Reference manual

FICO

FICO® Xpress Optimization

MIP Solution Pool

Reference manual

Release 31.01

Last update 04 April 2017

Xpress-Optimizer

Deliverable Version: A

Last Revised: 04 April 2017

Version 31.01

Contents

I. MIP Solution Pool

CHAPTER 1

Introduction

1.1 Overview

The MIP solution pool (`XPRSmipsolpool`) stores the column solution values for multiple solutions. No slack or dual information is stored with the solutions. The solutions will usually be for MIP problems although the pool will store any arbitrary length vector of double values. Solutions can be read into the MIP solution pool by the user via the file reader routine `XPRS_msp_readslxsol` or from memory via the `XPRS_msp_loadsol` routine. Solutions can also be captured automatically from problems (`XPRSprob`) as they are found during a search.

Since the `XPRSprob` problems only store the best MIP solution the MIP solution pool is useful when the user is interested in more than one MIP solution for a problem. This can happen when there are constraints or costs not reflected in the problem that the user wants to use to select a solution from a set that have been found by the FICO Xpress Optimizer.

Once a MIP solution pool contains some solutions the user can query for the solution values and also for more abstract attributes such as the objective value of the solution with respect to a given problem. Apart from querying information for a single solution the user has various options for managing the set of stored solutions. The user can query for lists of stored solutions. They may delete solutions from the pool and output solutions to file. Also, a set of policies are available to the user for automatically handling the exclusion of solutions that are duplicated.

The following sections discuss the MIP solution pool functionality and provide some hints on usage.

1.2 Data Model

The UML diagram in Figure 1.1 outlines the relationships the `XPRSmipsolpool` has with some other data entities.

Starting from the bottom right the diagram shows that the `XPRSmipsolpool` contains 0 or more solutions. Solutions are loaded into the MIP solution pool as a dense vector of double values. Solution vectors of varying length can be stored in the same MIP solution pool so each solution stores the number of columns `nCols` for the solution. When solutions are loaded they are stored with a name and an ID number. The names of the stored solutions are maintained so they are unique among the current set of stored solutions. The IDs are generated using a counter starting from 1 and are therefore unique over all solutions ever to be stored in the MIP solution pool. Note that no mapping of column names to column index values is stored with the solutions. Therefore, the solutions are stored as plain vectors of implicitly indexed values. Note that although the array of solution values `Vals` is represented in the Solution object in the diagram as a dense array of doubles the solutions are stored in a compact form.

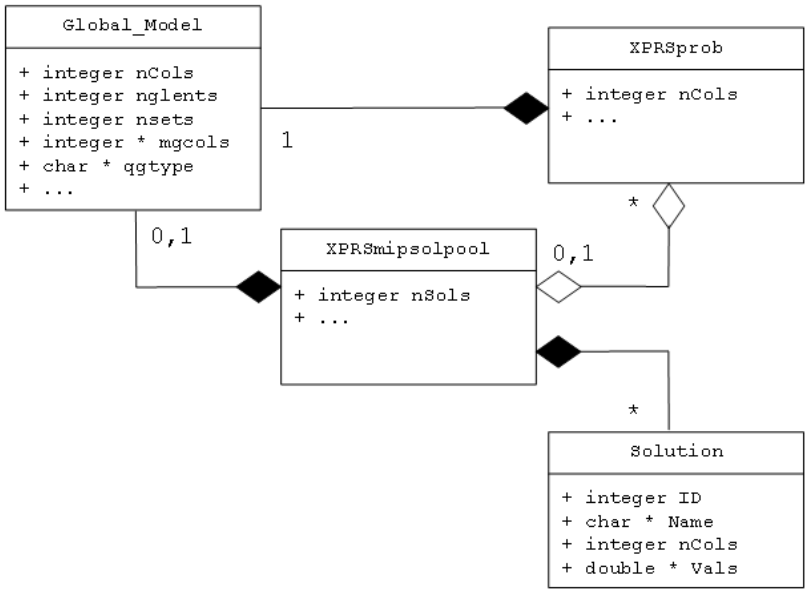

Figure 1.1:

Moving to the top right the diagram shows that the `XPRSmipsolpool` references 0 or more `XPRSprob` problems and that an `XPRSprob` references at most 1 `XPRSmipsolpool`. This relationship is managed through the "attaching" of problems to the MIP solution pool, which we discuss in section "Attaching problems". Note that by attaching problems it is possible, for example, to automatically capture MIP solutions from the problems as they are found by the solver.

Finally, moving to the top left the diagram shows the object defining the global entities for a problem. Clearly the `XPRSprob` has one of these (even if it is empty in the case of an LP problem). The `XPRSmipsolpool` may capture one of these from an attached problem and uses it only to detect duplicate solutions. We discuss duplicate solutions in section "Duplicate solutions".

Once the MIP solution pool contains some solutions the user can administer the stored solutions using the following functionality:-

- Solution names can be changed using the routine `XPRS_msp_setsolname`.
- Solutions can be deleted from the MIP solution pool by calling `XPRS_msp_delsol` with the ID of the solution to delete.
- A mapping between solution name and ID is provided with the routines `XPRS_msp_findsolbyname` and `XPRS_msp_getsolname`.
- Attributes of the solutions can be accessed using routines `XPRS_msp_getsol`, `XPRS_msp_getdblattribprobsol`, `XPRS_msp_getintattribprobsol`, `XPRS_msp_getdblattribsol`, `XPRS_msp_getintattribsol`, `XPRS_msp_getdblattribprob`, `XPRS_msp_getintattribprob`, `XPRS_msp_getdblattribprobextreme` and `XPRS_msp_getintattribprobextreme`.
- Lists of solution IDs can be generated based on attributes of the solutions with respect to a given problem using the routine `XPRS_msp_getsollist`.

1.3 Solution input

Solutions can be input into the MIP solution pool from file, from memory and they can be

captured automatically by the MIP solution pool directly from XPRSprob problems. The following two sections discuss this functionality.

1.3.1 Attaching problems

Of the more useful features of the MIP solution pool type has the ability to 'attach' XPRSprob problems to a MIP solution pool with a call to XPRS_msp_probattach. Once a problem is attached to a MIP solution pool they interact automatically with useful functions. For example, when an attached problem finds a solution it automatically stores the solution into the MIP solution pool.

As illustrated in the UML data model diagram a MIP solution pool can have one or more attached problems although a problem can only be attached to at most one MIP solution pool. The attached problems do not need to be copies of each other and they do not need to have the same number of columns.

The MIP solution pool ensures each solution is automatically loaded to all attached problems with the same number of columns as the solution. Solutions are loaded to a problem when the solver is running the problem with XPRSminim (p.285 of the Optimizer Reference Manual)/XPRSmaxim (p.285 of the Optimizer Reference Manual)/XPRSglobal (p.238 of the Optimizer Reference Manual). The solver can then use the loaded solutions to update the best solution and/or as a seed for heuristic searches.

Solutions loaded directly from attached problems (compared with solutions loaded by the user from file or from memory) are marked with the MSP_SOL_ISUSERSOLUTION solution attribute set to 0.

Note that problems are automatically detached from their MIP solution pool when they are destroyed (with a call to XPRSdestroyprob (p.163 of the Optimizer Reference Manual). A MIP solution pool automatically detaches all problems when it is destroyed (with a call to XPRS_msp_destroy).

Example:-

```
#include <stdio.h>
#include <stdio.h>
#include "xprs.h"

int main(int argc, char **argv) {
  XPRSprob prob;
  XPRSmipsolpool msp;
  int i, nSols, nCols, iSolutionId, iSolutionIdStatus;
  double dObj, dSol;
  const char *sProblem = argv[1];

  XPRSinit(NULL);

  XPRScreateprob(&prob);

  XPRSreadprob(prob, sProblem, "");

  XPRS_msp_create(&msp);

  XPRS_msp_probattach(msp, prob);

  XPRSmaxim(prob, "g");

  XPRS_msp_getintattrib(msp, XPRS_MSP_SOLUTIONS, &nSols);

  if(nSols) {

    XPRS_msp_getdblattribprobextreme(msp, prob, 0, &iSolutionId, XPRS_MSP_SOLPRB_OBJ,
                                     &dObj);

    printf("Optimal Solution ID: %i\n", iSolutionId);
```

```
    printf("Optimal Objective  : %12.5f\n", dObj);

    XPRS_msp_getintattribsol(msp, iSolutionId, &iSolutionIdStatus, XPRS_MSP_SOL_COLS,
                             &nCols);

    for(i = 0; i < nCols; i++) {
      XPRS_msp_getsol(msp, iSolutionId, &iSolutionIdStatus, &dSol, i, i, NULL);
      printf("%3i = %12.5f\n", i, dSol);
    }

  }

  XPRSdestroyprob(prob);

  XPRS_msp_destroy(msp);

  XPRSfree();
}
```

1.3.2 Input from file and from memory

Solutions can be input into the MIP solution pool via the file reader routine XPRS_msp_readslxsol or from memory via the XPRS_msp_loadsol routine. When solutions are loaded in these ways the user is returned the ID(s) of the stored solutions. When solutions are input from memory via XPRS_msp_loadsol only one solution is stored per call and the user is returned the ID of the stored solution each time. When solutions are input from file via XPRS_msp_readslxsol the user is returned the first and last ID of the set of solutions that were successfully stored.

Note that solutions are not stored with a mapping of column name to solution index. Therefore, once a solution is stored the user is responsible for mapping the solution values against any column names. The indexing of the solution values is determined when the solution is loaded. When loading solutions from memory with a call to XPRS_msp_loadsol, the solution values are indexed by their position in the dense array. Similarly, solutions loaded automatically to the MIP solution pool from a problem are also indexed in this way.

When loading solutions from .slx file using XPRS_msp_readslxsol the file format has column name/solution value pairs. Solution values may be mapped to solution index using a runtime supplied name map object of type XPRSnamelist passed in the call to XPRS_msp_readslxsol. The XPRSnamelist object is typically obtained from an XPRSprob by calling XPRSgetnamelistobject (p.210 of the Optimizer Reference Manual) on the problem to get the column name list (type 2). If the XPRSnamelist object is passed as NULL the solution values are indexed by the order they are encountered in the file.

Solutions loaded by the user from file or from memory (compared with solutions loaded directly from attached problems) are marked with the MSP_SOL_ISUSERSOLUTION solution attribute set to 1.

1.4 Solution querying

Generally, users will want to query stored solutions for their solution values. The values for solutions can be written to file or passed to the user in memory using the XPRS_msp_writeslxsol and XPRS_msp_getsol calls, respectively.

The user may be interested in abstract attributes of the solutions and the MIP solution pool itself. For example, the user may want to know the number of solutions stored in the solution pool or the number of non-zero values in a solution. They may also be interested in the objective value of a solution with respect to a given problem or they may want to know how many feasible solutions the MIP solution pool contains for a given problem. The user may perhaps also want to get the list of solutions that are feasible for a given problem. The following sections discuss the

various classes of these attributes and how they are accessed.

1.4.1 Solution value output to file and memory

The solution values are accessed in two ways. They are written to file using the routine XPRS_msp_writeslxsol and they are retrieved in memory using the routine XPRS_msp_getsol.

The routine XPRS_msp_writeslxsol allows one or more solution to be written to file. If XPRS_msp_writeslxsol is passed with a problem (XPRSprob) pointer then the user can specify that (i) a particular solution is written out, (ii) the best solution for the problem is written out or (iii) all feasible solutions for the problem are written out. If no problem pointer is provided then the user can specify that (i) a particular solution is written out or (ii) all solutions are written out. The output .slx file has a format based on the MPS format. Each solution has a section in the .slx file starting with the string 'NAME' followed on the same line by the name of the solution. The records of each section each correspond to the solution value at the associated position in the solution array. If a problem is provided in the call then the column name is taken from the given problem for the index position and is written out followed by the solution value; otherwise column names are generated automatically based on a one-based indexing of the solution values.

The routine XPRS_msp_getsol provides in memory access to the values of a particular solution. The routine can return all solution values or solution values over a range of indices.

1.4.2 Attributes

A MIP solution pool and its set of stored solutions have various attributes the user can query. For example, the user can query the number of solutions stored in the solution pool or the number of non-zeros in a solution. They may also, for example, query the objective value of a solution with respect to a given problem or how many feasible solutions the MIP solution pool contains for a given problem.

There are four classes of attributes:-

- MIP solution pool attributes.
- Solution attributes.
- Solution and problem pair attributes.
- Problem attributes.

Each class relates to a different set of entities in the data model of the MIP solution pool and has a particular associated routine for accessing the attribute values. The following sections discuss the attributes.

1.4.3 MIP solution pool attributes

These are attributes relating to the MIP solution pool as a whole. For example, the number of solutions stored in the solution pool. These attributes are accessed using routines XPRS_msp_getintattrib and XPRS_msp_getdblattrib.

1.4.4 Solution attributes

These are attributes relating to a particular solution. For example, the number of non-zero values in a solution. These attributes are accessed using routines XPRS_msp_getintattribsol and XPRS_msp_getdblattribsol.

1.4.5 Problem attributes

These are attributes relating to the set of solutions with respect to a particular problem. For example, how many solutions the MIP solution pool contains that have the same number of columns as a given problem. These attributes are accessed using routines `XPRS_msp_getintattribprob` and `XPRS_msp_getdblattribprob`.

1.4.6 Solution and problem pair attributes

These are attributes relating to a solution and problem pair. For example, the objective value of a solution with respect to a given problem. These attributes are accessed using routines `XPRS_msp_getintattribprobsol` and `XPRS_msp_getdblattribprobsol`.

The attributes here are a function of a solution and problem pair. However, the user may be interested in the maximum or minimum value of an attribute over all solutions for a given problem. For example, the minimum objective value of any stored solution with respect to a given problem. The values for the attributes in this case are accessed using routines `XPRS_msp_getintattribprobextreme` and `XPRS_msp_getdblattribprobextreme`. These routines are special in that, as well as the returning attribute value, they also return the ID of a solution that achieves the attribute value.

1.5 Getting lists of solutions

Apart from just getting an attribute value for a given solution and problem (using the functions `XPRS_msp_getintattribprobsol` and `XPRS_msp_getdblattribprobsol`) the user may also be interested in how an attribute behaves over a set of solutions. For example, the user may want the objective function values for the solutions with respect to a given problem. Another example may be that the user wants the MIP infeasibilities of the solutions with respect to a given problem.

The function `XPRS_msp_getsollist` is used to query for a list of solutions based on an attribute. The solutions are returned as a list of solution IDs. The user can choose to rank the returned solutions by their attribute values and they may also choose to return solutions in a range of the ranked ordering of the solutions. The user can, for example, get the objective functions of the solutions ranked in ascending order so they can find the one (or more) solutions that have the minimum objective value.

Note that `XPRS_msp_getsollist` will only return solutions for which the attribute is relevant. For example, if the user queries on the objective function values then only feasible solutions of the problem are returned. If, for example, the users queries on MIP infeasibilities then only those solutions with any MIP infeasibilities are returned.

A function `XPRS_msp_getsollist2`, similar to `XPRS_msp_getsollist`, provides the user with additional filtering control on the returned list of solution IDs.

1.6 Control options

1.6.1 Duplicate solutions

Handling solution duplicates is an issue of particular relevance to the storage of solutions. Firstly, it is an issue with regards to the efficient use of the storage memory. It is also an issue with regards to how the set of stored solutions for a problem reflects the properties of the problem. For example, consider that we query, using `XPRS_msp_getsollist`, the solution pool looking for MIP solutions of a problem and we find 10 solutions with the same, optimal MIP objective for the

problem. We may believe that the problem has multiple optimal MIP solutions. However, if these solutions are all duplicates then there is only one solution and we do not have proof the problem has multiple optimal solutions.

The MIP solution pool provides a set of policies the user can choose as an integer control parameter MSP_DUPLICATESOLUTIONSPOLICY for automatically handling the exclusion of solutions that are duplicated. These are:-

- Keep all: There is no checking for duplicate solutions.
- Exact: Solution pairs are the same if all variables have exactly the same value. Duplicates solutions are discarded.
- Exact and rounded: Solution pairs are the same if all continuous variables have exactly the same value and the global component of the solution matches i.e., matches within tolerance. Duplicates solutions are discarded.
- Rounded only: Solution pairs are the same if the global components of the solutions match. Duplicates solutions are discarded.

When the policy requires duplicate solutions be discarded the MIP solution pool compares all pairs of solutions to determine duplicates. If a solution is found to be duplicated by another solution then the solution with the larger solution id value is discarded i.e., a solution that is stored earlier is kept in place of a later, duplicate solution.

The global component of the solutions is determined by the global model of one of the 'attached' problems. The attached problem that defines the global model is the first problem to be attached to the MIP solution pool once the global model is undefined. The global model is undefined when the MIP solution pool is first created and after the problem currently defining the global model is 'detached'.

When comparing the global component of two solutions there are two types of tolerance used. The MIPTOL (p.428 of the Optimizer Reference Manual) tolerance is used to define the rounded values of the variables and the FEASTOL (p.402 of the Optimizer Reference Manual) tolerance is used to define when solution values are zero. Each solution has a control for each of these tolerances identified by MSP_SOL_MIPTOL and MSP_SOL_FEASTOL. The user can access these control values for a solution using XPRS_msp_getdblcontrolsol and XPRS_msp_setdblcontrolsol.

The global component of a solution (assuming it is MIP feasible with respect to the global model) is insensitive to small floating point variations in the values of the double solution variables e.g., a binary variable in a solution pair is compared as if the double solution value was rounded to 0 or 1. Therefore, comparing the global components of two solutions will match more readily than comparing the variable values with exact matches. As a result there will be no fewer solutions discarded using the global components comparison than there will be discarded using exact value matches (and likely there will be more solutions discarded). For this reason it is important to use care when using the policies that include global component comparisons since it is possible that solutions of interest may be deleted from the pool. Note that this may be a problem in the case where there is more than one global model for the solutions being stored in the MIP solution pool. We may delete solutions that are considered to be duplicates with regards to the global model used by the pool but are not considered duplicates by the other global model.

CHAPTER 2

MSP Functions

XPRS_msp_addcbmsghandler

Purpose

Declares an output callback function, called every time a line of message text is output by the MIP solution pool. This callback function will be called in addition to any output callbacks already added by XPRS_msp_addcbmsghandler.

Synopsis

```
int XPRS_CC XPRS_msp_addcbmsghandler(XPRSmipsolpool msp, int (XPRS_CC *f_msghandler)
       (XPRSobject vXPRSObject, void * vUserContext, void * vSystemThreadId, const
       char * sMsg, int iMsgType, int iMsgNumber), void * p, int priority);
```

Arguments

msp The current MIP solution pool

f_msghandler The callback function which takes six arguments, vXPRSObject, vUserContext, vSystemThreadId, sMsg, iMsgType and iMsgNumber. Use a NULL value to cancel a callback function.

vXPRSObject The object sending the message. Use XPRSgetobjecttypename (p.213 of the Optimizer Reference Manual) to get the name of the object type.

vUserContext The user-defined object passed to the callback function.

vSystemThreadId The system id of the thread sending the message caste to a void *.

sMsg A null terminated character array (string) containing the message, which may simply be a new line. When the callback is called for the first time sMsg will be a NULL pointer.

iMsgType Indicates the type of output message:

1 information messages;
2 (not used);
3 warning messages;
4 error messages.

A negative value means the callback is being called for the first time.

iMsgNumber The number associated with the message. If the message is an error or a warning then you can look up the number in the section Optimizer Error and Warning Messages for advice on what it means and how to resolve the associated issue.

p A user-defined object to be passed to the callback function.

priority An integer that determines the order in which multiple output callbacks will be invoked. The callback added with a higher priority will be called before a callback with a lower priority. Set to 0 if not required.

Further information

To send messages to a log file the built in message handler XPRSlogfilehandler can be used. This can be done with:

```
XPRS_msp_addcbmsghandler(msp, XPRSlogfilehandler, "log.txt");
```

Related topics

XPRS_msp_removecbmsghandler, XPRSgetobjecttypename (p.213 of the Optimizer Reference Manual).

XPRS_msp_create

Purpose

Sets up a new MIP solution pool object.

Synopsis

```
int XPRS_CC XPRS_msp_create(XPRSmipsolpool *msp)
```

Argument

`msp` Pointer to a variable holding the new MIP solution pool.

Further information

1. Calls to `XPRS_msp_create` must be made after the call to `XPRSinit` (p.252 of the Optimizer Reference Manual).
2. All MIP solution pools created using a call to `XPRS_msp_create` should be disposed of with a call to `XPRS_msp_destroy`.

Related topics

`XPRSinit` (p.252 of the Optimizer Reference Manual), `XPRS_msp_destroy`.

XPRS_msp_delsol

Purpose

Deletes a solution from the pool.

Synopsis

```
int XPRS_CC XPRS_msp_delsol(XPRSmipsolpool msp, const int iSolutionId, int * const
       iSolutionIdStatus)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` The id of the solution to be deleted.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution which is deleted.

Further information

The user will obtain the solution id `iSolutionId` from interaction with the MIP solution pool via functions such as `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme` and `XPRS_msp_getsollist`.

Related topics

`XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme`, `XPRS_msp_getsollist`.

XPRS_msp_destroy

Purpose

Destroys a MIP solution pool object and its resources. The object will generally be created by a call to the function XPRS_msp_create.

Synopsis

```
int XPRS_CC XPRS_msp_destroy(XPRSmipsolpool msp)
```

Argument

`msp` MIP solution pool to be destroyed.

Related topics

XPRS_msp_create.

XPRS_msp_findsolbyname

Purpose

Finds the id of a solution given the solution's name.

Synopsis

```
int XPRS_CC XPRS_msp_findsolbyname(XPRSmipsolpool msp, const char *sSolutionName, int
       *const iSolutionId)
```

Arguments

`msp` The current MIP solution pool.

`sSolutionName` A null terminated string containing the name of the solution to find.

`iSolutionId` Pointer to the integer where the solution id will be returned. A value of 0 will be returned if the solution name does not exist.

Further information

Names of solutions enter the MIP solution pool via functions such as `XPRS_msp_loadsol`, `XPRS_msp_setsolname` and `XPRS_msp_readslxsol`. Solutions entering the MIP solution pool directly from attached problems are given automatic names.

Related topics

`XPRS_msp_loadsol`, `XPRS_msp_setsolname`, `XPRS_msp_getsolname`, `XPRS_msp_readslxsol`.

XPRS_msp_getcbmsghandler

Purpose

Get the output callback function, as set by XPRS_msp_setcbmsghandler.

Synopsis

```
int XPRS_CC XPRS_msp_getcbmsghandler(XPRSmipsolpool msp, int (XPRS_CC
       **r_f_msghandler)
       (XPRSobject vXPRSObject, void * vUserContext, void * vSystemThreadId, const
       char * sMsg, int iMsgType, int iMsgNumber), void **object);
```

Arguments

`msp` The current MIP solution pool.

`r_f_msghandler` Pointer to the memory where the callback function will be returned.

Related topics

XPRS_msp_setcbmsghandler.

XPRS_msp_getdblattrib

Purpose

Provides read access to the values of double attributes associated with the MIP solution pool.

Synopsis

```
int XPRS_CC XPRS_msp_getdblattrib(XPRSmipsolpool msp, const int iAttribId, double *
       const Val)
```

Arguments

`msp` The current MIP solution pool.

`iAttribId` Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.

`Val` Pointer to a double where the value of the attribute will be returned.

Related topics

XPRS_msp_getintattrib

XPRS_msp_getdblattribprob

Purpose

Provides read access to the values of double attributes associated with the MIP solution pool with respect to the given problem.

Synopsis

```
int XPRS_CC XPRS_msp_getdblattribprob(XPRSmipsolpool msp, XPRSprob prob, const int
       iAttribId, double * const Dst)
```

Arguments

`msp`	The current MIP solution pool.
`prob`	Problem for which the attribute is required to be evaluated.
`iAttribId`	Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.
`Dst`	Pointer to a double where the value of the attribute will be returned.

Related topics

`XPRS_msp_getintattribprob`.

XPRS_msp_getdblattribprobextreme

Purpose

Retrieves the extreme value of a double attribute associated with the set of solutions stored in the MIP solution pool evaluated with respect to the given problem. For example, the minim of the objective values of feasible solutions for the problem across the solutions stored in the MIP solution pool.

Synopsis

```
int XPRS_CC XPRS_msp_getdblattribprobextreme(XPRSmipsolpool msp, XPRSprob
       prob_to_rank_against, const int bGet_Max_Otherwise_Min, int * const
       iSolutionId, const int iAttribId, double * const ExtremeVal)
```

Arguments

`msp` The current MIP solution pool.

`prob_to_rank_against` Problem for which the attribute is required to be evaluated.

`bGet_Max_Otherwise_Min` If set to a non-zero value the routine will return the maximum attribute value; otherwise the minim attribute value will be returned.

`iAttribId` Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.

`iSolutionId` The id of the solution for which the returned attribute value was evaluated with the problem `prob_to_rank_against`.

`ExtremeVal` Pointer to a double where the value of the attribute will be returned.

Further information

1. This function returns objective values only for feasible solutions. For attributes representing infeasibility this function returns a meaningful value only if there any solutions with an infeasibility associated with the attribute. For example, if the attribute represents the sum of column bound infeasibilities for solutions in the MIP solution pool then a meaningful value is only returned if there are any solutions with column bound infeasibilities for the problem.

2. The attributes accessed by this function are the same as those available through `XPRS_msp_getdblattribprobsol`. That is, double attributes relating to a problem and solution pair, e.g., the objective function value of a solution to the problem

3. If there is no value returned then the value of `iSolutionId` is set to 0; otherwise it will be the positive id of the solution with the returned attribute value.

Related topics

`XPRS_msp_getintattribprobextreme`, `XPRS_msp_getdblattribprobsol`.

XPRS_msp_getdblattribprobsol

Purpose

Provides read access to the values of double attributes associated with the MIP solution pool with respect to the given solution stored in the MIP solution pool and the given problem. For example, the objective value of the given solution with respect to the given problem.

Synopsis

```
int XPRS_CC XPRS_msp_getdblattribprobsol(MipSolPool msp, XPRSprob
       prob_to_rank_against, const int iSolutionId, int * const iSolutionIdStatus,
       const int iAttribId, double * const Dst)
```

Arguments

`msp` The current MIP solution pool.

`prob_context` Problem for which the attribute `iAttribId` is required to be evaluated using solution `iSolutionId` .

`iSolutionId` The id of the solution for which the attribute `iAttribId` is evaluated with respect to the problem `prob_context`.

`iSolutionIdStatus` Pointer to an int where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`iAttribId` Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.

`Dst` Pointer to a double where the value of the attribute will be returned.

Further information

1. The user will obtain the solution id `iSolutionId` from interaction with the MIP solution pool via functions such as `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme` and `XPRS_msp_getsollist`.

2. The attributes accessed by this function are the same as those available through `XPRS_msp_getdblattribprobextreme`. That is, double attributes relating to a problem and solution pair, e.g., the objective function value of a solution to the problem

Related topics

`XPRS_msp_getintattribprobsol`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_findsolbyname`, `XPRS_msp_getintattribprobextreme`, `XPRS_msp_getsollist`.

XPRS_msp_getdblattribsol

Purpose

Provides read access to the values of double attributes associated a solution stored in the MIP solution pool.

Synopsis

```
int XPRS_CC XPRS_msp_getdblattribsol(XPRSmipsolpool msp, const int iSolutionId, int *
       const iSolutionIdStatus, const int iAttribId, double * const Dst)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` The id of the solution for which the attribute is to be returned.

`iSolutionIdStatus` Pointer to an int where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`iAttribId` Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.

`Dst` Pointer to a double where the value of the attribute will be returned.

Further information

The user will obtain the solution id `iSolutionId` from interaction with the MIP solution pool via functions such as `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme` and `XPRS_msp_getsollist`.

Related topics

`XPRS_msp_getintattribsol`, `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme`, `XPRS_msp_getsollist`.

XPRS_msp_getdblcontrol

Purpose

Retrieves the value of a given double control parameter.

Synopsis

```
int XPRS_CC XPRS_msp_getdblcontrol(XPRSmipsolpool msp, const int iControlId, double *
       const Val)
```

Arguments

`msp` The current MIP solution pool.

`iControlId` Id of control whose value is to be returned. A full list of all available controls may be found in Chapter 4, or from the list in the `xprs.h` header file.

`Val` Pointer to a double where the value of the control will be returned.

Related topics

`XPRS_msp_getintcontrol`, `XPRS_msp_setdblcontrol`, `XPRS_msp_setintcontrol`.

XPRS_msp_getdblcontrolsol

Purpose

Retrieves the value of a given double control parameter.

Synopsis

```
int XPRS_CC XPRS_msp_getdblcontrolsol(XPRSmipsolpool msp, int iSolutionId, int *
       iSolutionIdStatus, const int iControlId, double * const Val)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` The id of the solution for which the control is to be returned.

`iSolutionIdStatus` Pointer to an int where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`iControlId` Id of control whose value is to be returned. A full list of all available controls may be found in Chapter 3, or from the list in the `xprs.h` header file.

`Val` Pointer to a double where the value of the control will be returned.

Related topics

`XPRS_msp_getintcontrolsol`, `XPRS_msp_setdblcontrolsol`, `XPRS_msp_setintcontrolsol`.

XPRS_msp_getintattrib

Purpose

Provides read access to the values of integer attributes associated with the MIP solution pool.

Synopsis

```
int XPRS_CC XPRS_msp_getintattrib(XPRSmipsolpool msp, const int iAttribId, int * const
       Val)
```

Arguments

`msp`	The current MIP solution pool.
`iAttribId`	Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.
`Val`	Pointer to an integer where the value of the attribute will be returned.

Related topics

`XPRS_msp_getdblattrib`.

XPRS_msp_getintattribprob

Purpose

Provides read access to the values of integer attributes associated with the MIP solution pool with respect to the given problem. For example, the number of solutions in the MIP solution pool that are feasible for the problem.

Synopsis

```
int XPRS_CC XPRS_msp_getintattribprob(XPRSmipsolpool msp, XPRSprob prob, const int
       iAttribId, int * const Dst)
```

Arguments

`msp`	The current MIP solution pool.
`prob`	Problem for which the attribute is required to be evaluated.
`iAttribId`	Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.
`Dst`	Pointer to an integer where the value of the attribute will be returned.

Related topics

`XPRS_msp_getdblattribprob`.

XPRS_msp_getintattribprobextreme

Purpose

Retrieves the extreme value of an integer attribute associated with the set of solutions stored in the MIP solution pool evaluated with respect to the given problem. For example, the minim of the count of column bound infeasibilities for solutions of the problem taken across solutions stored in the MIP solution pool that have at least one column bound infeasibility for the problem.

Synopsis

```
int XPRS_CC XPRS_msp_getintattribprobextreme(XPRSmipsolpool msp, XPRSprob
       prob_to_rank_against, const int bGet_Max_Otherwise_Min, int * const
       iSolutionId, const int iAttribId, int * const ExtremeVal)
```

Arguments

`msp` The current MIP solution pool.

`prob_to_rank_against` Problem for which the attribute is required to be evaluated.

`bGet_Max_Otherwise_Min` If set to a non-zero value the routine will return the maximum attribute value; otherwise the minim attribute value will be returned.

`iAttribId` Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.

`iSolutionId` The id of the solution for which the returned attribute value was evaluated with the problem `prob_to_rank_against`.

`ExtremeVal` Pointer to an integer where the value of the attribute will be returned.

Further information

1. For attributes representing infeasibility this function returns a meaningful value only if there any solutions with an infeasibility associated with the attribute. For example, if the attribute represents the count of column bound infeasibilities for solutions in the MIP solution pool then a meaningful value is only returned if there are any solutions with column bound infeasibilities for the problem.

2. The attributes accessed by this function are the same as those available through `XPRS_msp_getintattribprobsol`. That is, integer attributes relating to a problem and solution pair, e.g., the count of column bound infeasibilities for solutions of the problem.

3. If there is no value returned then the value of `iSolutionId` is set to 0; otherwise it will be the positive id of the solution with the returned attribute value.

Related topics

`XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobsol`.

XPRS_msp_getintattribprobsol

Purpose

Provides read access to the values of integer attributes associated with the MIP solution pool with respect to the given solution stored in the MIP solution pool and the given problem. For example, the count of column bound infeasibilities for solutions of the problem.

Synopsis

```
int XPRS_CC XPRS_msp_getdblattribprobsol(MipSolPool msp, XPRSprob
       prob_to_rank_against, const int iSolutionId, int * const iSolutionIdStatus,
       const int iAttribId, double * const Dst)
```

Arguments

`msp` The current MIP solution pool.

`prob_context` Problem for which the attribute `iAttribId` is required to be evaluated using solution `iSolutionId`.

`iSolutionId` The id of the solution for which the attribute `iAttribId` is evaluated with respect to the problem `prob_context`.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`iAttribId` Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.

`Dst` Pointer to an integer where the value of the attribute will be returned.

Further information

1. The user will obtain the solution id iSolutionId from interaction with the MIP solution pool via functions such as `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme` and `XPRS_msp_getsollist`.

2. The attributes accessed by this function are the same as those available through `XPRS_msp_getintattribprobextreme`. That is, integer attributes relating to a problem and solution pair, e.g., the count of column bound infeasibilities for solutions of the problem.

Related topics

`XPRS_msp_getdblattribprobsol`, `XPRS_msp_getintattribprobextreme`, `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getsollist`.

XPRS_msp_getintattribsol

Purpose

Provides read access to the values of integer attributes associated with a solution stored in the MIP solution pool. For example, the number of columns in the solution.

Synopsis

```
int XPRS_CC XPRS_msp_getintattribsol(XPRSmipsolpool msp, const int iSolutionId, int *
       const iSolutionIdStatus, const int iAttribId, int * const Dst)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` The id of the solution for which the attribute is to be returned.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`iAttribId` Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file.

`Dst` Pointer to an integer where the value of the attribute will be returned.

Further information

The user will obtain the solution id iSolutionId from interaction with the MIP solution pool via functions such as `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme` and `XPRS_msp_getsollist`.

Related topics

`XPRS_msp_getdblattribsol`, `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme`, `XPRS_msp_getsollist`.

XPRS_msp_getintcontrol

Purpose

Retrieves the value of a given integer control parameter.

Synopsis

```
int XPRS_CC XPRS_msp_getintcontrol(XPRSmipsolpool msp, const int iControlId, int *
       const Val)
```

Arguments

`msp` The current MIP solution pool.

`iControlId` Id of control whose value is to be returned. A full list of all available controls may be found in Chapter 3, or from the list in the `xprs.h` header file.

`Val` Pointer to an integer where the value of the control will be returned.

Related topics

`XPRS_msp_getdblcontrol`, `XPRS_msp_setdblcontrol`, `XPRS_msp_setintcontrol`.

XPRS_msp_getintcontrolsol

Purpose

Retrieves the value of a given integer control parameter.

Synopsis

```
int XPRS_CC XPRS_msp_getintcontrolsol(XPRSmipsolpool msp, int iSolutionId, int *
       iSolutionIdStatus, const int iControlId, int * const Val)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` The id of the solution for which the control is to be returned.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`iControlId` Id of control whose value is to be returned. A full list of all available controls may be found in Chapter 3, or from the list in the `xprs.h` header file.

`Val` Pointer to an integer where the value of the control will be returned.

Related topics

`XPRS_msp_getdblcontrolsol`, `XPRS_msp_setdblcontrolsol`, `XPRS_msp_setintcontrolsol`.

XPRS_msp_getlasterror

Purpose

Gets the last error message.

Synopsis

```
int XPRS_CC XPRS_msp_getlasterror(XPRSmipsolpool msp, int * iMsgNumber, char * msg,
       int iStringBufferBytes, int * iBytesInInternalString)
```

Arguments

`mse` The current MIP solution pool.

`iMsgNumber` A pointer to an integer to return the number of the last error message. Can be NULL if not required. Refer to Chapter 11 of the Optimizer Reference Manual for a list of possible error numbers, the errors and warnings that they indicate, and advice on what they mean and how to resolve them.

`msg` A character buffer of length at least `iStringBufferBytes` to return the error message. Can be NULL if not required.

`iStringBufferBytes` The length of the `msg` buffer.

`iBytesInInternalString` A pointer to an integer to return the number of bytes required to store the error message. Can be NULL if not required.

Related topics

Chapter 11 of the Optimizer Reference Manual , `XPRS_msp_setcbmsghandler`.

XPRS_msp_getsol

Purpose

Returns the solution values for a solution stored in the MIP solution pool.

Synopsis

```
int XPRS_CC XPRS_msp_getsol(XPRSmipsolpool msp, const int iSolutionId, int * const
       iSolutionIdStatus, double x[], const int iColFirst, const int iColLast, int *
       const nValuesReturned)
```

Arguments

`Msp` The current MIP solution pool.

`iSolutionId` The id of the required solution.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`x` Double array where the values of the solution will be returned. May be NULL if not required.

`iColFirst` Index of the column whose solution value is to be returned in the first element of array `x`. If `iColLast > iColFirst` then any subsequent columns are to have their values written to the subsequent elements of `x`.

`iColLast` If `x` is non-NULL then at most `iColLast - iColFirst + 1` solution values will be written to `x`. There will be fewer solution values written if `iColLast` is greater than or equal to the number of columns in the solution.

`nValuesReturned` Pointer to an integer where the number of solution values that were available to be written is returned. This number will always be less than or equal to `iColLast - iColFirst + 1`. A value is returned for this parameter regardless of whether `x` is passed as NULL. May be NULL if not required.

Further information

1. It is an error condition if `iColLast <iColFirst` or `iColFirst <0` or if `iColFirst` is greater than or equal to the number of columns in the solution.
2. The user will obtain the solution id `iSolutionId` from interaction with the MIP solution pool via functions such as `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme` and `XPRS_msp_getsollist`.
3. By passing `x` as NULL and setting `iColFirst` to 0 and `iColLast` to a large positive integer it is possible to use `XPRS_msp_getsol` to obtain in the `nValuesReturned` parameter the number of columns in the solution.

Related topics

`XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme`, `XPRS_msp_getsollist`.

XPRS_msp_getsollist

Purpose

Returns a list of solution ids of solutions with the same number of columns as `prob_to_rank_against`. The list may be sorted by the value of some attribute of the solutions e.g., the objective function. The list may be filtered to contain either only feasible solutions or only infeasible solutions.

Synopsis

```
int XPRS_CC XPRS_msp_getsollist(XPRSmipsolpool msp, XPRSprob prob_to_rank_against,
       const int iRankAttrib, const int bRankAscending, const int iRankFirstIndex,
       const int iRankLastIndex, int iSolutionIds_Zb[], int * const nReturnedSolIds,
       int * const nSols)
```

Arguments

`msp` The current MIP solution pool.

`prob_to_rank_against` Problem for which the attribute `iRankAttrib` is evaluated in order to rank the solution ids returned in the `iSolutionIds` array. Only solutions with the same number of columns as `prob_to_rank_against` are considered when generating the list of solution ids. May be NULL if the IDs of all solutions are to be returned.

`iRankAttrib` Id of the attribute whose value is used to rank the solution ids returned in the `iSolutionIds` array. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file. If the attribute represents infeasibility then only infeasible solutions will be returned; otherwise only feasible solutions will be returned. If `iRankAttrib` is passed in as 0, or if it is not a valid attribute id, then the solution ids of feasible solutions for the problem will be returned unranked. If iRankAttrib is not a valid attribute id then a warning is logged. This argument is ignored if prob_to_rank_against is passed as NULL.

`bRankAscending` If set to a non-zero value the function returns the solution ids ordered in ascending order of the attribute value; otherwise the solution ids are returned in descending order. This argument is ignored if `prob_to_rank_against` is passed as NULL or if `iRankAttrib` is passed in as 0, or if it is not a valid attribute id.

`iRankFirstIndex` Index (one-based) in the rank order of solutions for which the associated solution's id number, if there is a solution at this index, is to be returned in the first element of array `iSolutionIds`. If `iRankLastIndex > iRankFirstIndex` then any subsequent solutions in the rank ordering are to have their solution ids written to the subsequent elements of `iSolutionIds`.

`iRankLastIndex` If iSolutionIds is non-NULL then at most `iRankLastIndex - iRankFirstIndex + 1` solution ids will be written to `iSolutionIds`. There will be fewer solution ids written if `iRankLastIndex` is greater than the number of solutions in the MIP solution pool.

`iSolutionIds` Integer array where the solution ids will be returned. May be NULL if not required.

`nReturnedSolIds` Pointer to an integer where the number of solution ids that were available to be written is returned. This number will always be less than or equal to `iRankLastIndex - iRankFirstIndex + 1`. A value is returned for this parameter regardless of whether `iSolutionIds` is passed as NULL. May be NULL if not required.

`nSols` Pointer to an integer where the total number of solution ids that could possibly be written is returned. May be NULL if not required.

Further information

1. If `iRankFirstIndex` > `iRankLastIndex` and the `nSols` argument is passed down as NULL then the routine will return with no solution ids written.
2. Information about the solution with respect to the problem can be obtained via routines `XPRS_msp_getdblattribprobsol`, `XPRS_msp_getintattribprobsol`, `XPRS_msp_getdblattribsol` and `XPRS_msp_getintattribsol`. The solution values can be obtained via the routine `XPRS_msp_getsol`.

Related topics

`XPRS_msp_getsollist2`, `XPRS_msp_getsol`, `XPRS_msp_getdblattribprobsol`, `XPRS_msp_getintattribprobsol`, `XPRS_msp_getdblattribsol`, `XPRS_msp_getintattribsol`.

XPRS_msp_getsollist2

Purpose

Returns a list of solution ids of solutions with the same number of columns as `prob_to_rank_against`. The list may be sorted by the value of some attribute of the solutions e.g., the objective function. The list may be filtered to contain either only feasible solutions or only infeasible solutions. This function is the same as `XPRS_msp_getsollist` in terms of the purpose described so far. The additional functionality provided by `XPRS_msp_getsollist2` is to allow the returned solutions to be filtered logically on the values of `MSP_SOL_BITFIELDSSYS` and `MSP_SOL_BITFIELDSUSR`.

Synopsis

```
int XPRS_CC XPRS_msp_getsollist2(XPRSmipsolpool msp, XPRSprob prob_to_rank_against,
       int iRankAttrib, int bRankAscending, int iRankFirstIndex_Ob, int
       iRankLastIndex_Ob, int bUseUserBitFilter, int iUserBitMask, int
       iUserBitPattern, int bUseInternalBitFilter, int iInternalBitMask, int
       iInternalBitPattern, int iSolutionIds_Zb[], int * nReturnedSolIds, int * nSols)
```

Arguments

`msp` The current MIP solution pool.

`prob_to_rank_against` Problem for which the attribute `iRankAttrib` is evaluated in order to rank the solution ids returned in the `iSolutionIds` array. Only solutions with the same number of columns as `prob_to_rank_against` are considered when generating the list of solution ids. May be NULL if the IDs of all solutions are to be returned.

`iRankAttrib` Id of the attribute whose value is used to rank the solution ids returned in the `iSolutionIds` array. A full list of all available attributes may be found in Chapter 4, or from the list in the `xprs.h` header file. If the attribute represents infeasibility then only infeasible solutions will be returned; otherwise only feasible solutions will be returned. If `iRankAttrib` is passed in as 0, or if it is not a valid attribute id, then the solution ids of feasible solutions for the problem will be returned unranked. If `iRankAttrib` is not a valid attribute id then a warning is logged. This argument is ignored if `prob_to_rank_against` is passed as NULL.

`bRankAscending` If set to a non-zero value the function returns the solution ids ordered in ascending order of the attribute value; otherwise the solution ids are returned in descending order. This argument is ignored if `prob_to_rank_against` is passed as NULL or if `iRankAttrib` is passed in as 0, or if it is not a valid attribute id.

`iRankFirstIndex` Index (one-based) in the rank order of solutions for which the associated solution's id number, if there is a solution at this index, is to be returned in the first element of array `iSolutionIds`. If `iRankLastIndex > iRankFirstIndex` then any subsequent solutions in the rank ordering are to have their solution ids written to the subsequent elements of `iSolutionIds`.

`iRankLastIndex` If `iSolutionIds` is non-NULL then at most `iRankLastIndex - iRankFirstIndex + 1` solution ids will be written to `iSolutionIds`. There will be fewer solution ids written if `iRankLastIndex` is greater than the number of solutions in the MIP solution pool.

`bUseUserBitFilter` Set this value to 1 if the solutions should be filtered by their `MSP_SOL_BITFIELDSUSR` control values using the values of `iUserBitMask` and `iUserBitPattern`. If we let `iBitFieldUser` be the value of the `MSP_SOL_BITFIELDSUSR` control for a solution then the solution will not be returned in the list if `(iBitFieldUser & iUserBitMask) != iUserBitPattern`. Set to 0 if filtering is not required on the `MSP_SOL_BITFIELDSUSR` control values.

`iUserBitMask` Bit mask used to filter solutions by their `MSP_SOL_BITFIELDSUSR` control values.

`iUserBitPattern` Bit pattern used to filter solutions by their `MSP_SOL_BITFIELDSUSR` control

values.

`bUseInternalBitFilter` Set this value to 1 if the solutions should be filtered by their `MSP_SOL_BITFIELDSSYS` attribute values using the values of `iInternalBitMask` and `iInternalBitPattern`. If we let `iBitFieldSys` be the value of the `MSP_SOL_BITFIELDSSYS` attribute for a solution then the solution will not be returned in the list if (`iBitFieldUser` & `iInternalBitMask`) `!= iInternalBitPattern`. Set to 0 if filtering is not required on the `MSP_SOL_BITFIELDSSYS` attribute values.

`iInternalBitMask` Bit mask used to filter solutions by `MSP_SOL_BITFIELDSSYS` attribute values.

`iInternalBitPattern` Bit pattern used to filter solutions by `MSP_SOL_BITFIELDSSYS` attribute values.

`iSolutionIds` Integer array where the solution ids will be returned. May be NULL if not required.

`nReturnedSolIds` Pointer to an integer where the number of solution ids that were available to be written is returned. This number will always be less than or equal to `iRankLastIndex - iRankFirstIndex + 1`. A value is returned for this parameter regardless of whether `iSolutionIds` is passed as NULL. May be NULL if not required.

`nSols` Pointer to an integer where the total number of solution ids that could possibly be written is returned. May be NULL if not required.

Further information

1. If `iRankFirstIndex > iRankLastIndex` and the `nSols` argument is passed down as NULL then the routine will return with no solution ids written.
2. Information about the solution with respect to the problem can be obtained via routines `XPRS_msp_getdblattribprobsol`, `XPRS_msp_getintattribprobsol`, `XPRS_msp_getdblattribsol` and `XPRS_msp_getintattribsol`. The solution values can be obtained via the routine `XPRS_msp_getsol`.

Related topics

`XPRS_msp_getsollist`, `XPRS_msp_getsol`, `XPRS_msp_getdblattribprobsol`, `XPRS_msp_getintattribprobsol`, `XPRS_msp_getdblattribsol`, `XPRS_msp_getintattribsol`.

XPRS_msp_getsolname

Purpose

Gets the name of a solution stored in a MIP solution pool.

Synopsis

```
int XPRS_CC XPRS_msp_getsolname(XPRSmipsolpool msp, const int iSolutionId, char
       *sname, const int iStringBufferBytes, int * const iBytesInInternalString, int *
       const iSolutionIdStatus)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` Id of the solution whose name is to be retrieved.

`sname` Pointer to a string where the name of the solution (plus null terminator) will be returned. This argument can be passed with NULL.

`iStringBufferBytes` Maximum number of bytes to be returned in sname.

`iBytesInInternalString` Pointer to the integer where the number of bytes required storing the name (including null terminator) will be returned.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

Further information

1. Names of solutions stored in a MIP solution pool are unique.
2. The user will obtain the solution id `iSolutionId` from interaction with the MIP solution pool via functions such as `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme` and `XPRS_msp_getsollist`.
3. Names of solutions enter the MIP solution pool via functions such as `XPRS_msp_loadsol`, `XPRS_msp_setsolname` and `XPRS_msp_readslxsol`. Solutions entering the MIP solution pool directly from attached problems are given automatic names.

Related topics

`XPRS_msp_setsolname`, `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme`, `XPRS_msp_getsollist`.

XPRS_msp_loadsol

Purpose

Loads a solution into a MIP solution pool.

Synopsis

```
int XPRS_CC XPRS_msp_loadsol(XPRSmipsolpool msp, int * const iSolutionId, const double
       x[], const int nCols, const char *sSolutionName, int * const
       bNameModifiedForUniqueness, int * const bSolutionNotLoadedBecauseOfDuplication)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` Pointer to the integer where the id of the loaded solution will be returned.

`x` Double array of length `nCols` containing the solution to be stored.

`nCols` The number of elements in the solution array `x`.

`sSolutionName` The name to use to identify the solution to be stored. If the given name already exists then the name to be stored will have characters appended to ensure the resulting name is unique among the stored solutions. If `sSolutionName` is passed as NULL then a unique name will be generated automatically for the solution.

`bNameModifiedForUniqueness` Pointer to an integer returning 1 if it was necessary for the given solution name `sNewSolutionBaseName` to be modified to maintain uniqueness of solution names; 0 otherwise.

`bSolutionNotLoadedBecauseOfDuplication` Pointer to an integer returning 1 if the solution was rejected because it was found to a duplicate of an existing solution; 0 otherwise.

Further information

Information about the solution with respect to a problem can be obtained via routines `XPRS_msp_getdblattribprobsol`, `XPRS_msp_getintattribprobsol`, `XPRS_msp_getdblattribsol` and `XPRS_msp_getintattribsol`.

Related topics

`XPRS_msp_getdblattribprobsol`, `XPRS_msp_getintattribprobsol`, `XPRS_msp_getdblattribsol`, `XPRS_msp_getintattribsol`.

XPRS_msp_probattach

Purpose

Attaches a problem to a MIP solution pool.

Synopsis

```
int XPRS_CC XPRS_msp_probattach(XPRSmipsolpool msp, XPRSprob prob)
```

Arguments

`msp`	The current MIP solution pool.
`prob`	Problem to be attached to the MIP solution pool.

Further information

1. The function will return successfully if the solution is already attached to the MIP solution pool.
2. One or more problems can be attached to the MIP solution pool. A problem can only be attached to one MIP solution pool.
3. The function will return an error if the problem is already attached to another MIP solution pool.
4. The user can detach the problem from the MIP solution pool with a call to `XPRS_msp_probdetach`.
5. A problem is automatically detached from a MIP solution pool when it is destroyed. A MIP solution pool automatically detaches all problems when it is destroyed.

Related topics

`XPRS_msp_probdetach`.

XPRS_msp_probdetach

Purpose

Detaches a problem from a MIP solution pool that was previously attached with a call to XPRS_msp_probattach.

Synopsis

```
int XPRS_CC XPRS_msp_probdetach(XPRSmipsolpool msp, XPRSprob prob)
```

Arguments

msp	The current MIP solution pool.
prob	Problem to be detached from the MIP solution pool.

Further information

The function will return successfully if the solution is not attached to the MIP solution pool.

Related topics

XPRS_msp_probattach.

XPRS_msp_readslxsol

Purpose

Reads one or more solutions from an ASCII solution file (`.slx`). Solution files can be created with a call to XPRS_msp_writeslxsol.

Synopsis

```
int XPRS_CC XPRS_msp_readslxsol(XPRSmipsolpool msp, XPRSnamelist col_name_list, const
       char * sFileName, const char *sFlags, int * const iSolutionId_Beg, int * const
       iSolutionId_End)
```

Arguments

`msp` The current MIP solution pool.

`col_name_list` An object used to map column names on to solution indices. May be NULL if not required.

`sFileName` Null terminated string containing the file name from which the solution(s) are to be read.

`sFlags` Flags for XPRS_msp_readslxsol. None currently.

`iSolutionId_Beg` Pointer to an integer returning the solution ID of the first solution successfully loaded from the file. May be NULL if not required.

`iSolutionId_End` Pointer to an integer returning the solution ID of the last solution successfully loaded from the file. May be NULL if not required.

Further information

1. The user can obtain a reference to the object mapping column names to indices for a problem by calling `XPRSgetnamelistobject` (p.210 of the Optimizer Reference Manual) with type 2. Passing this object down to XPRS_msp_readslxsol means that only solutions in the `.slx` file with column names contained in the mapping are loaded into the MIP solution pool. The loaded solutions will have the same column count as the mapping. The values for any columns that do not have an entry in the `.slx` file for a solution are set to zero.

2. If `col_name_list` is passed as NULL then all solutions are read into the MIP solution pool using the position of the solution value in the file section as the solution index.

Related topics

XPRS_msp_writeslxsol.

XPRS_msp_removecbmsghandler

Purpose

Removes an output callback function previously added by XPRS_msp_addcbmsghandler. The specified callback function will no longer be called after it has been removed.

Synopsis

```
int XPRS_CC XPRS_msp_removecbmsghandler(XPRSmipsolpool msp, int (XPRS_CC
       *f_msghandler)
       (XPRSobject vXPRSObject, void * vUserContext, void * vSystemThreadId, const
       char * sMsg, int iMsgType, int iMsgNumber), void* object);
```

Arguments

`msp` The current MIP solution pool.

`f_msghandler` The callback function to remove. If NULL then all output callback functions added with the given user-defined object value will be removed.

`object` The object value that the callback was added with. If NULL, then the object value will not be checked and all variable branching callbacks with the function pointer `f_chgbranch` will be removed.

Related topics

XPRS_msp_addcbmsghandler.

XPRS_msp_setcbmsghandler

Purpose

Declares an output callback function, called every time a line of message text is output by a MIP solution pool object.

Synopsis

```
int XPRS_CC XPRS_msp_setcbmsghandler(XPRSmipsolpool msp, int (XPRS_CC
       *f_msghandler)(XPRSobject vXPRSObject, void * vUserContext, void *
       vSystemThreadId, const char * sMsg, int iMsgType, int iMsgCode), void * p)
```

Arguments

`msp` The current MIP solution pool.

`f_msghandler` The callback function which takes six arguments, `vXPRSObject`, `vUserContext`, `vSystemThreadId`, `sMsg`, `iMsgType` and `iMsgCode`. Use a NULL value to cancel a callback function.

`vXPRSObject` A generic pointer to the msp object sending the message.

`vUserContext` The user-defined object passed to the callback function.

`vSystemThreadId` The system id of the thread sending the message caste to a void *.

`sMsg` A null terminated character array (string) containing the message, which may simply be a new line. When the callback is called for the first time sMsg will be a NULL pointer.

`iMsgType` Indicates the type of output message:

- `1` information messages;
- `2` (not used);
- `3` warning messages;
- `4` error messages.

A negative value means the callback is being called for the first time.

`iMsgCode` The code associated with the message. If the message is an error or a warning then you can look up the code in Chapter 11 of the Optimizer Reference Manual for advice on what it means and how to resolve the associated issue.

`p` A user-defined object to be passed to the callback function.

Further information

1. To send all messages to a log file the built in message handler `XPRSlogfilehandler` can be used. This can be done with:

   ```
   XPRS_msp_setcbmsghandler(msp, XPRSlogfilehandler, "log.txt");
   ```

2. The return value for this callback is ignored.

Related topics

None.

XPRS_msp_setdblcontrol

Purpose

Sets the value of a given double control parameter.

Synopsis

```
int XPRS_CC XPRS_msp_setdblcontrol(XPRSmipsolpool msp, const int iControlId, const
       double Val)
```

Arguments

`msp` The current MIP solution pool.

`iControlId` Id of control whose value is to be set. A full list of all available controls may be found in Chapter 3, or from the list in the `xprs.h` header file.

`Val` Value to which the control parameter is to be set.

Related topics

`XPRS_msp_getdblcontrol`, `XPRS_msp_setintcontrol`, `XPRS_msp_getintcontrol`.

XPRS_msp_setdblcontrolsol

Purpose

Sets the value of a given double control parameter.

Synopsis

```
int XPRS_CC XPRS_msp_setdblcontrolsol(XPRSmipsolpool msp, int iSolutionId, int *
       iSolutionIdStatus, const int iControlId, const double Val)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` The id of the solution for which the control is to be set.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`iControlId` Id of control whose value is to be set. A full list of all available controls may be found in Chapter 3, or from the list in the `xprs.h` header file.

`Val` Value to which the control parameter is to be set.

Related topics

`XPRS_msp_getdblcontrolsol`, `XPRS_msp_setintcontrolsol`, `XPRS_msp_getintcontrolsol`.

XPRS_msp_setintcontrol

Purpose

Sets the value of a given integer control parameter.

Synopsis

```
int XPRS_CC XPRS_msp_setintcontrol(XPRSmipsolpool msp, const int iControlId, const int
       Val)
```

Arguments

`msp` The current MIP solution pool.

`iControlId` Id of control whose value is to be set. A full list of all available controls may be found in Chapter 3, or from the list in the `xprs.h` header file.

`Val` Value to which the control parameter is to be set.

Related topics

`XPRS_msp_getintcontrol`, `XPRS_msp_setdblcontrol`, `XPRS_msp_getdblcontrol`.

XPRS_msp_setintcontrolsol

Purpose

Sets the value of a given integer control parameter.

Synopsis

```
int XPRS_CC XPRS_msp_setintcontrolsol(XPRSmipsolpool msp, int iSolutionId, int *
       iSolutionIdStatus, const int iControlId, const int Val)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` The id of the solution for which the control is to be set.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`iControlId` Id of control whose value is to be set. A full list of all available controls may be found in Chapter 3, or from the list in the `xprs.h` header file.

`Val` Value to which the control parameter is to be set.

Related topics

`XPRS_msp_getintcontrolsol`, `XPRS_msp_getdblcontrolsol`, `XPRS_msp_setdblcontrolsol`.

XPRS_msp_setsolname

Purpose

Changes the name of a solution stored in the MIP solution pool.

Synopsis

```
int XPRS_CC XPRS_msp_setsolname(XPRSmipsolpool msp, const int iSolutionId, const char
       *sNewSolutionBaseName, int * const gbNameModifiedForUniqueness, int * const
       iSolutionIdStatus)
```

Arguments

`msp` The current MIP solution pool.

`iSolutionId` Id of the solution whose name is to be changed.

`sNewSolutionBaseName` The new name to use to identify the solution. If the given name already exists and is used by another solution then the name to be stored will have characters appended to ensure the resulting name is unique among the stored solutions. If `sNewSolutionBaseName` is passed as NULL then a unique name will be generated automatically for the solution.

`bNameModifiedForUniqueness` Pointer to an integer returning 1 if it was necessary for the given solution name `sNewSolutionBaseName` to be modified to maintain uniqueness of solution names; 0 otherwise.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

Further information

The user will obtain the solution id `iSolutionId` from interaction with the MIP solution pool via functions such as `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme` and `XPRS_msp_getsollist`.

Related topics

`XPRS_msp_getsolname`, `XPRS_msp_findsolbyname`, `XPRS_msp_getdblattribprobextreme`, `XPRS_msp_getintattribprobextreme`, `XPRS_msp_getsollist`.

XPRS_msp_writeslxsol

Purpose

Creates an ASCII solution file (`.slx`) using a similar format to MPS files. The file can contain one or more solutions. These files can be read back into a MIP solution pool using the `XPRS_msp_readslxsol` function.

Synopsis

```
int XPRS_CC XPRS_msp_writeslxsol(XPRSmipsolpool msp, XPRSprob prob_context, int
       iSolutionId, int * iSolutionIdStatus, const char * sFileName, const char *
       sFlags)
```

Arguments

`msp` The current MIP solution pool.

`prob_context` Pointer to a problem. Can be NULL. See table below for interpretation in terms of function behaviour.

`iSolutionId` The id of a solution to write; otherwise a flag. See table below for interpretation in terms of function behaviour.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`sFileName` Null terminated string containing the file name to which the solution(s) is to be written.

`sFlags` Flags for `XPRS_msp_writeslxsol`:

- `p` use full precision for numerical values;
- `x` use hexadecimal format to write values.

Further information

This table defines the interpretation of the values of arguments `prob_context` and `iSolutionId` in terms of the behaviour of `XPRS_msp_writeslxsol`.

prob_context	*iSolutionId*	*Behaviour*
Non-NULL	0	Write out all feasible solutions to `prob_context`. Use the column names in `prob_context`.
Non-NULL	0	Write out a feasible solution to `prob_context` with the best objective (given the `OBJSENSE` (p.478 of the Optimizer Reference Manual) value of `prob_context`). Use the column names in `prob_context`.
Non-NULL	> 0	Write out the solution with ID `iSolutionId`. It is an error if the solution does not have the same number of columns as `prob_context`. Use the column names in `prob_context`.
NULL	≤ 0	Write out all solutions. Use automatically generated column names.
NULL	> 0	Write out the solution with ID `iSolutionId`. Use automatically generated column names.

Related topics

`XPRS_msp_readslxsol`.

CHAPTER 3

MSP Control Parameters

MSP_DEFAULTUSERSOLFEASTOL

Description This is the zero tolerance on the value of integer, semi-continuous, partial and semi-continuos integer variables and SOS variables. If one of these is less than or equal to FEASTOL (p.402 of the Optimizer Reference Manual) in absolute value, it is treated as zero.

Type Double

Default value 1.0E-06

MSP_DEFAULTUSERSOLMIPTOL

Description This is the tolerance within which a decision variable's value is considered to be integral. It is the default value for the tolerance `MSP_SOL_MIPTOL` when a solution is added by the user via routines `XPRS_msp_loadsol` and `XPRS_msp_readslxsol`.

Type Double

Default value 5.0E-06

MSP_DUPLICATESOLUTIONSPOLICY

Description Policy to use when handling storage of duplicate solutions.

Type Integer

Values

0	Keep all: All solutions are kept including duplicates.
1	Continuous: All variables are compared with an exact match. Duplicate solutions are discarded.
2	Discrete and continuous separate: Both the discrete component of a solution pair and the continuous solution variables are compared. The continuous variables are compared with an exact match. Duplicate solutions are discarded.
3	Discrete only: Only the discrete component of a solution pair is compared. Duplicate solutions are discarded.

Default value 3

MSP_INCLUDEPROBNAMEINLOGGING

Description Controls whether or not the MIP solution pool logging uses the problem name when a message references a problem.

Type Integer

Values

0	The problem name is not included.
1	The problem name is included.

Default value 1

MSP_SOL_BITFIELDSUSR

Description A user-definable bit map to be stored with a solution. This bit map can be used to define subsets of solutions returned by the `XPRS_msp_getsollist2` function.

Type Integer

Default value 0

Affects routines XPRS_msp_getintcontrolsol, XPRS_msp_setintcontrolsol.

See also XPRS_msp_getsollist2.

MSP_SOL_FEASTOL

Description This is the zero tolerance on the value of integer, semi-continuous, partial and semi-continuos integer variables and SOS variables. If one of these is less than or equal to FEASTOL (p.402 of the Optimizer Reference Manual) in absolute value, it is treated as zero.

Type Double

Default value 1.0E-06

MSP_SOL_MIPTOL

Description This is the tolerance within which a decision variable's value is considered to be integral.

Type Double

Default value 5.0E-06

CHAPTER 4

MSP Attributes

MSP_PRB_FEASIBLESOLS Number of feasible solutions for the problem. p. 56

MSP_PRB_VALIDSOLS Number of solutions with the same number of columns as the problem. p. 57

MSP_SOL_BITFIELDSSYS A bit map with fields containing the logical attribute values for the solution e.g., MSP_SOL_ISREPROCESSEDUSERSOLUTION and MSP_SOL_ISUSERSOLUTION. p. 65

MSP_SOL_COLS Number of columns in the solution. p. 65

MSP_SOL_ISREPROCESSEDUSERSOLUTION Whether or not the solution arises from a user solution (i.e., passed in by the user via XPRS_msp_loadsol or XPRS_msp_readslxsol) being loaded into an attached problem and the resulting "cleaned up" solution being loaded back into the MIP solution pool. p. 65

MSP_SOL_ISUSERSOLUTION Whether or not the solution was passed in by the user (via XPRS_msp_loadsol or XPRS_msp_readslxsol) or whether the solution was loaded internally from an attached problem. p. 65

MSP_SOL_NONZEROS Number of non-zeros in the solution. p. 65

MSP_SOLPRB_INFCNT_BIN Number of binary variable MIP infeasibilities for the solution with respect to the problem. p. 57

MSP_SOLPRB_INFCNT_COLUMN Number of column bound infeasibilities for the solution with respect to the problem. p. 57

MSP_SOLPRB_INFCNT_DELAYEDROW Number of delayed row infeasibilities for the solution with respect to the problem. p. 57

MSP_SOLPRB_INFCNT_INT Number of integer variable MIP infeasibilities for the solution with respect to the problem. p. 57

MSP_SOLPRB_INFCNT_MIP Number of MIP infeasibilities that the solution has with respect to the problem. p. 57

MSP_SOLPRB_INFCNT_PI Number of partial integer variable MIP infeasibilities for the solution with respect to the problem. p. 58

MSP_SOLPRB_INFCNT_PRIMAL Number of primal infeasibilities that the solution has with respect to the problem. This count includes the column bound infeasibilities and both the row and delayed row infeasibilities. p. 58

MSP_SOLPRB_INFCNT_SC Number of semi-continuous variable MIP infeasibilities for the solution with respect to the problem. p. 58

MSP_SOLPRB_INFMXI_SET2 Index of a set that has the maximum SOS2 set MIP infeasibility. p. 62

MSP_SOLPRB_INFMXI_SI Index of a column achieving the maximum semi-continuous integer variable MIP infeasibility for the solution with respect to the problem. p. 62

MSP_SOLPRB_INFMXI_SLACK Index of a row achieving the maximum row infeasibility for the solution with respect to the problem. p. 62

MSP_SOLPRB_INFSUM_BIN Sum of binary variable MIP infeasibilities for the solution with respect to the problem. p. 62

MSP_SOLPRB_INFSUM_COLUMN Sum of column bound infeasibilities for the solution with respect to the problem. p. 62

MSP_SOLPRB_INFSUM_DELAYEDROW Sum of delayed row infeasibilities for the solution with respect to the problem. p. 63

MSP_SOLPRB_INFSUM_INT Sum of integer variable MIP infeasibilities for the solution with respect to the problem. p. 63

MSP_SOLPRB_INFSUM_MIP Sum of MIP infeasibilities that the solution has with respect to the problem. p. 63

MSP_SOLPRB_INFSUM_PI Sum of partial integer variable MIP infeasibilities for the solution with respect to the problem. p. 63

MSP_SOLPRB_INFSUM_PRIMAL Sum of primal infeasibilities that the solution has with respect to the problem. This sum includes the column bound infeasibilities and both the row and delayed row infeasibilities. p. 63

MSP_SOLPRB_INFSUM_SC Sum of semi-continuous variable MIP infeasibilities for the solution with respect to the problem. p. 63

MSP_SOLPRB_INFSUM_SET1 Sum of SOS1 set MIP infeasibilities for the solution with respect to the problem. p. 64

MSP_SOLPRB_INFSUM_SET2 Sum of SOS2 set MIP infeasibilities for the solution with respect to the problem. p. 64

MSP_SOLPRB_INFSUM_SI Sum of semi-continuous integer variable MIP infeasibilities for the solution with respect to the problem. p. 64

MSP_SOLPRB_INFSUM_SLACK Sum of row infeasibilities for the solution with respect to the problem. p. 64

MSP_SOLPRB_OBJ Objective value of the solution with the problem. p. 64

MSP_SOLUTIONS The number of solutions stored by the MIP solution pool. p. 64

MSP_PRB_FEASIBLESOLS

Description Number of feasible solutions for the problem.

Type Integer

MSP_PRB_VALIDSOLS

Description Number of solutions with the same number of columns as the problem.

Type Integer

MSP_SOLPRB_INFCNT_BIN

Description Number of binary variable MIP infeasibilities for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_COLUMN

Description Number of column bound infeasibilities for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_DELAYEDROW

Description Number of delayed row infeasibilities for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_INT

Description Number of integer variable MIP infeasibilities for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_MIP

Description Number of MIP infeasibilities that the solution has with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_PI

Description Number of partial integer variable MIP infeasibilities for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_PRIMAL

Description Number of primal infeasibilities that the solution has with respect to the problem. This count includes the column bound infeasibilities and both the row and delayed row infeasibilities.

Type Integer

MSP_SOLPRB_INFCNT_SC

Description Number of semi-continuous variable MIP infeasibilities for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_SET1

Description Number of SOS1 set MIP infeasibilities for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_SET2

Description Number of SOS2 set MIP infeasibilities for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_SI

Description Number of semi-continuous integer variable MIP infeasibilities for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFCNT_SLACK

Description Number of rows of the problem that are violated for the solution.

Type Integer

MSP_SOLPRB_INFEASCOUNT

Description The sum of the MIP and primal infeasibility counts for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFMAX_BIN

Description Maximum binary variable MIP infeasibility for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFMAX_COLUMN

Description Maximum column bound infeasibility for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFMAX_DELAYEDROW

Description Maximum delayed row infeasibility for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFMAX_INT

Description Maximum integer variable MIP infeasibility for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFMAX_PI

Description Maximum partial integer variable MIP infeasibility for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFMAX_SC

Description Maximum semi-continuous variable MIP infeasibility for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFMAX_SET1

Description Maximum variable value causing a SOS1 set MIP infeasibility.

Type Double

MSP_SOLPRB_INFMAX_SET2

Description Maximum variable value causing a SOS2 set MIP infeasibility.

Type Double

MSP_SOLPRB_INFMAX_SI

Description Maximum semi-continuous integer variable MIP infeasibility for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFMAX_SLACK

Description Maximum row infeasibility for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFMXI_BIN

Description Index of a column achieving the maximum binary variable MIP infeasibility for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFMXI_COLUMN

Description Index of a column achieving the maximum column bound infeasibility for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFMXI_DELAYEDROW

Description Index of a delayed row achieving the maximum delayed row infeasibility for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFMXI_INT

Description Index of a column achieving the maximum integer variable MIP infeasibility for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFMXI_PI

Description Index of a column achieving the maximum partial integer variable MIP infeasibility for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFMXI_SC

Description Index of a column achieving the maximum semi-continuous variable MIP infeasibility for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFMXI_SET1

Description Index of a set that has the maximum SOS1 set MIP infeasibility.

Type Integer

MSP_SOLPRB_INFMXI_SET2

Description Index of a set that has the maximum SOS2 set MIP infeasibility.

Type Integer

MSP_SOLPRB_INFMXI_SI

Description Index of a column achieving the maximum semi-continuous integer variable MIP infeasibility for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFMXI_SLACK

Description Index of a row achieving the maximum row infeasibility for the solution with respect to the problem.

Type Integer

MSP_SOLPRB_INFSUM_BIN

Description Sum of binary variable MIP infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_COLUMN

Description Sum of column bound infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_DELAYEDROW

Description Sum of delayed row infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_INT

Description Sum of integer variable MIP infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_MIP

Description Sum of MIP infeasibilities that the solution has with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_PI

Description Sum of partial integer variable MIP infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_PRIMAL

Description Sum of primal infeasibilities that the solution has with respect to the problem. This sum includes the column bound infeasibilities and both the row and delayed row infeasibilities.

Type Double

MSP_SOLPRB_INFSUM_SC

Description Sum of semi-continuous variable MIP infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_SET1

Description Sum of SOS1 set MIP infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_SET2

Description Sum of SOS2 set MIP infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_SI

Description Sum of semi-continuous integer variable MIP infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_INFSUM_SLACK

Description Sum of row infeasibilities for the solution with respect to the problem.

Type Double

MSP_SOLPRB_OBJ

Description Objective value of the solution with the problem.

Type Double

Note For infeasible solutions, the objective value returned is infinity. Calling `XPRSfixglobals` (p.168 of the Optimizer Reference Manual) will invalidate a solution in the pool if any of the global variables in the solution are rounded and if this happens the objective will be returned as infinity.

MSP_SOLUTIONS

Description The number of solutions stored by the MIP solution pool.

Type Integer

MSP_SOL_BITFIELDSSYS

Description A bit map with fields containing the logical attribute values for the solution e.g., MSP_SOL_ISREPROCESSEDUSERSOLUTION and MSP_SOL_ISUSERSOLUTION.

Type Integer

MSP_SOL_COLS

Description Number of columns in the solution.

Type Integer

MSP_SOL_ISREPROCESSEDUSERSOLUTION

Description Whether or not the solution arises from a user solution (i.e., passed in by the user via XPRS_msp_loadsol or XPRS_msp_readslxsol) being loaded into an attached problem and the resulting "cleaned up" solution being loaded back into the MIP solution pool.

Type Integer

Values

0	The solution was not derived from a user solution i.e., it is a user solution or it is a solution found by an attached XPRSprob problem.
1	The solution was derived from a user solution.

MSP_SOL_ISUSERSOLUTION

Description Whether or not the solution was passed in by the user (via XPRS_msp_loadsol or XPRS_msp_readslxsol) or whether the solution was loaded internally from an attached problem.

Type Integer

Values

0	The solution was loaded internally from an attached problem.
1	The solution was loaded by the user via XPRS_msp_loadsol or XPRS_msp_readslxsol.

MSP_SOL_NONZEROS

Description Number of non-zeros in the solution.

Type Integer

II. MIP Solution Enumerator

CHAPTER 5

Introduction

5.1 Overview

The MIP solution enumerator (`XPRSmipsolenum`) runs a customized MIP search on a user provided problem (`XPRSprob`). The search is customized such that nodes are not cut-off by bounding and integer solution nodes are branched. The MIP solution enumerator marshals the solutions found to storage in a user provided MIP solution pool (`XPRSmipsolpool`).

In general the MIP solution enumerator is useful for generating a set of solutions for a problem. In particular, the MIP solution enumerator can be used to generate the n-best solutions to a problem. The user will typically be interested in such a set when there are constraints and/or costs not reflected in the problem and the user wants a set of solutions from which a final solution can be selected that best meets the external constraints and objective.

5.2 Applications: N-Best Solutions Example

A typical use of the MIP solution enumerator is to generate the set of n-best solutions. The following example shows how this can be done:-

```
#include <stdio.h>
#include "xprs.h"
#include "xprs_mse_defaulthandler.h"

int main(int argc, char **argv)
{
  XPRSprob prob;
  XPRSmipsolpool msp;
  XPRSmipsolenum mse;
  int nMaxSols;
  int i, j, nSols, nCols, iSolutionId, iSolutionIdStatus;
  double dObj, dSol;
  int iPresolveOps;
  const char *sProblem = argv[1];

  XPRSinit(NULL);
  XPRScreateprob(&prob);
  XPRSreadprob(prob, sProblem, "");

  XPRS_msp_create(&msp);

  XPRS_mse_create(&mse);

  XPRSgetintattrib(prob, XPRS_COLS, &nCols);

  /* Disable heuristics to avoid duplicate solutions being stored */
  XPRSsetintcontrol(prob, XPRS_HEURSTRATEGY, 0);
```

```
  XPRSgetintcontrol(prob, XPRS_PRESOLVEOPS, &iPresolveOps);
  iPresolveOps &= ~(1 << 3); /* Disable dual reduction operations */
  iPresolveOps &= ~(1 << 5); /* Disable duplicate column removal  */
  XPRSsetintcontrol(prob, XPRS_PRESOLVEOPS, iPresolveOps);

  /* Run the enumeration */
  nMaxSols = 10;
  XPRS_mse_maxim(mse, prob, msp, XPRS_mse_defaulthandler, NULL, &nMaxSols);

  /* Print out the solutions found */
  XPRS_mse_getintattrib(mse, XPRS_MSE_SOLUTIONS, &nSols);
  for(i = 1; i <= nSols; i++) {
    XPRS_mse_getsollist(mse, XPRS_MSE_METRIC_MIPOBJECT, i, i, &iSolutionId, NULL, NULL);
    XPRS_mse_getsolmetric(mse, iSolutionId, &iSolutionIdStatus, XPRS_MSE_METRIC_MIPOBJECT, &dObj);
    printf("--------\n%3i = %12.5f\n", i, dObj);
    for(j = 0; j < nCols; j++) {
      XPRS_msp_getsol(msp, iSolutionId, &iSolutionIdStatus, &dSol, j, j, NULL);
      printf("%3i = %12.5f\n", j, dSol);
    }
  }

  XPRS_msp_destroy(msp);
  XPRSdestroyprob(prob);
  XPRS_mse_destroy(mse);

  XPRSfree();
}
```

In the example we use the XPRS_mse_defaulthandler function to manage the storage of at most nMaxSols (= 10) solutions. The code for the XPRS_mse_defaulthandler callback is provided in the FICO Xpress Optimizer. Using the default control settings when a solution is found and there are nMaxSols (= 10) solutions stored the XPRS_mse_defaulthandler callback will either reject the current solution or delete the worst existing solution depending on their objective values (when there is a tie the current solution is rejected). In this way the MIP solution enumerator will generate the set of n-best solutions.

It is important to note that duplicate solutions can potentially be stored in the cases where the MIP solution pool contains initial solutions prior to the enumeration run or if heuristics are activated in the XPRSprob. In the example we have disabled heuristics and the MIP solution pool does not contain any solutions prior to the enumeration run so there is no need to check for duplicates. The management of duplicate solutions is left to the MIP solution pool functionality which is discussed in Duplicate Solutions.

It is also important to note that it is sometimes necessary to disable certain FICO Xpress Optimizer presolve operations (i.e., setting the integer control PRESOLVEOPS in the example) to avoid spurious deletion of solutions from the enumeration search space. This is discussed in the following section Presolve considerations.

In the event that initial solutions are contained in the MIP solution pool prior to the enumeration run the MIP solution enumerator will load the set of feasible solutions (with the same number of columns as the XPRSprob) from the initial set of solutions as if they were found during the MIP search on the XPRSprob. Note that new versions of these solutions are created and the original versions are deleted from the MIP solution pool.

Once the enumeration run terminates the example demonstrates simple retrieval of the solution information. Note that the solution values are stored in the MIP solution pool and the metric information for the solutions is stored in the MIP solution enumerator.

5.3 Presolve considerations

The default control settings of the FICO Xpress Optimizer presolver assume that the user is

satisfied if the MIP search finds a single solution with the best objective. The choice of the default presolve settings attempts to improve the efficiency of this type of search by cutting off MIP solutions from the feasible region that are either degenerate (i.e., that have equivalent representations with different feasible values of the variables) or dominated (i.e., that can be deduced to be worse than solutions contained in the remaining feasible region) or symmetric to another solution. The user must take care to disable these default presolve operations when solutions may be removed from the enumeration search space that are of interest. Also, the user should be aware that disabling these presolve operations can significantly increase the enumeration runtime.

Presolve operations removing dominated MIP solutions are collectively referred to as dual reduction operations. To disable duplicate column detection the user must unset bit 5 of the integer control PRESOLVEOPS. To disable all dual reduction operations the user must unset bit 3 of PRESOLVEOPS. See the Applications: N-Best Solutions Example for an example of how to unset these bits.

To disable FICO Xpress Optimizer from removing symmetric solutions, set the SYMMETRY control to 0.

5.4 Basic customization

By customizing the controls for the XPRS_mse_defaulthandler callback (i.e., setting MSE_CALLBACKCULLSOLS_MIPOBJECT and/or MSE_CALLBACKCULLSOLS_DIVERSITY) the user can modify the behavior of the callback when the number of solutions exceeds the maximum. The user can delete p solutions based on the MIP objective values and then delete q of the remaining nMaxSols – p solutions based on the diversity metric (see below). The current solution will be rejected if it is no better than any of the solutions that were deleted. In this way the user has simple control of how the solution set is generated.

In addition to the MIP objective the MIP solution enumerator provides a metric for solutions based on the 'diversity' of the solution with respect to the other stored solutions. The diversity metric for a solution is the sum of difference metrics between the solution and the other stored solutions. By default the difference value between two solutions is calculated by the MIP solution enumerator using a simple difference metric that considers only the MIP entities in the solution. Note that for the MIP solution enumerator to calculate the diversity metrics a problem needs to be attached to the MIP solution pool (see 'Duplicate solutions'). This is because the global structure of the solution variables is required for the difference metric calculation. The user will typically attach to the MIP solution pool the XPRSprob to be used to run the enumeration. Finally, note that the user can provide their own difference metric calculation for solution pairs in a callback defined with a call to XPRS_mse_setcbgetsolutiondiff (in this case there does not need to be a problem attached to the MIP solution pool).

5.5 Advanced customization

By providing a customized version of the XPRS_mse_defaulthandler the user can define a modified objective value for solutions. Typically this value reflects the 'true' objective of the solution in cases where the MIP objective does not completely model the user's problem. The modified objective value is stored with the solution and can be used to obtain a ranked order of solutions in the case where the user wishes to delete solutions based on this metric (see XPRS_mse_getcullchoice). Also with a customized callback the user is able to arbitrarily accept or reject the current solution. Typically the user will want to do this in cases where the constraints and/or objective of the MIP problem do not completely model the user's problem.

From the customized XPRS_mse_defaulthandler callback the user is able to set a return value that

causes the MIP solution enumerator to set the cut-off for the search to the worst MIP objective of the active solutions. This can be useful to improve the run time of the enumeration. Note that the default `n-best` search uses this to improve the enumeration performance.

5.6 Data Model

The following UML diagram outlines the relationships the `XPRSmipsolenum` has with some other data entities.

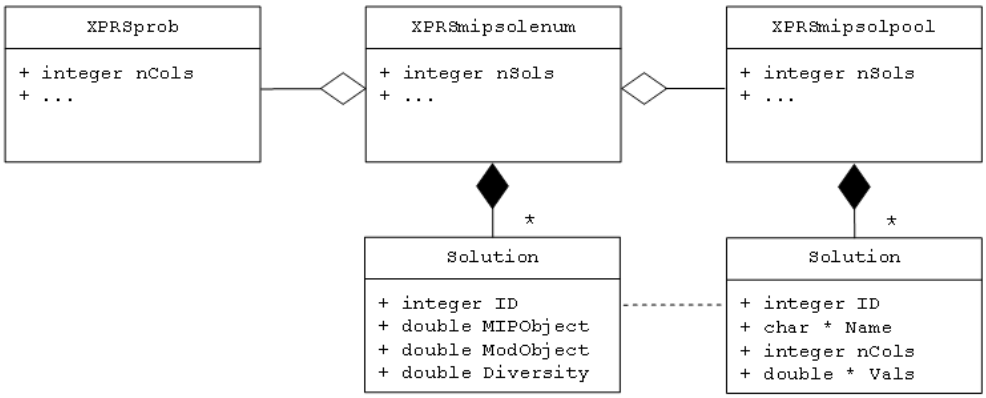

Figure 5.1:

Starting from the top left the diagram shows the `XPRSprob` referenced by the `XPRSmipsolenum` in the top middle. The `XPRSmipsolenum` references the `XPRSprob` only for the duration of the enumeration-run routines `XPRS_mse_minim`, `XPRS_mse_maxim` and `XPRS_mse_opt`. The `XPRSmipsolenum` sets certain controls on the `XPRSprob`, registers an internal callback to receive solutions found during the enumeration and calls the relevant optimization function to execute the enumeration.

Moving to the top right the diagram shows the `XPRSmipsolpool` referenced by the `XPRSmipsolenum`. The `XPRSmipsolenum` references the `XPRSmipsolpool` only for the duration of the enumeration-run routines (in the same way as the `XPRSprob`).

While running the enumeration the `XPRSmipsolenum` maintains storage of additional information for each solution found that cannot be stored by the `XPRSmipsolpool`. The additional information is the MIP objective value (`MIPObject`) of the solution, any user-defined objective function value (`ModObject`) for the solution and a diversity-metric (`Diversity`) for the solution that measures how different the solution is with respect to the other solutions in storage. As indicated by the dashed line in the diagram, the solution ID is used to link the information stored separately in the `XPRSmipsolenum` and the `XPRSmipsolpool`.

Since a parallel storage of solution information is required the `XPRSmipsolenum` marshals the storage of solutions to the `XPRSmipsolpool`. To do this the `XPRSmipsolenum` registers an internal callback with the `XPRSmipsolpool` to track when solutions are deleted by the user from the `XPRSmipsolpool`. It is also necessary that the `XPRSmipsolenum` loads the solutions found during the enumeration directly into the `XPRSmipsolpool` overriding the automatic loading of solutions from the `XPRSprob` to `XPRSmipsolpool` in the situation where the user has attached the `XPRSprob` to the `XPRSmipsolpool` prior to the enumeration run.

Once the enumeration run terminates the MIP solutions are stored in the `XPRSmipsolpool` and the additional information for the solutions is stored in the `XPRSmipsolenum`. At this point the solution information is linked only by the solution ID.

CHAPTER 6

MSE Functions

XPRS_mse_addcbmsghandler

Purpose

Declares an output callback function, called every time a line of message text is output by the MIP solution enumerator. This callback function will be called in addition to any output callbacks already added by XPRS_mse_addcbmsghandler.

Synopsis

```
int XPRS_CC XPRS_mse_addcbmsghandler(XPRSmipsolenum mse, int (XPRS_CC *f_msghandler)
       (XPRSobject vXPRSObject, void * vUserContext, void * vSystemThreadId, const
       char * sMsg, int iMsgType, int iMsgNumber), void * p, int priority);
```

Arguments

mse The current MIP solution enumerator

f_msghandler The callback function which takes six arguments, vXPRSObject, vUserContext, vSystemThreadId, sMsg, iMsgType and iMsgNumber. Use a NULL value to cancel a callback function.

vXPRSObject The object sending the message. Use XPRSgetobjecttypename (p.213 of the Optimizer Reference Manual) to get the name of the object type.

vUserContext The user-defined object passed to the callback function.

vSystemThreadId The system id of the thread sending the message caste to a void *.

sMsg A null terminated character array (string) containing the message, which may simply be a new line. When the callback is called for the first time sMsg will be a NULL pointer.

iMsgType Indicates the type of output message:

1 information messages;
2 (not used);
3 warning messages;
4 error messages.

A negative value means the callback is being called for the first time.

iMsgNumber The number associated with the message. If the message is an error or a warning then you can look up the number in the section Optimizer Error and Warning Messages for advice on what it means and how to resolve the associated issue.

p A user-defined object to be passed to the callback function.

priority An integer that determines the order in which multiple output callbacks will be invoked. The callback added with a higher priority will be called before a callback with a lower priority. Set to 0 if not required.

Further information

To send messages to a log file the built in message handler XPRSlogfilehandler can be used. This can be done with:

```
XPRS_mse_addcbmsghandler(msp, XPRSlogfilehandler, "log.txt");
```

Related topics

XPRS_mse_removecbmsghandler,XPRSgetobjecttypename (p.213 of the Optimizer Reference Manual).

XPRS_mse_create

Purpose

Sets up a new MIP solution enumerator object.

Synopsis

```
int XPRS_CC XPRS_mse_create(XPRSmipsolenum * mse)
```

Argument

`mse` Pointer to a variable holding the new MIP solution enumerator.

Further information

1. Calls to `XPRS_mse_create` must be made after the call to `XPRSinit` (p.252 of the Optimizer Reference Manual).
2. All MIP solution enumerators created using a call to `XPRS_mse_create` should be disposed of with a call to `XPRS_mse_destroy`.

Related topics

`XPRSinit` (p.252 of the Optimizer Reference Manual), `XPRS_mse_destroy`.

XPRS_mse_defaulthandler

Purpose

A routine defined in xprs_mse_defaulthandler.h intended only to be passed as a callback into the enumeration-run routines XPRS_mse_minim, XPRS_mse_maxim and XPRS_mse_opt. When used to run an enumeration this routine is called each time a solution is found in the enumeration. This routine provides simple functionality to manage common strategies for keeping a 'good' set of solutions found during enumeration and controlling the enumeration run e.g., n-best solutions. The user may wish to base any customized version of the callback on the contents of this routine.

Synopsis

```
int XPRS_CC XPRS_mse_defaulthandler(XPRSmipsolenum mse, XPRSprob prob, XPRSmipsolpool
       msp, void *ctx, int *const nMaxSols, const double *const x, const int nCols,
       const double dMipObject, double *const dModifiedObject, int *const bRejectSoln,
       int *const bUpdateMipAbsCutOffOnCurrentSet)
```

Arguments

`mse` The current MIP solution enumerator.

`prob` The problem used to enumerate solutions.

`msp` The MIP solution pool used to store the solutions found during enumeration.

`ctx` The user-defined object passed into the enumeration-run routine that started this run (i.e., XPRS_mse_minim, XPRS_mse_maxim or XPRS_mse_opt).

`nMaxSols` The integer pointer passed into the enumeration-run routine that started this run (i.e., XPRS_mse_minim, XPRS_mse_maxim or XPRS_mse_opt). Note that this pointer is only used by the callback and is intended to store the maximum number of solutions the user is prepared to capture during the enumeration.

`x` A dense solution array of doubles containing the `nCols` solution values of the new solution.

`nCols` The number elements in `x`. This is the same as the column dimension of the problem when the enumeration was started.

`dMipObject` A double with the value of the MIP objective of the solution.

`dModifiedObject` A pointer to a double that contains, on entry to the routine, the MIP objective. The user can assign it a new value that reflects the 'true' objective of the solution in cases where the MIP objective does not completely model the user's problem. The returned value is stored as the MSE_METRIC_MODOBJECT metric of the solution.

`bRejectSoln` A pointer to an integer that is at value zero on entry to the routine. The solution will not be stored if the user returns this variable at a non-zero value.

`bUpdateMipAbsCutOffOnCurrentSet` A pointer to an integer that is at value zero on entry to the routine. If the user returns this variable with a non-zero value then the MIP solution enumerator will update the MIPABSCUTOFF (p.422 of the Optimizer Reference Manual) of the problem to reflect the worst MIP objective of the stored solutions (including the new solution if it is not rejected).

Further information

This simple routine uses only four different API function calls. These calls are to the MIP solution enumerator routines XPRS_mse_getcullchoice, XPRS_mse_getintattrib and XPRS_mse_getintcontrol and the MIP solution pool routine XPRS_msp_delsol.

Related topics

XPRS_mse_minim, XPRS_mse_maxim, XPRS_mse_opt, XPRS_mse_getcullchoice, XPRS_mse_getintattrib, XPRS_mse_getintcontrol, XPRS_msp_delsol.

XPRS_mse_destroy

Purpose

Destroys a MIP solution enumerator object and its resources. The object will generally be created by a call to the function XPRS_mse_create.

Synopsis

```
int XPRS_CC XPRS_mse_destroy(XPRSmipsolenum mse)
```

Argument

`mse` MIP solution enumerator to be destroyed.

Related topics

XPRS_mse_create.

XPRS_mse_getcbmsghandler

Purpose

Get the output callback function, as set by XPRS_mse_setcbmsghandler.

Synopsis

```
int XPRS_CC XPRS_mse_getcbmsghandler(XPRSmipsolenum mse, int (XPRS_CC
       **r_f_msghandler)
       (XPRSobject vXPRSObject, void * vUserContext, void * vSystemThreadId, const
       char * sMsg, int iMsgType, int iMsgNumber), void **object);
```

Arguments

mse	The current MIP solution enumerator
r_f_msghandler	Pointer to the memory where the callback function will be returned.

Related topics

XPRS_mse_setcbmsghandler.

XPRS_mse_getcullchoice

Purpose

Generates a list of solution ids for solutions recommended to be dropped assuming that the given number of solutions are required to be dropped and the solutions are to be compared with each other with respect to the given metric. This function also decides whether a given, new solution defined by its metric value (or its solution values) should also be dropped given that the returned list of solutions are recommended to be dropped.

Synopsis

```
int XPRS_CC XPRS_mse_getcullchoice(XPRSmipsolenum mse, int iMetricId, int
       cull_sol_id_list[], int nMaxSolsToCull, int * nSolsToCull, double
       dNewSolMetric, const double x[], int nCols, int * bRejectSoln)
```

Arguments

`mse` The current MIP solution enumerator.

`iMetricId` Id of metric for which the solutions should be compared e.g., the MIP objective value is used when `MSE_METRIC_MIPOBJECT` is passed down. A full list of all available metrics may be found in Chapter 8, or from the list in the `xprs.h` header file.

`cull_sol_id_list` Integer array where the solution ids will be returned. May be NULL if not required.

`nMaxSolsToCull` The maximum number of solutions required to be returned in `cull_sol_id_list`.

`nSolsToCull` Pointer to an integer where the number of solution ids that were written to `cull_sol_id_list` is returned. May be NULL if not required.

`dNewSolMetric` The metric value for the new solution for the metric identified by iMetricId. It is assumed that the user wishes to decide whether or not to keep the new solution given that solutions in the returned list `cull_sol_id_list` are recommended to be dropped.

`x` The solution values of the new solution. The diversity metric (`MSE_METRIC_DIVERSITY`) requires the solution values in order to calculate the new solution's metric with respect to the currently stored solutions. Other metrics simply use `dNewSolMetric` and do not require the solution values. May be NULL if not required.

`nCols` The number of columns in the solution array x. This must be the same as the number of (initial) columns in the enumeration problem.

`bRejectSoln` If this is passed down as NULL or if its value is non-zero then the given new solution is ignored.

Further information

1. If `cull_sol_id_list` is NULL or `nMaxSolsToCull` is zero or `nSolsToCull` is NULL (i.e., a list of solution ids cannot be returned) then the routine simply decides whether or not to reject the new solution based on the assumption that no currently existing solutions are to be dropped.

2. The user can drop solutions by calling `XPRS_msp_delsol` on the MIP solution pool being used by the enumerator.

3. To call the routine ignoring the new solution the user can pass the argument `bRejectSoln` as NULL. Note that if the new solution is not to be ignored then the user must pass `bRejectSoln` as non-NULL with an initial value of zero.

Related topics

`XPRS_msp_delsol`.

XPRS_mse_getdblattrib

Purpose

Provides read access to the values of double attributes associated with the MIP solution enumerator.

Synopsis

```
int XPRS_CC XPRS_mse_getdblattrib(XPRSmipsolenum mse, int iAttribId, double * Val)
```

Arguments

`mse` The current MIP solution enumerator.

`iAttribId` Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 8, or from the list in the `xprs.h` header file.

`Val` Pointer to an double where the value of the attribute will be returned.

Related topics

`XPRS_mse_getintattrib`.

XPRS_mse_getdblcontrol

Purpose

Retrieves the value of a given double control parameter.

Synopsis

```
int XPRS_CC XPRS_mse_getdblcontrol(XPRSmipsolenum mse, int iControlId, double * Val)
```

Arguments

`mse` The current MIP solution enumerator.

`iControlId` Id of control whose value is to be returned. A full list of all available controls may be found in Chapter 7, or from the list in the `xprs.h` header file.

`Val` Pointer to a double where the value of the control will be returned.

Related topics

`XPRS_mse_getintcontrol`, `XPRS_mse_setdblcontrol`, `XPRS_mse_setintcontrol`.

XPRS_mse_getintattrib

Purpose

Provides read access to the values of integer attributes associated with the MIP solution enumerator.

Synopsis

```
int XPRS_CC XPRS_mse_getintattrib(XPRSmipsolenum mse, int iAttribId, int * Val)
```

Arguments

`mse` The current MIP solution enumerator.

`iAttribId` Id of attribute whose value is to be returned. A full list of all available attributes may be found in Chapter 8, or from the list in the `xprs.h` header file.

`Val` Pointer to an integer where the value of the attribute will be returned.

Related topics

`XPRS_mse_getdblattrib`.

XPRS_mse_getintcontrol

Purpose

Retrieves the value of a given integer control parameter.

Synopsis

```
int XPRS_CC XPRS_mse_getintcontrol(XPRSmipsolenum mse, int iControlId, int * Val)
```

Arguments

`mse` The current MIP solution enumerator.

`iControlId` Id of control whose value is to be returned. A full list of all available controls may be found in Chapter 7, or from the list in the `xprs.h` header file.

`Val` Pointer to a integer where the value of the control will be returned.

Related topics

`XPRS_mse_getdblcontrol`, `XPRS_mse_setdblcontrol`, `XPRS_mse_setintcontrol`.

XPRS_mse_getlasterror

Purpose

Gets the last error message.

Synopsis

```
int XPRS_CC XPRS_mse_getlasterror(XPRSmipsolenum mse, int * iMsgNumber, char * msg,
       int iStringBufferBytes, int * iBytesInInternalString)
```

Arguments

`mse` The current MIP solution enumerator.

`iMsgNumber` A pointer to an integer to return the number of the last error message. Can be NULL if not required. Refer to Chapter 11 of the Optimizer Reference Manual for a list of possible error numbers, the errors and warnings that they indicate, and advice on what they mean and how to resolve them.

`msg` A character buffer of length at least `iStringBufferBytes` to return the error message. Can be NULL if not required.

`iStringBufferBytes` The length of the `msg` buffer.

`iBytesInInternalString` A pointer to an integer to return the number of bytes required to store the error message. Can be NULL if not required.

Related topics

Chapter 11 of the Optimizer Reference Manual , `XPRS_mse_setcbmsghandler`.

XPRS_mse_getsolbasename

Purpose

Gets the name currently used as a prefix for solutions found during the enumeration run.

Synopsis

```
int XPRS_CC XPRS_mse_getsolbasename(XPRSmipsolenum mse, char * sname, int
       iStringBufferBytes, int * iBytesInInternalString)
```

Arguments

`mse` The current MIP solution enumerator.

`sname` A character buffer of length at least `iStringBufferBytes` to return the name. Can be NULL if not required.

`iStringBufferBytes` The length of the `sname` buffer.

`iBytesInInternalString` A pointer to an integer to return the number of bytes required to store the name. Can be NULL if not required.

Related topics

`XPRS_mse_setsolbasename`.

XPRS_mse_getsollist

Purpose

Returns a list of solution ids of solutions found during the enumeration. The list is sorted by the value of some attribute of the solutions e.g., the MIP objective value.

Synopsis

```
int XPRS_CC XPRS_mse_getsollist(XPRSmipsolenum mse, int iMetricId, int
       iRankFirstIndex, int iRankLastIndex, int iSolutionIds[], int * nReturnedSolIds,
       int * nSols)
```

Arguments

`mse` The current MIP solution enumerator.

`iMetricId` Id of the attribute whose value is used to rank the solution ids returned in the iSolutionIds array e.g., the MIP objective value is used when `MSE_METRIC_MIPOBJECT` is passed down. A full list of all available attributes may be found in Chapter 8, or from the list in the `xprs.h` header file.

`iRankFirstIndex` Index (one-based) in the rank order of solutions for which the associated solution's id number, if there is a solution at this index, is to be returned in the first element of array `iSolutionIds`. If `iRankLastIndex > iRankFirstIndex` then any subsequent solutions in the rank ordering are to have their solution ids written to the subsequent elements of `iSolutionIds`.

`iRankLastIndex` If `iSolutionIds` is non-NULL then at most `iRankLastIndex - iRankFirstIndex + 1` solution ids will be written to `iSolutionIds`. There will be fewer solution ids written if `iRankLastIndex` is greater than the number of solutions registered in the MIP solution enumerator.

`iSolutionIds` Integer array where the solution ids will be returned. May be NULL if not required.

`nReturnedSolIds` Pointer to an integer where the number of solution ids that were available to be written is returned. This number will always be less than or equal to `iRankLastIndex - iRankFirstIndex + 1`. A value is returned for this parameter regardless of whether `iSolutionIds` is passed as NULL. May be NULL if not required.

`nSols` Pointer to an integer where the total number of solution ids that could possibly be written is returned. May be NULL if not required.

Related topics

`XPRS_mse_opt`, `XPRS_mse_minim`, `XPRS_mse_maxim`.

XPRS_mse_getsolmetric

Purpose

Gets the value of a metric for a solution.

Synopsis

```
int XPRS_CC XPRS_mse_getsolmetric(XPRSmipsolenum mse, int iSolutionId, int *
       iSolutionIdStatus, int iMetricId, double * dMetric)
```

Arguments

`mse` The current MIP solution enumerator.

`iSolutionId` The id of the solution for which the metric is to be returned.

`iSolutionIdStatus` Pointer to an `int` where the status of the `iSolutionId` will be returned. The returned value is one of:

- `-2` Solution id does not exist;
- `-1` Solution with the given id is already deleted;
- `0` Solution id was for an active solution.

`iMetricId` Id of metric whose value is to be returned. A full list of all available metrics may be found in Chapter 8, or from the list in the `xprs.h` header file.

`dMetric` Pointer to a double where the value of the metric will be returned.

Further information

1. The user will obtain the solution id `iSolutionId` from interaction with the MIP solution enumerator via functions such as `XPRS_mse_getsollist` and `XPRS_mse_getcullchoice`.
2. If `iSolutionId` is passed in as less than or equal to zero then the routine returns the worst value of the metric for the current set of solutions.

Related topics

`XPRS_mse_getsollist`, `XPRS_mse_getcullchoice`.

XPRS_mse_maxim

Purpose

One of three routines to run the enumeration. This routine starts the enumeration run on the problem by calling `XPRSmaxim` (p.285 of the Optimizer Reference Manual).

Synopsis

```
int XPRS_CC XPRS_mse_maxim(XPRSmipsolenum mse, XPRSprob prob, XPRSmipsolpool msp, int
        (XPRS_CC * f_mse_handler)(XPRSmipsolenum mse, XPRSprob prob, XPRSmipsolpool
        msp, void *ctx, int * const nMaxSols, const double * const x, const int nCols,
        const double dMipObject, double * const dModifiedObject, int * const
        bRejectSoln, int * const bUpdateMipAbsCutOffOnCurrentSet), void * p, int *
        nMaxSols)
```

Further information

See `XPRS_mse_minim` for details.

Related topics

`XPRSmaxim` (p.285 of the Optimizer Reference Manual), `XPRS_mse_minim`, `XPRS_mse_opt`.

XPRS_mse_minim

Purpose

One of three routines to run the enumeration. This routine starts the enumeration run on the problem by calling `XPRSminim` (p.285 of the Optimizer Reference Manual).

Synopsis

```
int XPRS_CC XPRS_mse_minim(XPRSmipsolenum mse, XPRSprob prob, XPRSmipsolpool msp, int
       (XPRS_CC * f_mse_handler)(XPRSmipsolenum mse, XPRSprob prob, XPRSmipsolpool
       msp, void *ctx, int * const nMaxSols, const double * const x, const int nCols,
       const double dMipObject, double * const dModifiedObject, int * const
       bRejectSoln, int * const bUpdateMipAbsCutOffOnCurrentSet), void * p, int *
       nMaxSols)
```

Arguments

`mse` The current MIP solution enumerator.

`prob` The problem used to enumerate solutions.

`msp` The MIP solution pool used to store the solutions found during enumeration.

`f_mse_handler` A callback function to handle the event that a MIP feasible solution is found in the enumeration.

`p` The user-defined object to be passed to the callback function `f_mse_handler`.

`nMaxSols` A pointer to an integer to be passed to the callback function `f_mse_handler` (and used only by the callback function) that is intended to store the maximum number of solutions the user is prepared to capture during the enumeration.

Further information

1. Although the callback function `f_mse_handler` may be user-defined a simple version (`XPRS_mse_defaulthandler`) is provided in the FICO Xpress Optimizer that can be used to manage common strategies for keeping a 'good' set of solutions found during enumeration and controlling the enumeration run e.g., `n-best` solutions.

2. Any MIP feasible solutions to `prob` stored in the MIP solution pool prior to the enumeration run are passed through the callback `f_mse_handler` as if they were found during the MIP search on the `prob`. Note that new versions of these solutions are created and the original versions are deleted from the MIP solution pool.

3. The integer pointer `nMaxSols` is used only by the callback function `f_mse_handler`.

Related topics

`XPRSminim` (p.285 of the Optimizer Reference Manual), `XPRS_mse_maxim`, `XPRS_mse_opt`, `XPRS_mse_defaulthandler`.

XPRS_mse_opt

Purpose

One of three routines to run the enumeration. This routine starts the enumeration run on the problem by calling `XPRSmipoptimize` (p.287 of the Optimizer Reference Manual).

Synopsis

```
int XPRS_CC XPRS_mse_opt(XPRSmipsolenum mse, XPRSprob prob, XPRSmipsolpool msp, int
        (XPRS_CC * f_mse_handler)(XPRSmipsolenum mse, XPRSprob prob, XPRSmipsolpool
        msp, void *ctx, int * const nMaxSols, const double * const x, const int nCols,
        const double dMipObject, double * const dModifiedObject, int * const
        bRejectSoln, int * const bUpdateMipAbsCutOffOnCurrentSet), void * p, int *
        nMaxSols)
```

Further information

See `XPRS_mse_minim` for details.

Related topics

`XPRSmipoptimize` (p.287 of the Optimizer Reference Manual), `XPRS_mse_minim`, `XPRS_mse_maxim`.

XPRS_mse_removecbmsghandler

Purpose

Removes an output callback function previously added by `XPRS_mse_addcbmsghandler`. The specified callback function will no longer be called after it has been removed.

Synopsis

```
int XPRS_CC XPRS_mse_removecbmsghandler(XPRSmipsolenum mse, int (XPRS_CC
       *f_msghandler)
       (XPRSobject vXPRSObject, void * vUserContext, void * vSystemThreadId, const
       char * sMsg, int iMsgType, int iMsgNumber), void* object);
```

Arguments

`mse` The current MIP solution enumerator

`f_msghandler` The callback function to remove. If NULL then all output callback functions added with the given user-defined object value will be removed.

`object` The object value that the callback was added with. If NULL, then the object value will not be checked and all variable branching callbacks with the function pointer `f_chgbranch` will be removed.

Related topics

`XPRS_mse_addcbmsghandler`.

XPRS_mse_setcbgetsolutiondiff

Purpose

Declares a user-defined solution difference calculation routine, called each time a new pair of solutions are required to have a difference metric calculated. This functionality is required when the MSE_METRIC_DIVERSITY metric is applied to the set of stored solutions with a call to XPRS_mse_getcullchoice and XPRS_mse_getsolmetric.

Synopsis

```
int XPRS_CC XPRS_mse_setcbgetsolutiondiff(XPRSmipsolenum mse, int (XPRS_CC *
       f_mse_getsolutiondiff)(XPRSmipsolenum mse, void * vContext, int nCols, int
       iSolutionId_1, int iElemCount_1, double dMipObj_1, const double * Vals_1, const
       int * iSparseIndices_1, int iSolutionId_2, int iElemCount_2, double dMipObj_2,
       const double * Vals_2, const int * iSparseIndices_2, double * dDiffMetric),
       void * p)
```

Arguments

`mse` The current MIP solution enumerator.

`f_mse_getsolutiondiff` The callback function which takes 14 arguments, `mse`, `vContext`, `nCols`, `iSolutionId_1`, `iElemCount_1`, `dMipObj_1`, `Vals_1`, `iSparseIndices_1`, `iSolutionId_2`, `iElemCount_2`, `dMipObj_2`, `Vals_2`, `iSparseIndices_2` and `dDiffMetric`. Use a NULL value to cancel a callback function.

`mse` The current MIP solution enumerator.

`vContext` The user-defined object passed to the callback function.

`nCols` The number of columns in the solutions. This is the same as the column dimension of the problem when the enumeration was started.

`iSolutionId_1` The solution id of the first of the solution pair. This may be -1 in which case the solution has not yet be stored in the MIP solution pool. This happens when the user passes a new solution to XPRS_mse_getcullchoice using the diversity metric (MSE_METRIC_DIVERSITY).

`iElemCount_1` The number of non-zero elements in the first solution.

`dMipObj_1` The MIP objective value of the first solution.

`Vals_1` A sparse array of doubles of length `iElemCount_1` containing the first solution's solution values.

`iSparseIndices_1` A sparse array of integers of length `iElemCount_1` containing the first solution's column indices.

`iSolutionId_2` The solution id of the second of the solution pair.

`iElemCount_2` The number of non-zero elements in the second solution.

`dMipObj_2` The MIP objective value of the second solution.

`Vals_2` A sparse array of doubles of length `iElemCount_2` containing the second solution's solution values.

`iSparseIndices_2` A sparse array of integers of length `iElemCount_2` containing the second solution's column indices.

`dDiffMetric` A pointer to a double to return the metric of the difference between the two solutions.

`p` A user-defined object to be passed to the callback function.

Further information

If the user does not provide this callback function then the MIP solution enumerator uses a simple difference metric that considers only the MIP entities in the solution.

Related topics

XPRS_mse_getcullchoice, XPRS_mse_getsolmetric.

XPRS_mse_setcbmsghandler

Purpose

Declares an output callback function, called every time a line of message text is output by a MIP solution enumerator object.

Synopsis

```
int XPRS_CC XPRS_mse_setcbmsghandler(XPRSmipsolenum mse, int (XPRS_CC
       *f_msghandler)(XPRSobject vXPRSObject, void * vUserContext, void *
       vSystemThreadId, const char * sMsg, int iMsgType, int iMsgNumber), void * p)
```

Arguments

`mse` The current MIP solution enumerator.

`f_msghandler` The callback function which takes six arguments, `vXPRSObject`, `vUserContext`, `vSystemThreadId`, `sMsg`, `iMsgType` and `iMsgNumber`. Use a NULL value to cancel a callback function.

`vXPRSObject` A generic pointer to the mse object sending the message.

`vUserContext` The user-defined object passed to the callback function.

`vSystemThreadId` The system id of the thread sending the message caste to a void *.

`sMsg` A null terminated character array (string) containing the message, which may simply be a new line. When the callback is called for the first time `sMsg` will be a NULL pointer.

`iMsgType` Indicates the type of output message:

- `1` information messages;
- `2` (not used);
- `3` warning messages;
- `4` error messages.

A negative value means the callback is being called for the first time.

`iMsgNumber` The number associated with the message. If the message is an error or a warning then you can look up the number in Chapter 11 of the Optimizer Reference Manual for advice on what it means and how to resolve the associated issue.

`p` A user-defined object to be passed to the callback function.

Further information

To send all messages to a log file the built in message handler `XPRSlogfilehandler` can be used. This can be done with:

```
XPRS_mse_setcbmsghandler(mse, XPRSlogfilehandler, "log.txt");
```

Related topics

None.

XPRS_mse_setdblcontrol

Purpose

Sets the value of a given double control parameter.

Synopsis

```
int XPRS_CC XPRS_mse_setdblcontrol(XPRSmipsolenum mse, int iControlId, double Val)
```

Arguments

`mse` The current MIP solution enumerator.

`iControlId` Id of control whose value is to be set. A full list of all available controls may be found in Chapter 7, or from the list in the `xprs.h` header file.

`Val` Value to which the control parameter is to be set.

Related topics

`XPRS_mse_getdblcontrol`, `XPRS_mse_setintcontrol`, `XPRS_mse_getintcontrol`.

XPRS_mse_setintcontrol

Purpose

Sets the value of a given integer control parameter.

Synopsis

```
int XPRS_CC XPRS_mse_setintcontrol(XPRSmipsolenum mse, int iControlId, int Val)
```

Arguments

`mse` The current MIP solution enumerator.

`iControlId` Id of control whose value is to be set. A full list of all available controls may be found in Chapter 7, or from the list in the `xprs.h` header file.

`Val` Value to which the control parameter is to be set.

Related topics

`XPRS_mse_getintcontrol`, `XPRS_mse_setdblcontrol`, `XPRS_mse_getdblcontrol`.

XPRS_mse_setsolbasename

Purpose

Sets the name to be used as a prefix for solutions found during the enumeration run.

Synopsis

```
int XPRS_CC XPRS_mse_setsolbasename(XPRSmipsolenum mse, const char *
       sSolutionBaseName)
```

Arguments

`mse` The current MIP solution enumerator.

`sSolutionBaseName` A null terminated string containing the name to use as a prefix for the solution names.

Related topics

`XPRS_mse_getsolbasename`.

CHAPTER 7

MSE Controls

MSE_CALLBACKCULLSOLS_DIVERSITY This control is used only by the callback passed to the enumeration-run routines XPRS_mse_minim, XPRS_mse_maxim and XPRS_mse_opt. If the user passes the default callback XPRS_mse_defaulthandler then the control is used to decide how many solutions to discard based on the diversity metric values. See XPRS_mse_defaulthandler for details of how the control is used. If the user provides a customized version of XPRS_mse_defaulthandler to use in the enumeration-run then the user is free to interpret this control value. p. 97

MSE_CALLBACKCULLSOLS_MIPOBJECT This control is used only by the callback passed to the enumeration-run routines XPRS_mse_minum, XPRS_mse_maxim and XPRS_mse_opt. If the user passes the default callback XPRS_mse_defaulthandler then the control is used to decide how many solutions to discard based on the MIP objective values. See XPRS_mse_defaulthandler for details of how the control is used. If the user provides a customized version of XPRS_mse_defaulthandler to use in the enumeration-run then the user is free to interpret this control value. p. 98

MSE_CALLBACKCULLSOLS_MODOBJECT This control is used only by the callback passed to the enumeration-run routines XPRS_mse_minim, XPRS_mse_maxim and XPRS_mse_opt. If the user passes the default callback XPRS_mse_defaulthandler then the control is used to decide how many solutions to discard based on the user's modified objective values. See XPRS_mse_defaulthandler for details of how the control is used. If the user provides a customized version of XPRS_mse_defaulthandler to use in the enumeration-run then the user is free to interpret this control value. p. 98

MSE_OPTIMIZEDIVERSITY Controls whether or not the values returned by XPRS_mse_getcullchoice for the diversity metric are defined by simply sorting the diversity values or if an optimization-based heuristic is to be used. p. 98

MSE_OUTPUTTOL Zero tolerance on print values. p. 99

MSE_CALLBACKCULLSOLS_DIVERSITY

Description This control is used only by the callback passed to the enumeration-run routines XPRS_mse_minim, XPRS_mse_maxim and XPRS_mse_opt. If the user passes the default

callback XPRS_mse_defaulthandler then the control is used to decide how many solutions to discard based on the diversity metric values. See XPRS_mse_defaulthandler for details of how the control is used. If the user provides a customized version of XPRS_mse_defaulthandler to use in the enumeration-run then the user is free to interpret this control value.

Type Integer

Default value -1

Affects routines XPRS_msp_getintcontrol, XPRS_msp_setintcontrol.

See also MSE_METRIC_DIVERSITY

MSE_CALLBACKCULLSOLS_MIPOBJECT

Description This control is used only by the callback passed to the enumeration-run routines XPRS_mse_minum, XPRS_mse_maxim and XPRS_mse_opt. If the user passes the default callback XPRS_mse_defaulthandler then the control is used to decide how many solutions to discard based on the MIP objective values. See XPRS_mse_defaulthandler for details of how the control is used. If the user provides a customized version of XPRS_mse_defaulthandler to use in the enumeration-run then the user is free to interpret this control value.

Type Integer

Default value -1

Affects routines XPRS_msp_getintcontrol, XPRS_msp_setintcontrol

See also XPRS_mse_defaulthandler, XPRS_mse_getcullchoice, MSE_METRIC_MODOBJECT.

MSE_CALLBACKCULLSOLS_MODOBJECT

Description This control is used only by the callback passed to the enumeration-run routines XPRS_mse_minim, XPRS_mse_maxim and XPRS_mse_opt. If the user passes the default callback XPRS_mse_defaulthandler then the control is used to decide how many solutions to discard based on the user's modified objective values. See XPRS_mse_defaulthandler for details of how the control is used. If the user provides a customized version of XPRS_mse_defaulthandler to use in the enumeration-run then the user is free to interpret this control value.

Type Integer

Default value -1

MSE_OPTIMIZEDIVERSITY

Description Controls whether or not the values returned by XPRS_mse_getcullchoice for the diversity metric are defined by simply sorting the diversity values or if an optimization-based heuristic is to be used.

Type Integer

Values

0	Sorting is used.
1	An optimization-based heuristic is used.

Default value 0

Affects routines XPRS_msp_getintcontrol, XPRS_msp_setintcontrol

See also MSE_METRIC_DIVERSITY, XPRS_mse_getcullchoice.

MSE_OUTPUTTOL

Description Zero tolerance on print values.

Type Double

Default value 1.0E-05

CHAPTER 8

MSE Attributes

MSE_DIVERSITYSUM

Description Sum of diversity metrics for the current set of solutions.

Type Double

Set by routines XPRS_msp_getdblattrib

See also MSE_METRIC_DIVERSITY

MSE_METRIC_DIVERSITY

Description The diversity metric for a solution is the sum of difference metrics between the solution and all others in the current set. The difference value between two solutions is either calculated internally by the MIP solution enumerator (using a simple difference metric

that considers only the MIP entities in the solution) or else by the user in a callback defined with a call to XPRS_mse_setcbgetsolutiondiff.

Type Double

Set by routines XPRS_msp_getsolmetric

See also XPRS_mse_setcbgetsolutiondiff

MSE_METRIC_MIPOBJECT

Description MIP objective value.

Type Double

MSE_METRIC_MODOBJECT

Description User-defined modified objective. The user can assign a solution with an objective value that reflects the 'true' objective of the solution in cases where the MIP objective does not completely model the user's problem. The user needs to return this value from a callback passed to the enumeration-run routines XPRS_mse_minim, XPRS_mse_maxim and XPRS_mse_opt. This routine will be some customized version of the XPRS_mse_defaulthandler routine provided with the FICO Xpress Optimizer.

Type Double

Set by routines XPRS_msp_getsolmetric

See also XPRS_mse_defaulthandler, XPRS_mse_minim, XPRS_mse_maxim, XPRS_mse_opt.

MSE_SOLUTIONS

Description The number of solutions found and currently stored by the MIP solution enumerator.

Type Integer

Set by routines XPRS_msp_getintattrib

See also XPRS_mse_defaulthandler

Appendix

APPENDIX A

Error codes

Following an error exit from a function call to the MIP solution pool or the MIP solution enumerator library users may access the error code and a short string description of the error using the functions XPRS_msp_getlasterror or XPRS_mse_getlasterror. The following sections list the error codes returned by calls to the MIP solution pool and MIP solution enumerator functions. The main Optimizer Reference manual contains the list of error codes returned by the rest of the API.

A.1 MIP Solution Pool errors

800 ***First index outside bounds: <col_index>not in [0,<num_cols>- 1]***

The index col_index of the first column passed to XPRS_msp_getsol is outside the range [0, <num_cols>- 1]

801 ***First index greater than last***

The index of the first column passed to XPRS_msp_getsol is greater than the index of the last column.

802 ***Attempt to attach problem more than one XPRSmipsolpool: Attach failed***

A problem cannot be attached to more than one MIP solution pool.

804 ***Unable to check if loaded sol is dupld: No global model avail***

A check if the solution being loaded is a duplicate of an existing solution cannot be made since no global model is available. No model is available because no problem has been attached to the MIP solution pool.

805 ***Unable to check if loaded sol is dupld: Memory allocation failure***

A check if the solution being loaded is a duplicate of an existing solution failed because of a memory allocation failure.

806 ***Unable to check if loaded sol is dupld: Incompat src prob***

A check if the solution being loaded is a duplicate of an existing solution cannot be made because the problem from which the solution originates does not match the global model used for duplicate checking.

807 ***Unable to check if loaded sol is dupld: Global model unreliable***

A check if the solution being loaded is a duplicate of an existing solution cannot be made because a required update of the global model could not be made reliably.

808 ***Unable to check if loaded sol is dupld: Global model reqd but inaccessible***

A check if the solution being loaded is a duplicate of an existing solution cannot be made because a required update of the global model could not be made.

809 ***Duplicate sol chck may not work: Global model inaccessible in attaching prob***

A check if the solution being loaded is a duplicate of an existing solution cannot be made because a required capture of the global model could not be made from the problem being attached.

810 ***Destroying XPRSmipsolpool with problems still attached: Probs detached automatically***

The MIP solution pool is being destroyed and problems are still attached. This should be ok although the user should be aware. This message can be avoided by detaching the attached problems before destroying the MIP solution pool.

812 ***Compl duplicate sol chck: Failed***

A reset operation for the duplicate solution information failed. The duplicate solution checking may now not work reliably.

814 ***Failed to capture solution information***

Failure calculating the information for a solution with respect to a problem.

815 ***Failed to generate solution check sum***

A check of the problem state information failed when trying to calculate a check sum.

816 ***Failed to capture solution***

An unexpected failure when attempting to capture a solution.

817 ***Failure in internal event registration***

An object failed to register to receive system events.

818 ***Failure in internal event deregistration***

An object failed to deregister the receipt of system events.

821 ***Unrecoverable memory allocation failure***

A call made by the user failed because of a memory allocation failure.

822 ***Recoverable memory allocation failure: <operation_attempted>***

A memory allocation failed when attempting the operation <operation_attempted >. The system may still continue to function normally.

826 ***Soln col count does not match prob: <prob_id_string>: <solution_id>***

The numbers of columns of solution <solution_id>and of problem <prob_id_string>are required to be the equal (but they are not).

827 ***Soln is deleted and no longer available: <solution_id>***

The user has attempted to access information for the solution with id <solution_id>but the solution has been deleted.

829 ***Soln id does not exist: <solution_id>***

The user has attempted to access information for the solution with id <solution_id>but the id is for a solution that does not exist.

830 ***Unable to make a reliable list of sols for prob: <prob_id_string>: Unable to access prob state***

An attempt to make a list of solutions that relate to problem <prob_id_string>failed because state information of the problem could not be reliably accessed.

831 ***Unrecognized attribute id <attribute_id>: Solutions will not be sorted***

The user seems to want a list of solutions sorted by some attribute but the attribute id is not recognized.

835 ***Column count does not match prob: <num_solution_cols>: <num_prob_cols>***

The numbers of columns of a given solution vector and of a given problem are required to be the equal (but they are not).

836 ***Failure writing solution to file: <file_name>***

An unexpected failure was encountered writing solution(s) to file <file_name>.

837 ***Failure opening file for writing: <file_name>***

An unexpected failure was encountered opening file <file_name>for writing.

838 ***Failure opening file for reading: <file_name>***

An unexpected failure was encountered opening file <file_name>for reading.

839 ***Loading sol failed : <prob_id_string>: id=<solution_id><message>***

A failure occurred when loading solution <solution_id>into problem <prob_id_string>. The failure is described in string <message>.

842 ***Loading sol interrupted: <prob_id_string>: id=<solution_id>***

Loading was interrupted of solution <solution_id>into problem <prob_id_string>.

851 ***Unrecognized duplicate solution policy id: <policy_id>***

The user has attempted to set an unrecognized value <policy_id>for the control `MSP_DUPLICATESOLUTIONSPOLICY`.

852 ***Reading solution skipped: '<solution_name>'***

The solution named <solution_name>was skipped while reading solutions from file. A previous message describes why the solution could not be read.

853 ***Error detected in debug duplicate solution check: <message>***

A rigorous duplicate solution checking routine found an anomaly described in <message>.

855 ***Changing <tolerance_name>for solution '<solution_name>' captured from problem: <system_tol_value>-><user_tol_value>***

A warning that the user is changing the tolerance <tolerance_name>(MIPTOL or FEASTOL) for a solution from the value it was set to by the system <system_tol_value>to a different value <user_tol_value>.

859 ***No solutions written***

No solutions were written in a call to `XPRS_msp_writeslxsol`.

868 ***<operation_name>is only allowed for attached problems***

The operation <operation_name>is only allowed for problems attached to the MIP solution pool.

869 ***Soln is deleted and no longer available: <solution_id>***

The user has attempted to access information for the solution with id <solution_id>but the solution seems to have been deleted.

870 ***Soln id does not exist: <solution_id>***

The user has attempted to access information for the solution with id <solution_id>but the id is for a solution that has not yet been encountered.

871 ***Unrecoverable memory allocation failure***

A call made by the user failed because of a memory allocation failure.

874 ***Unrecognized solution metric id: <metric_id>***

The metric id <metric_id>passed by the user is unrecognized.

875 ***No solutions available***

The user is attempting to access information about the stored solutions but the solution storage is empty.

876 ***Enumerator is already running***

The user is attempting to run the enumeration while the enumeration is already running.

875 ***Failed to access XPRSmipsolpool state: <access_operation>***

The MIP solution enumerator failed accessing the MIP solution pool state with access operation <access_operation>.

878 ***Failed to access XPRSprob state: <access_operation>***

The MIP solution enumerator failed accessing the problem state with access operation <access_operation>.

879 ***Failed to allocate memory***

The MIP solution enumerator failed to allocate memory.

882 ***Missing required arguments for running enumerator***

Arguments passed to the enumeration-run routine (e.g., XPRS_mse_minim) were required to be non-NULL but were passed as NULL.

883 ***Problem no longer has the same number of columns***

The column dimension of the problem used by the MIP solution enumerator was changed by user.

885 ***Failure storing solution with memory allocation failure***

A memory allocation failure meant that a solution could not be captured.

886 ***Failure storing solution(s) with failed access to XPRSmipsolpool state: <access_operation>***

A solution could not be captured because of a failure accessing the state of the MIP solution pool with access operation <access_operation>.

887 ***Failure storing solution(s) with failed access to XPRSprob state: <access_operation>***

A solution could not be captured because of a failure accessing the state of the problem with access operation <access_operation>.

1000 ***Invalid parameter access : <error_message>(<routine_name>)***

Access to a parameter with a call to routine <routine_name>failed because of error described in <error_message>.

1006 ***Failed to capture solution information from problem***

Failure calculating the information for a solution with respect to a problem.

1012 ***Problem <prob_id_string>detached unexpectedly***

A problem <prob_id_string>attached to the MIP solution pool was unexpectedly detached by the user.

1013 ***Cannot attach thread problem to MIP solution pool***

The user has attempted to attach an internal problem being solved by the parallel MIP search to a MIP solution pool.

1014 ***Column count mismatch: <num_solution_cols>!= <num_row_vector_cols>***

A row vector of length <num_row_vector_cols>passed to the MIP solution pool for evaluation against a solution does not contain the same number of columns as the solution <num_solution_cols>.

A.2 MIP Solution Enumerator errors

869 ***Soln is deleted and no longer available: <solution_id>***

The user has attempted to access information for the solution with id <solution_id>but the solution seems to have been deleted.

870 ***Soln id does not exist: <solution_id>***

The user has attempted to access information for the solution with id <solution_id>but the id is for a solution that has not yet been encountered.

871 ***Unrecoverable memory allocation failure***

A call made by the user failed because of a memory allocation failure.

874 ***Unrecognized solution metric id: <metric_id>***

The metric id <metric_id>passed by the user is unrecognized.

875 ***No solutions available***

The user is attempting to access information about the stored solutions but the solution storage is empty.

876 ***Enumerator is already running***

The user is attempting to run the enumeration while the enumeration is already running.

877 ***Failed to access XPRSmipsolpool state: <access_operation>***

The MIP solution enumerator failed accessing the MIP solution pool state with access operation <access_operation>.

878 ***Failed to access XPRSprob state: <access_operation>***

The MIP solution enumerator failed accessing the problem state with access operation <access_operation>.

879 ***Failed to allocate memory***

The MIP solution enumerator failed to allocate memory.

882 ***Missing required arguments for running enumerator***

Arguments passed to the enumeration-run routine (e.g., `XPRS_mse_minim`) were required to be non-NULL but were passed as NULL.

883 ***Problem no longer has the same number of columns***

The column dimension of the problem used by the MIP solution enumerator was changed by user.

885 ***Failure storing solution with memory allocation failure***

A memory allocation failure meant that a solution could not be captured.

886 ***Failure storing solution(s) with failed access to `XPRSmipsolpool` state: <access_operation>***

A solution could not be captured because of a failure accessing the state of the MIP solution pool with access operation <access_operation>.

887 ***Failure storing solution(s) with failed access to XPRSprob state: <access_operation>***

A solution could not be captured because of a failure accessing the state of the problem with access operation <access_operation>.

1000 ***Invalid parameter access : <error_message>(<routine_name>)***

Access to a parameter with a call to routine <routine_name>failed because of error described in <error_message>.

1007 ***Unexpected failure evaluating solution difference metric***

An unexpected failure occurred while evaluating difference metrics for solutions.

1008 ***Solution difference metric value is reserved : -1.7976931348623158e+308***

The double value -1.7976931348623158e+308 to be assigned for a difference metric between two solutions is reserved by the system.

1009 ***Unexpected failure evaluating solution differences***

An unexpected error occurred while evaluating the diversity of the stored solutions.

1010 ***User signaled failure in solution difference metric callback***

The user signaled a failure in the callback defined by a call to `XPRS_mse_setcbgetsolutiondiff`.

1011 ***Cannot evaluate diversity of new solution: Column number mismatch: <num_solution_cols>!= <num_prob_cols>***

The solution passed down to `XPRS_mse_getcullchoice` for the diversity metric does not have the same number of columns as the problem used in the enumeration run.

1015 ***Destroying while enumeration running***

The user is attempting to destroy the MIP solution enumerator while it is running.

1016 ***Failed to update MIPABSCUTOFF***

The MIP solution enumerator failed to update the `MIPABSCUTOFF` (p.422 of the Optimizer Reference Manual) on the problem running the enumeration.

1017 ***Unable to check if loaded sol is dupld: Sol cols (num_solution_cols) does not match global model (num_global_model_cols)***

A check if the solution being loaded is a duplicate of an existing solution cannot be made because the solution array does not have the same number of columns as the global model used for duplicate checking.

1018 ***Global model unavailable for diversity: <reason_string>***

The MIP solution enumerator is setup such that internal solution diversity calculations may be made although this will fail because it requires that an appropriate problem is attached to the MIP solution pool (see <reason_string> for details).

1019 ***Cannot handle diversity: <reason_string>***

The user has called a function that requires solution diversity calculations to be made internally by the MIP solution enumerator and this has failed because it requires that an appropriate problem is attached to the MIP solution pool (see <reason_string> for details).

APPENDIX B

Contacting FICO

FICO provides clients with support and services for all our products. Refer to the following sections for more information.

Product support

FICO offers technical support and services ranging from self-help tools to direct assistance with a FICO technical support engineer. Support is available to all clients who have purchased a FICO product and have an active support or maintenance contract. You can find support contact information on the Product Support home page (www.fico.com/support).

On the Product Support home page, you can also register for credentials to log on to FICO Online Support, our web-based support tool to access Product Support 24x7 from anywhere in the world. Using FICO Online Support, you can enter cases online, track them through resolution, find articles in the FICO Knowledge Base, and query known issues.

Please include *'Xpress'* in the subject line of your support queries.

Product education

FICO Product Education is the principal provider of product training for our clients and partners. Product Education offers instructor-led classroom courses, web-based training, seminars, and training tools for both new user enablement and ongoing performance support. For additional information, visit the Product Education homepage at www.fico.com/en/product-training or email producteducation@fico.com.

Product documentation

FICO continually looks for new ways to improve and enhance the value of the products and services we provide. If you have comments or suggestions regarding how we can improve this documentation, let us know by sending your suggestions to techpubs@fico.com.

Sales and maintenance

USA, CANADA AND ALL AMERICAS

Email: XpressSalesUS@fico.com

WORLDWIDE

Email: XpressSalesUK@fico.com

Tel: +44 207 940 8718
Fax: +44 870 420 3601

Xpress Optimization, FICO
FICO House
International Square
Starley Way
Birmingham B37 7GN
UK

Related services

Strategy Consulting: Included in your contract with FICO may be a specified amount of consulting time to assist you in using FICO Optimization Modeler to meet your business needs. Additional consulting time can be arranged by contract.

Conferences and Seminars: FICO offers conferences and seminars on our products and services. For announcements concerning these events, go to www.fico.com or contact your FICO account representative.

About FICO

FICO (NYSE:FICO) delivers superior predictive analytics solutions that drive smarter decisions. The company's groundbreaking use of mathematics to predict consumer behavior has transformed entire industries and revolutionized the way risk is managed and products are marketed. FICO's innovative solutions include the FICO® Score—the standard measure of consumer credit risk in the United States—along with industry-leading solutions for managing credit accounts, identifying and minimizing the impact of fraud, and customizing consumer offers with pinpoint accuracy. Most of the world's top banks, as well as leading insurers, retailers, pharmaceutical companies, and government agencies, rely on FICO solutions to accelerate growth, control risk, boost profits, and meet regulatory and competitive demands. FICO also helps millions of individuals manage their personal credit health through www.myfico.com. Learn more at www.fico.com. FICO: Make every decision count™.

Index

X