routine returns the total number of nonlinear elements for a problem.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12

NLPGetNumberOfElementsO

Purpose

Returns the total number of nonlinear elements in the Objective.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfElementsiO(P);
```

`int`	n	The number of elements.
`NLProblem`	P	The problem.

Description

This routine returns the total number of nonlinear elements in the Objective.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12

NLPGetNumberOfElementsE

Purpose

Returns the total number of nonlinear elements in the equality constraints.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfElementsE(P);
```

`int`	*n*	The number of elements.
`NLProblem`	*P*	The problem.

Description

This routine returns the total number of nonlinear elements in the equality constraints.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12

NLPGetNumberOfElementsI

Purpose

Returns the total number of nonlinear elements in the inequality constraints.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfElementsI(P);
```

`int`	n	The number of elements.
`NLProblem`	P	The problem.

Description

This routine returns the total number of nonlinear elements in the inequality constraints.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12

NLPGetElementTypeName

Purpose

Returns the type name of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
name=`NLPGetElementTypeName(`*P*`,`*group*`,`*element*`);`

`char`	**name*	The type name of the element.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the element.

Description

This routine returns the type name of a nonlinear element. Element types are assigned with the `NLCreateNonlinearElement` (page 171) subroutine. A new type is assigned a number, and the name is stored.

Errors

Errors return (char*)NULL.

Message	Severity
"Problem (argument 1) is NULL"	12
"Group %d is illegal (argument 2). Must be in range 0 to %d"	12
"Element %d is illegal (argument 3). Must be in range 0 to %d"	12

NLPGetTypeOfElement

Purpose

Returns the type name of a nonlinear element.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
type=`LNPTypeOfElement(`*P*`,`*group*`,`*element*`);`

`int`	*type*	The type of the element.
`NLProblem`	*P*	The problem.
`int`	*group*	The index of the group.
`int`	*element*	The number of the element.

Description

This routine returns the type of a nonlinear element. Element types are assigned with the `NLCreateNonlinearElement` (page 171) subroutine. A new type name is assigned a number, and the name is stored.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Group %d is illegal (argument 2). Must be in range 0 to %d”	12
”Element %d is illegal (argument 3). Must be in range 0 to %d”	12

NLPGetNumberOfElementTypes

Purpose

Returns the number of distinct types of nonlinear elements.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfElementTypes(P);
```

`int`	n	The number of element types.
`NLProblem`	P	The problem.

Description

This routine returns the number of distinct element types. Element types are assigned with the `NLCreateNonlinearElement` (page 171) subroutine. A new type name is assigned a number, and the name is stored.

Errors

Errors return -1.

Message	Severity
"Problem (argument 1) is NULL"	12

NLPGetElementType

Purpose

Returns the index of a type of nonlinear element.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
type=NLPGetElementType(P,i);
```

`int`	*type*	The index of the element type.
`NLProblem`	*P*	The problem.
`int`	*i*	Which type.

Description

This routine returns the index of an element type. Element types are assigned with the `NLCreateNonlinearElement` (page 171) subroutine. A new type name is assigned a number, and the name is stored.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Element type %d is illegal (argument 2). Must be in range 0 to %d”	12

NLPGetNumberOfGroupTypes

Purpose

Returns the number of distinct types of groups.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
n=NLPGetNumberOfGroupTypes(P);
```

`int`	n	The number of group types.
`NLProblem`	P	The problem.

Description

This routine returns the number of distinct group types. Group types are assigned with the `NLPAddGroupToObjective` (page 43) subroutine. A new type name is assigned a number, and the name is stored.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12

NLPGetGroupType

Purpose

Returns the index of a type of group.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
type=NLPGetGroupType(P,i);
```

`int`	*type*	The type.
`NLProblem`	*P*	The problem.
`int`	*i*	The index of the type.

Description

This routine returns the index of a group type. Group types are assigned with the `NLPAddGroupToObjective` (page 43) subroutine. A new type name is assigned a number, and the name is stored.

Errors

Errors return -1.

Message	Severity
”Problem (argument 1) is NULL”	12
”Type %d is illegal (argument 2). Must be in range 0 to %d”	12

NLCreateLancelot

Purpose

Allocates and initializes an NLLancelot data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
Lan=NLCreateLancelot();
```

`NLLancelot`	*Lan*	The solver.

Description

The routine `NLCreateLancelot` allocates and initializes an NLLancelot data structure. The solver returned has default parameters values, which can be set with various subroutines. Multiple instances are legal.

The storage used by the solver can be returned to the system using the `NLFreeLancelot` subroutine (page 224).

Errors

Errors return (NLLancelot)NULL.

Message	Severity
"Out of memory, trying to allocate %d bytes"	12

NLRefLancelot

Purpose

Registers a reference to an NLLancelot data structure.

Library

```
libNLPAPI.a
```

C Syntax

```
#include <NLPAPI.h>
void NLRefLancelot(lan);
```

`NLLancelot`	*lan*	The LANCELOT solver.

Description

The NLLancelot data structure uses reference counting. This routine should be used to indicate that a vector is needed by another data structure. The vector will not be deleted until the same data structure indicates that it no longer needed (for example, when the data structure itself is deleted). This works as long as the `NLFreeLancelot` subroutine (page 224) is used to delete the vector, and is only used once per added reference.

Errors

Severity 4 errors return without changing the vector.

Message	Severity
”Pointer to Lancelot (argument 1) is NULL”	4

NLFreeLancelot

Purpose

Releases storage associated with an NLLancelot data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void NLFreeLancelot(Lan);
```

`NLLancelot` *Lan* The solver.

Description

The routine `NLFreeLancelot` returns storage associated with a solver to the system.

Errors

Errors return without changing the solver.

Message	Severity
”Solver (argument 1) is NULL”	12

LNMinimize

Purpose

Allocates and initializes an NLLancelot data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
c=LNMinimize(Lan,P,x0,z0,l0,x);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`NLProblem`	*P*	The problem to be solved.
`double`	**x0*	The initial guess.
`double`	**z0*	The initial values of the min-max variables (currently only one value), or (double*)NULL.
`double`	**l0*	The initial guess at the multipliers, or (double*)NULL.
`double`	**x*	The solution.

Description

The routine `LNMinimize` invokes Lancelot to find a minimizer of the objective function of a problem. If there are min-max constraints the vector containing the inital values for the minmax variables should be either NULL (no value), or one. The vector containing the initial Lagrange multipliers should either be NULL (no values given), or length as long as the total number of constraints, including min-max constraints).

Note: the user is responsible for allocating sufficient space for the solution.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12
"Pointer to initial guess (argument 2) is NULL"	12
"Pointer to area to store the solution (argument 4) is NULL"	12
"Error opening file %s for writing in current directory"	12
"Error closing file %s"	4
"Error opening file SOLUTION.d for reading in current directory"	12

LNMaximize and LNMaximizeDLL

Purpose

Allocates and initializes an NLLancelot data structure.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
c=LNMaximize(Lan,P,x0,z0,l0,x);
c=LNMaximizeDLL(Lan,P,x0,z0,l0,x);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`NLProblem`	*P*	The problem to be solved.
`double`	**x0*	The initial guess.
`double`	**z0*	The initial value of the min-max variabls, or (double*)NULL.
`double`	**l0*	The initial guess at the multipliers, or (double*)NULL.
`double`	**x*	The solution.

Description

The routine `LNMinimize` invokes Lancelot to find a maximizer of the objective function of a problem. If there are min-max constraints the vector containing the inital values for the minmax variables should be either NULL (no value), or one. The vector containing the initial Lagrange multipliers should either be NULL (no values given), or length as long as the total number of constraints, including min-max constraints).

Note: the user is responsible for allocating sufficient space for the solution.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12
"Pointer to initial guess (argument 2) is NULL"	12
"Pointer to area to store the solution (argument 4) is NULL"	12
"Error opening file %s for writing in current directory"	12
"Error closing file %s"	4
"Error opening file SOLUTION.d for reading in current directory"	12

LNSetCheckDerivatives

Purpose

Sets the parameter controlling how Lancelot testtests derivatives.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetCheckDerivatives(Lan,flag);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*flag*	How to test.

Description

The routine `LNSetCheckDerivatives` sets the parameter controlling how Lancelot tests derivatives. Legal values for the flag and their meaning –

0	No checking
1	Check all derivatives
2	Check derivatives
3	Check element derivatives
4	Check group derivatives

The default value is 0.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12
”Option %d (argument 2) is invalid, must be in range 0-4”	12

LNGetCheckDerivatives

Purpose

Gets the parameter controlling how Lancelot test derivatives.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
flag=LNGetCheckDerivatives(Lan);
```

`int`	*flag*	How to test.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetCheckDerivatives` gets the parameter controlling how Lancelot test derivatives. Legal values for the flag and their meaning –

0	No checking
1	Check all derivatives
2	Check derivatives
3	Check element derivatives
4	Check group derivatives

The default value is 0.

Errors

Errors return -1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetConstraintAccuracy

Purpose

Sets the parameter controlling how accurately constraints are solved.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
rc=`LNSetConstraintAccuracy(`*Lan*`,`*limit*`);`

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`double`	*limit*	The accuracy.

Description

The routine `LNSetConstraintAccuracy` sets the parameter controlling how accurately the constraints are solved. The default value is 0.00001. The `SPEC.SPC` file entry this corresponds to is `CONSTRAINT-ACCURACY-REQUIRED`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNGetConstraintAccuracy

Purpose

Gets the parameter controlling how accurately Lancelot solves constraints.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
limit=LNGetConstraintAccuracy(Lan);
```

`double`	*limit*	The accuracy.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetConstraintAccuracy` gets the parameter controlling how accurately Lancelot solves the constraints. The default value is 0.00001. The `SPEC.SPC` file entry this gets is `CONSTRAINT-ACCURACY-REQUIRED`.

Errors

Errors return DBL_QNAN.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetFirstConstraintAccuracy

Purpose

Sets the parameter controlling the initial accuracy Lancelot uses for the constraints.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetFirstConstraintAccuracy(Lan,acc);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`double`	*acc*	The accuracy.

Description

The routine `LNSetFirstConstraintAccuracy` sets the parameter controlling the initial accuracy Lancelot uses for the constraints. The default value is 0.1. The `SPEC.SPC` file entry this sets is `FIRST-CONSTRAINT-ACCURACY-REQUIRED`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNGetFirstConstraintAccuracy

Purpose

Gets the parameter controlling the initial accuracy Lancelot uses for the constraints.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
acc=LNGetFirstConstraintAccuracy(Lan);
```

`double`	*acc*	The accuracy.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetFirstConstraintAccuracy` gets the parameter controlling the initial accuracy Lancelot uses for the constraints. The default value is 0.1. The `SPEC.SPC` file entry this gets is `FIRST-CONSTRAINT-ACCURACY-REQUIRED`.

Errors

Errors return DBL_QNAN.

Message	Severity
”Solver (argument 1) is NULL”	12

LNSetFirstGradientAccuracy

Purpose

Sets the parameter controlling the initial accuracy for the gradients.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetFirstGradientAccuracy(Lan,limit);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`double`	*limit*	The accuracy.

Description

The routine `LNSetFirstGradientAccuracy` sets the parameter controlling the initial accuracy for the gradients. The default value is 0.1. The `SPEC.SPC` file entry this sets is `FIRST-GRADIENT-ACCURACY-REQUIRED`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNGetFirstGradientAccuracy

Purpose

Gets the parameter controlling the initial accuracy for the gradients.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
acc=LNGetFirstGradientAccuracy(Lan);
```

`double`	*acc*	The accuracy.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetFirstGradientAccuracy` gets the parameter controlling the initial accuracy for the gradients. The default value is 0.1. The `SPEC.SPC` file entry this corresponds to is `FIRST-GRADIENT-ACCURACY-REQUIRED`.

Errors

Errors return DBL_QNAN.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetGradientAccuracy

Purpose

Sets the parameter controlling the accuracy for the gradients.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetGradientAccuracy(Lan,limit);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`double`	*limit*	The accuracy.

Description

The routine `LNSetGradientAccuracy` sets the parameter controlling the accuracy for the gradients. The default value is 0.00001. The `SPEC.SPC` file entry this corresponds to is `GRADIENT-ACCURACY-REQUIRED`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNGetGradientAccuracy

Purpose

Gets the parameter controlling the accuracy for the gradients.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
limit=LNGetGradientAccuracy(Lan);
```

`double`	*limit*	The accuracy.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetGradientAccuracy` gets the parameter controlling the accuracy for the gradients. The default value is 0.00001. The `SPEC.SPC` file entry this corresponds to is `GRADIENT-ACCURACY-REQUIRED`.

Errors

Errors return DBL_QNAN.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetInitialPenalty

Purpose

Sets the parameter controlling the initial penalty.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetInitialPenalty(Lan,penalty);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`double`	*penalty*	The penalty.

Description

The routine `LNSetInitialPenalty` sets the parameter controlling the initial penalty. The default value is 0.1. The `SPEC.SPC` file entry this sets is `INITIAL-PENALTY-PARAMETER`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNGetInitialPenalty

Purpose

Gets the parameter controlling the initial penalty.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
penalty=`LNGetInitialPenalty(`*Lan*`);`

`double`	*penalty*	The penalty.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetInitialPenalty` gets the parameter controlling the initial penalty. The default value is 0. The `SPEC.SPC` file entry this sets is `INITIAL-PENALTY-PARAMETER`.

Errors

Errors return DBL_QNAN.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetMaximumNumberOfIterations

Purpose

Sets the parameter controlling how long Lancelot runs.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
rc=`LNSetMaximumNumberOfIterations(`*Lan*`,`*iter*`);`

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*iter*	Maximum number of iterations.

Description

The routine `LNSetMaximumNumberOfIterations` sets the parameter controlling how long Lancelot runs. The default value is 100.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNGetMaximumNumberOfIterations

Purpose

Gets the parameter controlling how long Lancelot runs.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
iter=LNGetMaximumNumberOfIterations(Lan);
```

`int`	*iter*	Maximum number of iterations.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetMaximumNumberOfIterations` sets the parameter controlling how long Lancelot runs. The default value is 100.

Errors

Errors return -1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNSetPenaltyBound

Purpose

Sets the parameter controlling the bound on the penalty Lancelot uses.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetPenaltyBound(Lan,penalty);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`double`	*penalty*	The bound.

Description

The routine `LNSetPenaltyBound` sets the parameter controlling the bound on the penalty Lancelot uses. The default value is 0.1. The `SPEC.SPC` file entry this sets is
`DECREASE-PENALTY-PARAMETER-UNTIL`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNGetPenaltyBound

Purpose

Gets the parameter controlling the bound on the penalty Lancelot uses.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void penalty=LNGetPenaltyBound(Lan);
```

`double`	*penalty*	The bound.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetPenaltyBound` gets the parameter controlling the bound on the penalty Lancelot uses. The default value is 0.1. The `SPEC.SPC` file entry this sets is
`DECREASE-PENALTY-PARAMETER-UNTIL`.

Errors

Errors return DBL_QNAN.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetPrintEvery

Purpose

Sets the parameter controlling how often Lancelot prints.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetPrintEvery(Lan,iter);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*iter*	

Description

The routine `LNSetPrintEvery` sets the parameter controlling how often Lancelot prints. The default value is 1.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNGetPrintEvery

Purpose

Gets the parameter controlling how often Lancelot prints.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
iter=LNGetPrintEvery(Lan);
```

`int`	*iter*	
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetPrintEvery` sets the parameter controlling how often Lancelot prints. The default value is 1.

Errors

Errors return -1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetPrintLevel

Purpose

Sets the parameter controlling how much output Lancelot produces.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetPrintLevel(Lan,level);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*level*	

Description

The routine `LNSetPrintLevel` Sets the parameter controlling how Lancelot how much output Lancelot produces. The default value is 0.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12
"PrintLevel %d (argument 2) is invalid, must be nonnegative"	12

LNGetPrintLevel

Purpose

Gets the parameter controlling how much output Lancelot produces.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
level=LNGetPrintLevel(Lan);
```

`int`	*level*	
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetPrintLevel` Gets the parameter controlling how Lancelot how much output Lancelot produces. The default value is 0.

Errors

Errors return -1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetPrintStart

Purpose

Sets the parameter controlling when Lancelot starts printing.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetPrintStart(Lan,iter);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*iter*	

Description

The routine `LNSetPrintStart` sets the parameter controlling when Lancelot starts printing. The default value is 0.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNGetPrintStart

Purpose

Gets the parameter controlling when Lancelot starts printing.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
iter=LNGetPrintStart(Lan);
```

`int`	*iter*	
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetPrintStart` sets the parameter controlling when Lancelot starts printing. The default value is 0.

Errors

Errors return -1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetPrintStop

Purpose

Sets the parameter controlling when Lancelot stops printing.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetPrintStop(Lan,iter);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*iter*	

Description

The routine `LNSetPrintStop` sets the parameter controlling when Lancelot stops printing. The default value is 100.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNGetPrintStop

Purpose

Gets the parameter controlling when Lancelot stops printing.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
iter=LNGetPrintStop(Lan);
```

`int`	*iter*	
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetPrintStop` sets the parameter controlling when Lancelot stops printing. The default value is 100.

Errors

Errors return -1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNSetRequireExactCauchyPoint

Purpose

Sets the parameter determining whether an exact cauchy point is required.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetRequireExactCauchyPoint(Lan,choice);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*choice*	

Description

The routine `LNSetRequireExactCauchyPoint` sets the parameter determining whether an exact cauchy point is required. The default is 1.

The corresponding `SPEC.SPC` file entries are `EXACT-CAUCHY-POINT-REQUIRED`

and `INEXACT-CAUCHY-POINT-REQUIRED`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNGetRequireExactCauchyPoint

Purpose

Gets the parameter determining whether an exact cauchy point is required.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
choice`=LNGetRequireExactCauchyPoint(`*Lan*`);`

`int`	*choice*	
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetRequireExactCauchyPoint` gets the parameter determining whether an exact cauchy point is required. The default is 1.

The corresponding `SPEC.SPC` file entries are `EXACT-CAUCHY-POINT-REQUIRED`

and `INEXACT-CAUCHY-POINT-REQUIRED`.

Errors

Errors return -1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNSetSaveDataEvery

Purpose

Sets the parameter controlling how often Lancelot saves data.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetSaveDataEvery(Lan,iter);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*iter*	

Description

The routine `LNSetSaveDataEvery` sets the parameter controlling how often Lancelot saves data. The default value is 0 (don't save).

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNGetSaveDataEvery

Purpose

Gets the parameter controlling how often Lancelot saves data.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
iter=LNGetSaveDataEvery(Lan);
```

`int`	*iter*	
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetSaveDataEvery` sets the parameter controlling how often Lancelot saves data. The default value is 0 (don't save).

Errors

Errors return -1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetScalings

Purpose

Sets the parameter controlling how Lancelot uses scalings.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetScalings(Lan,choice);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`char`	**choice*	How to use scalings.

Description

The routine `LNSetScalings` sets the parameter controlling how Lancelot uses scalings.

The legal values for *choice* and the corresponding `SPEC.SPC` file entries are

"no scaling"	No entry in SPEC.SPC
"scale constraints"	USE-CONSTRAINT-SCALING-FACTORS
"scale variables"	USE-VARIABLE-SCALING-FACTORS
"scale both"	USE-SCALING-FACTORS
"print but don't use"	PRINT-SCALING-FACTORS

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12
"Out of memory, trying to allocate %d bytes"	12
"Scaling "%s" (argument 2) is invalid, must be one of "No Scaling","Scale Constraints","Scale Variables","Scale Both","Print but don't use""	12

LNGetScalings

Purpose

Gets the parameter controlling how Lancelot uses scalings.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
choice=`LNGetScalings(`*Lan*`);`

`char`	**choice*	How to use scalings.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetScalings` gets the parameter controlling how Lancelot uses scalings.

The legal values for *choice* and the corresponding `SPEC.SPC` file entries are

"no scaling"	USE-SCALING-FACTORS
"scale constraints"	USE-CONSTRAINT-SCALING-FACTORS
"scale variables"	USE-VARIABLE-SCALING-FACTORS
"scale both"	USE-SCALING-FACTORS
"print but don't use"	PRINT-SCALING-FACTORS

Errors

Errors return (char*)NULL.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetSolveBQPAccurately

Purpose

Sets the parameter controlling the solution of the BQP.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetSolveBQPAccurately(Lan,choice);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*choice*	

Description

The routine `LNSetSolveBQPAccurately` sets the parameter controlling the solution of the BQP. The default is 0.

The corresponding `SPEC.SPC` file entry is `SOLVE-BQP-ACCURATELY`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNGetSolveBQPAccurately

Purpose

Gets the parameter controlling the solution of the BQP.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
choice=LNGetSolveBQPAccurately(Lan);
```

`int`	*choice*	
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetSolveBQPAccurately` gets the parameter controlling the solution of the BQP. The default is 0.

The corresponding `SPEC.SPC` file entry is `SOLVE-BQP-ACCURATELY`.

Errors

Errors return -1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetLinearSolverMethod

Purpose

Sets the parameter determining what linear solver is used.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
rc=`LNSetLinearSolverMethod(`*Lan*`,`*choice*`,`*bandwidth*`);`

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`char`	**choice*	
`int`	*bandwidth*	Bandwidth, if choice is ”bandsolver preconditioned”

Description

The routine `LNSetLinearSolverMethod` sets the parameter determining what linear solver is used.

Legal values of *choice*, and the corresponding `SPEC.SPC` file entries are

”Diagonal preconditioned”
`DIAGONAL-PRECONDITIONED-CG-SOLVER-USED`

”Munksgaards preconditioned”
`MUNKSGAARDS-PRECONDITIONED-CG-SOLVER-USED`

”Expanding band preconditioned”
`EXPANDING-BAND-PRECONDITIONED-CG-SOLVER-USED`

”Full matrix preconditioned”
`FULL-MATRIX-PRECONDITIONED-CG-SOLVER-USED`

”Gill-Murray-Ponceleon-Saunders preconditioned”
`GILL-MURRAY-PONCELEON-SAUNDERS-PRECONDITIONED-CG-SOLVER-USED`

”Modified MA27 preconditioned”
`MODIFIED-MA27-PRECONDITIONED-CG-SOLVER-USED`

”Schnabel-Eskow preconditioned”
`SCHNABEL-ESKOW-PRECONDITIONED-CG-SOLVER-USED`

”Users preconditioned”
`USERS-PRECONDITIONED-CG-SOLVER-USED`

”Bandsolver preconditioned”
`BANDSOLVER-PRECONDITIONED-CG-SOLVER-USED`

”Multifront”
`MULTIFRONT-SOLVER-USED`

”Direct modified”
`DIRECT-MODIFIED-MULTIFRONTAL-SOLVER-USED`

”CG method used”
`CG-METHOD-USED`

Errors
Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12
”Out of memory, trying to allocate %d bytes”	12
”Linear Solver Type ”%s” (argument 2) is invalid”	12

LNGetLinearSolverMethod

Purpose

Gets the parameter determining what linear solver is used.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
int choice=LNGetLinearSolverMethod(Lan);
```

`int`	*choice*	
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetLinearSolverMethod` gets the parameter determining what linear solver is used. If a banded solver is used, the bandwidth can be retrieved with the `LNGetLinearSolverBandwidth` routine (page 265).

Legal values of *choice*, and the corresponding `SPEC.SPC` file entries are

”Diagonal preconditioned”
`DIAGONAL-PRECONDITIONED-CG-SOLVER-USED`

”Munksgaards preconditioned”
`MUNKSGAARDS-PRECONDITIONED-CG-SOLVER-USED`

”Expanding band preconditioned”
`EXPANDING-BAND-PRECONDITIONED-CG-SOLVER-USED`

”Full matrix preconditioned”
`FULL-MATRIX-PRECONDITIONED-CG-SOLVER-USED`

”Gill-Murray-Ponceleon-Saunders preconditioned”
`GILL-MURRAY-PONCELEON-SAUNDERS-PRECONDITIONED-CG-SOLVER-USED`

”Modified MA27 preconditioned”
`MODIFIED-MA27-PRECONDITIONED-CG-SOLVER-USED`

"Schnabel-Eskow preconditioned"
`SCHNABEL-ESKOW-PRECONDITIONED-CG-SOLVER-USED`

"Users preconditioned"
`USERS-PRECONDITIONED-CG-SOLVER-USED`

"Bandsolver preconditioned"
`BANDSOLVER-PRECONDITIONED-CG-SOLVER-USED`

"Multifront"
`MULTIFRONT-SOLVER-USED`

"Direct modified"
`DIRECT-MODIFIED-MULTIFRONTAL-SOLVER-USED`

"CG method used"
`CG-METHOD-USED`

Errors
Errors return -1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNGetLinearSolverBandwidth

Purpose

Gets the parameter determining what bandwidth the linear solver uses.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
bandwidth=LNGetLinearSolverBandwidth(Lan);
```

`int`	*bandwidth*	The bandwidth.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetLinearSolverBandwidth` gets the parameter determining what bandwidth the linear solver uses.

Errors

Errors return -1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNSetStopOnBadDerivatives

Purpose

Sets the parameter controlling how Lancelot deals with bad derivatives.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetStopOnBadDerivatives(Lan,flag);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*flag*	What to do.

Description

The routine `LNSetStopOnBadDerivatives` Sets the parameter controlling how Lancelot deals with bad derivatives. Legal values for the flag and their meaning –

0	stop on warning
1	stop on element derivative warning
2	stop on group derivative warning

The default value is 0.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12
"Option %d (argument 2) is invalid, must be in range 0-2"	12

LNGetStopOnBadDerivatives

Purpose

Gets the parameter controlling how Lancelot deals with bad derivatives.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
void LNGetStopOnBadDerivatives(Lan,flag);
```

`NLLancelot`	*Lan*	The solver.
`int`	*flag*	What to do.

Description

The routine `LNGetStopOnBadDerivatives` Gets the parameter controlling how Lancelot deals with bad derivatives. Legal values for the flag and their meaning –

0	stop on warning
1	stop on element derivative warning
2	stop on group derivative warning

The default value is 0.

Errors

Errors return -1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetTrustRegionRadius

Purpose

Sets the parameter controlling the radius of the trust region.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetTrustRegionRadius(Lan,radius);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`double`	*radius*	The radius.

Description

The routine `LNSetTrustRegionRadius` sets the parameter controlling the radius of the trust region. The default value is 0. The `SPEC.SPC` file entry this sets is `TRUST-REGION-RADIUS`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNGetTrustRegionRadius

Purpose

Gets the parameter controlling the radius of the trust region.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
radius=LNGetTrustRegionRadius(Lan);
```

`double`	*radius*	The radius.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetTrustRegionRadius` gets the parameter controlling the radius of the trust region. The default value is 0. The `SPEC.SPC` file entry this gets is `TRUST-REGION-RADIUS`.

Errors

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetTrustRegionType

Purpose

Sets the parameter controlling the type of trust region Lancelot uses.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetTrustRegionType(Lan, choice);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`char`	**choice*	Which type.

Description

The routine `LNSetTrustRegionType` sets the parameter controlling the type of trust region Lancelot uses. Legal values for the *choice* and their meaning –

Message	Severity
”two norm”	TWO-NORM-TRUST-REGION-USED
”infinity norm”	INFINITY-NORM-TRUST-REGION-USED

The default value is ”infinity norm”.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12
”Out of memory, trying to allocate %d bytes”	12
”TrustRegionType ”%s” (argument 2) is invalid, must be one of ”Two Norm” ”Infinity Norm”,choice	12

LNGetTrustRegionType

Purpose

Gets the parameter controlling the type of trust region Lancelot uses.

Library

`libNLPAPI.a`

C Syntax

`#include <NLPAPI.h>`
choice=`LNGetTrustRegionType(`*Lan*`);`

`char`	$*$*choice*	Which type.
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetTrustRegionType` gets the parameter controlling the type of trust region Lancelot uses. Legal values for the *choice* and their meaning –

”two norm”	TWO-NORM-TRUST-REGION-USED
”infinity norm”	INFINITY-NORM-TRUST-REGION-USED

The default value is ”infinity norm”.

Errors

Errors return (char*)NULL.

Message	Severity
”Solver (argument 1) is NULL”	12

LNSetUseExactFirstDerivatives

Purpose

Sets the parameter controlling how Lancelot gets derivatives.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetUseExactFirstDerivatives(Lan,flag);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`int`	*flag*	What to do

Description

The routine `LNSetUseExactFirstDerivatives` sets the parameter controlling how Lancelot gets derivatives. If flag is 0, differencing is used, otherwise exact derivatives are expected. The default value is 1 (exact derivatives). The `SPEC.SPC` entry this corresponds to is `FINITE-DIFFERENCE-GRADIENTS`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
”Solver (argument 1) is NULL”	12

LNGetUseExactFirstDerivatives

Purpose

Gets the parameter controlling how Lancelot gets derivatives.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
flag=LNGetUseExactFirstDerivatives(Lan);
```

`int`	*flag*	What to do
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetUseExactFirstDerivatives` gets the parameter controlling how Lancelot gets derivatives. If flag is 0, differencing is used, otherwise exact derivatives are expected. The default value is 1 (exact derivatives). The `SPEC.SPC` entry this gets is the logical negative of `FINITE-DIFFERENCE-GRADIENTS`.

Errors

Errors return -1.

Message	Severity
"Solver (argument 1) is NULL"	12

LNSetUseExactSecondDerivatives

Purpose

Sets the parameter controlling second derivatives.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
rc=LNSetUseExactSecondDerivatives(Lan,flag);
```

`int`	*rc*	The return code.
`NLLancelot`	*Lan*	The solver.
`char`	**flag*	What to do - one of "Exact", "BFGS","DFP","PSB", or "SR1"

Description

The routine `LNSetUseExactSecondDerivatives` sets the parameter controlling how second derivatives are handled. If flag is 0, differencing is used, otherwise exact second derivatives are expected. The default value is 1 (exact second derivatives). The `SPEC.SPC` entry this corresponds to is `EXACT-SECOND-DERIVATIVES-USED`.

Errors

Errors return 0, normal execution returns 1.

Message	Severity
"Solver (argument 1) is NULL"	12
"Out of memory, trying to allocate %d bytes"	12
"Option "%s" is invalid, must be one of "Exact", "BFGS","DFP","PSB", "SR1","	12

LNGetUseExactSecondDerivatives

Purpose

Gets the parameter controlling how Lancelot gets second derivatives.

Library

`libNLPAPI.a`

C Syntax

```
#include <NLPAPI.h>
flag=LNGetUseExactSecondDerivatives(Lan);
```

`int`	*flag*	What to do
`NLLancelot`	*Lan*	The solver.

Description

The routine `LNGetUseExactSecondDerivatives` gets the parameter controlling gets second derivatives. If flag is 0, differencing is used, otherwise exact second derivatives are expected. The default value is 1 (exact second derivatives). The `SPEC.SPC` entry this gets is `EXACT-SECOND-DERIVATIVES-USED`.

Errors

Errors return -1.

Message	Severity
”Solver (argument 1) is NULL”	12