Xpress-BCL Reference Manual

Release 2.0

Dashへのお問い合わせ

Xpress-MP の使用についてのご質問，ご意見は、以下のテクニカルサポートにご連絡ください。
Xpress-MP ソフトウェアの販売や注文**について**のお問い合わせは、あなたの地域の販売所か以下のDash 販売所にご連絡下さい。
最新のニュース、Xpress-MP ソフトウェアやドキュメントのアップデートについては、Xpress-MP のウェブサイトをご覧下さい。
http://www.dashoptimization.com/

USA, Canada and The Americas Elsewhere
Dash Optimization Inc. Dash Optimization Ltd.
560 Sylvan Avenue Quinton Lodge, Binswood Avenue
Englewood Cliffs Leamington Spa
NJ 07632 Warwickshire CV32 5RX
USA UK
Telephone: (201) 567 9445 *Telephone:* +44 1926 315862
Fax: (201) 567 9443 *Fax:* +44 1926 315854
email: support-usa@dashoptimization.com *email:* support@dashoptimization.com
USA, Canada and The Americas Elsewhere
Dash Optimization Inc. Dash Optimization Ltd.
560 Sylvan Avenue Blisworth House, Church Lane
Englewood Cliffs Blisworth
NJ 07632 Northants NN7 3BX
USA UK
Telephone: (201) 567 9445 *Telephone:* +44 1604 858993
Fax: (201) 567 9443 *Fax:* +44 1604 858147
email: sales@dashoptimization.com *email:* sales@dashoptimization.com

目次

1 はじめに 1
1.1 Xpress-BCLの概要 1
1.2 Optimizer Library利用者への注意 2
1.3 マニュアルの構成. 3
1.4 表記法 3
2 BCLによるモデリング 5
2.1 問題の処理 5
2.2 変数 8
2.3 制約式 10
2.4 問題の解についての情報. 13
3 進んだ使い方 15
3.1 データ入力とインデックス集合 15
3.2 Special Ordered Sets 16
3.3 出力とプリント 17
3.4 BCLによる2次計画問題 18
4 BCL ライブラリ関数 21
4.1 関数記述のための表記 21
5 BCL Error Messages 139
A Optimizer ライブラリーによるBCLの利用 128
A.1 ライブラリー間の切り替え 128
A.2 初期化と終了 129
A.3 行列の読み込み 129
A.4 行列要素のインデックス 130
A.5 BCL互換な関数の利用. 130
B BCL in C++ and Java 151
B.1 C++によるBCLの利用の概要. 132
B.2 JavaによるBCLの利用の概要. 132
B.3 C++ と Java のクロスレファレンス. 133
クラスＸＰＲＢｐｒｏｂ. 134
クラスＸＰＲＢｖａｒ. 136
クラスＸＰＲＢｃｔｒ. 137
クラスＸＰＲＢｓｏｓ. 138
クラスＸＰＲＢｉｎｄｅｘＳｅｔ. 138
クラスＸＰＲＢｂａｓｉｓ. 139
B.4 追加クラス 140
クラスＸＰＲＢｌｉｎＥｘｐ. 140
クラスＸＰＲＢｑｕａｄＥｘｐによるクラスＸＰＲＥＢｌｉｎＥｘｐの拡張. 141
クラスＸＰＲＢｌｉｎＲｅｌによるクラスＸＰＲＥＢｌｉｎＥｘｐの拡張. 142
クラスＸＰＲＢ. 142

Index 171. 143

Introduction 1

1 はじめに

1.1 Xpress-BCLの概要

Xpress-BCL (Builder Component Library)はXpress-MPの利用者が簡単に線形計画問題や混合型整数計画問題，二次計画問題を定式化して解くための環境を提供する．BCLの豊富な関数を使うことで，複雑なモデルも迅速かつ簡単に構築でき，最適化問題として扱うことができる．BCLは柔軟性のあるソフトなので，単に個々のモデル構築に利用するばかりでなく，汎用的なモデリングソフトウェアの構築に際してもその理想的な主要部分として利用可能である． Xpress-Optimizerと一緒に利用することで強力なツールとなる．

Xpress-BCL によるモデル定式化は制約式志向である．制約式は，係数を一つひとつ入れて入力することもできるし，係数の配列と変数の配列を使って内積の形で一度に入力することもできる．前者の方法は，一旦構築したモデルを簡単に手直しできるし，後者は新しい制約式を即座に追加することができる．

BCLではXpress-Optimizerで利用できる全ての変数の型を利用できる．すなわち，連続，半連続，２値，半整数，整数，部分整数の各変数とSpecial Ordered 1と2 (SOS1とSOS2)である．全探索の実行を制御するいくつかの関数をあわせて利用することで，BCLは複雑な（混合型）整数計画問題に対応することができる．

モデル構築を補完するために，BCLはXpress-Optimizerに直接モデルを渡して解かせ，結果をレポートさせることができる関数群を提供している．もし，BCLの関数が能力不十分で利用者の要求に応えられなければ， Xpress-Optimizerで使われる高度なルーチンを利用して問題を解くことができる．BCLとXpress-Optimizer のこのようなインタラクティブな利用は，両者の完全な相互利用を可能にする．.

BCLは，モデルとその解の入出力を簡単に行うための多くの関数を提供している．多くの有用なプリントのための関数のほか，構築したモデルを行列形式のファイルや，その他の標準的な形式のファイルにして出力するための多くの関数を提供している．

1.2 Optimizer Library 利用者への注意

BCLの関数はモデリングのあらゆる局面を扱い，利用者がその背後で使われる解法で問題がどのように表現されているか（行列形式）を気にすることなく，簡単に最適化問題を実行することができる．しかし，高度な利用者が問題の行列を直接扱いたいと思うならば，BCLが生成した行列に対してOptimizerのライブラリ関数を使うことも可能である．このために，BCLは行列を扱うためのいくつかの関数を提供している．

関数 `XPRBloadmat` はBCLの制約式表現をOptimizerで使われる行列表現に変換する．BCLは自動的にこの関数を実行するので，通常はこの関数を呼ぶ必要はない．

関数 `XPRBgetcolnum` と `XPRBgetrownum` はそれぞれ，指定した変数に対応する列番号，指定した制約式に対応する行番号を返す．`XPRBloadmat`により行列を読み込むとき，BCLは制約式に現れない変数や空の制約式は無視し，変数や制約式の番号を付け替える（これらの無視された変数や制約式には負の番号が付けられて，Optimizerが扱っている行列の要素ではないこと示す）

注意して欲しいことは，BCLはOptimizerの中で行われる処理によって生じる行列の変化を掌握しないということである．よって，もし線形や整数の問題で前処理が行われると，正しい解についての情報は，後処理をして始めて得られる．これらの処理は，解が正常に求められた場合は自動的に行われる（４章 BCL ライブラリ関数の`XPRBsolve` を参照）．

もしOptimizerの関数（たとえば`XPRSaddrows`や`XPRSchgcoef`）を使って行列を直接変更したら，もはや，その変更結果をBCLのモデル側へと反映させることはできない．BCL とOptimizerの間で一貫したモデルを保つためには，モデルの変更はBCL側で行って，`XPRBloadmat`関数を呼出す．

Appendix A, “Using BCL with the Optimizer Library” ではBCLのプログラムの中からOptimizerの関数を利用する方法が詳細に記述されている．興味のある利用者は参照されたい．

1.3 マニュアルの構成

このマニュアルは大きく分けて２つの部分よりなる．最初の部分では，２章の「BCLによるモデリング」でよく使われるBCL関数を概観する．モデルの構築や，最適化によって得られた解および付帯情報について概説する．続く3章「進んだ使い方」では，より高度なしかしあまり知られていないライブラリの使い方を述べる．インデックスセットやSOSの使い方，2次計画問題などが扱われる．

後半では，4章「BCLライブラリ関数」で全てのライブラリ関数についての詳細な情報が述べられる．5章「BCLエラーメッセージ」はBCLが返すエラーメッセージの一覧である．付録AではXpress Optimizerの関数をBCLで使うときの説明の概要，付録Bでは，C++ およびJavaとのインターフェースについて述べる．

1.4 表記法

このマニュアル中では，標準的な活字表記法を用いる．コンピュータのコードを表すのには，`fixed width font`を用いる．数式は*italic* で表記する．ライブラリ関数の表記にはいくつかの可能性があるが，C言語の表記法に近いものが使われる．

- square brackets [...] はオプション;
- curly brackets {...} オプションだが，このうちの1つは選ばなければいけない;
- 表記中にあらわれる*italics* はメタ変数を表す．メタ変数の後に続く記述は，それがどのように使われるかを記述している;
- 記号は‘RETURN または Enter を押す’を意味する ;
- CTRL のあとに文字が続く場合は，CTRL キーを押しながらその文字を押す;

• 縦バー | はキーボード上ではしばしば ¦ で表示される．ディスプレイ上で中央に穴があいて表示されなくても惑わされないこと．UNIX では，パイプと呼ばれる．これは，ボックスなどを書くときに用いる特別なキャラクタとは違うので注意する．ASCIIコードでは，記号 | は16進で7C，10進で124 である．

Xpress-BCL Reference Manual Conventions Used 4

Introduction 1

Xpress-BCL Reference Manual Modeling with BCL 5

2 BCLによるモデリング

2.1 問題の処理

初期化と終了

BCL関数のプロトタイプはヘッダファイルxprb.hに記述されているので，BCL関数を呼ぶプログラムの最初にこのファイル読み込むこと．モデル構築の最初の段階は関数XPRBinitを呼んで，BCLの初期化を行うことである．この関数はXpress-Optimizerの初期化も同時におこうなうので，BCLとOptimizerを両方使うとき，個々のソフトでXPRSinitを呼ぶ必要はない． このXPRBinitは必要なライブラリのチェックと，インストールされているXpress-MPのライセンスに関する情報のチェックを行う．一度このチェックをして始めて全ての関数が利用できるようになる．
モデルの構築が終わり，BCLのルーチンが必要なくなれば，XPRBfreeを終了前に呼ぶ．
この関数はBCL関数を利用する過程で開いたファイルを閉じ，各種リソースを解放する．

問題と問題ポインタ

Xpress-Optimizer libraryの場合と同様に，BCLで問題を扱うときは，問題ポインタというものを利用する．このBCLおよびOptimizerの枠組みでは，構築された各問題はポインタ（レファレンスということもある）によって保持され，各関数はこのポインタを受け取って問題を処理する．この枠組みは多くの利点を持つ．なかでも一番の利点は多くの異なる問題を同時に扱うことができる点であろう．また問題ポインタの利用により，現行の問題（active problem）という概念はBCLにはない．この枠組みはOptimizerにも共通のものなので，両者の間での問題の受け渡しは容易であり，様々なモデリングや最適化問題の構築を可能にしている．
BCLの問題ポインタはXPRBprob型の変数であり，各プログラムの先頭で宣言されなければいけない．このポインタは関数XPRBnewprobによって（必要ならば問題名をつけて）初期化される．以下はその例である．

```
XPRBprob prob;
...
XPRBinit("");
prob = XPRBnewprob("MyProb");
```


問題ポインタprobは，問題を処理する各関数の第1引数として以後使われる．ある問題の処理が全て終わったら，その問題ポインタを関数XPRBdelprobを呼んで削除し，関連する資源を解放する．注意しなければいけないことは，XPRBfreeによっては各問題に関連した資源の解放は行われないことである．個々の問題ポインタの削除を行わないとメモリ不足を生じさせる可能性がある．

```
新しいモデルの初期化 XPRBprob pb1;
...
pb1 = XPRBnewprob("Problem1");
モデルの削除 XPRBdelprob(pb1);
問題の行列の読み込み XPRBloadmat(pb1);
問題名の獲得 XPRBgetprobname(pb1);
```

Figure 2.1 問題の読み込み，アクセス，変更

その他の基本関数

その他にも問題の処理に役立つ関数がある． 関数XPRBgetprobnameによって指定した問題の問題名を獲得することができる．

関数XPRBloadmatはOptimizer libraryの利用者に必要なの関数である．この関数はBCLで定義された問題をOptimizerで使われる行列表現に変換し，Optimizerに渡す．通常この作業はBCLによって自動的に行われるのでが，実際に行列を自分で作ったり修正したりしてこの関数を呼んでみるのは，いい練習であり，また実用上も有用である．

入出力の設定

BCL は入出力を制御する多くの関数を提供している．これらの関数は特定の問題とは無関係なので，問題ポインタを引数としてとることはなく，XPRBnewprobによる問題の初期化に先立って呼ぶことが可能である．実際これらの関数はXPRBinitによるBCLの初期化の前でさえ呼ぶことが可能である．BCLの他の関数はXPRBinitの前に呼ばれるとエラーを返す．
BCLの出すメッセージ類は（関数XPRBsetmsglevelによって）抑制できる．関数XPRBdefcbmsgを使うと，他のBCLのプリント出力（Optimizerからのメッセージ，ユーザ作成プログラムでXPRBprintfでのプリント出力）に割り込むメッセージコールバック関数を定義できる．

BCLでのデータ入力に際し（関数 XPRBreadlineやXPRBreadarrlineを使う），（既定の）英米で標準の小数点の記法を他の記法（小数点にコンマを使うなど）に変更することが可能である（XPRBsetdecsignを使う）．

小数点の記号の設定
```
XPRBsetdecsign(',');
```
メッセージ出力の制御
```
XPRBsetmsglevel(prob,1);
```
エラーメッセージの制御
```
XPRBseterrctrl(0);
```
エラー制御のためのコールバック
```
void myerror(XPRBprob my_prob,
void *my_object, int num, int type,
char *txt);
XPRBdefcberr(prob,myerror,object);
```
プリントコールバック
```
void myprint(XPRBprob my_prob,
void *my_object, char *msgtext);
XPRBdefcbmsg(prob,myprint,object);
```
バージョンの確認
```
char *version;
version = XPRBgetversion();
```

Figure 2.2 入出力の設定，エラー処理

エラー処理

既定の設定ではBCLは，なんらかのエラーがおこるとプログラムの実行を停止する．関数XPRBseterrctrlを使うと利用者はこれを変更することができる：エラーメッセージは出続ける，プログラムは何らかのエラー処理をしなければならない．この関数はBCLのプログラムを他のアプリケーションに埋め込んで使うときなどに有効である．
コールバックは（XPRBdefcberrによって），BCLが出力する他の警告やエラーメッセージに割り込むように定義できる．この関数はXPRBsetmsglevelの影響を受けないので，利用者は他のメッセージの出力を抑制しても，この関数による出力を知ることができる．
ソフトウェアに関する問題を問い合わせるときは，必ずBCLのバージョンを知らせて欲しい．これはXPRBgetversionによって知ることができる．

Modeling with BCL 2

2.2 変数

基本機能

BCLでは，変数は関数XPRBnewvarを呼んで，1個1個生成させる．これらの変数はC言語で使われる多次元配列といえる．多くの関数で変数の１次元配列が使われるので，BCLでは特別な変数型XPRBarrvarを用意している．この型は変数の1次元配列をそのサイズと一緒に保存しておく．こうすることで，配列を関数に渡すとき，わざわざその配列のサイズを渡す必要がなくなる．詳しくは以下の変数配列の節を参照すること．

変数名は64文字まで使える．もし変数名が指定されなければ，既定の名前('VAR'の後に連番)が与えられる．同じ名前の変数がまた使われたら，BCLは変数名の前に 0. からはじまる連番を付け加える．

Xpress-MPで使える全ての分枝方法は，関数XPRBsetdir によって設定可能である．これは，分枝の優先付け，好ましい分枝方向の選択，擬似コストの定義などを含んでいる．変数への限定はXPRBsetub, XPRBsetlb, XPRBfixvar, XPRBsetlimなどの関数によって定義できる．関数XPRBsetlimは部分整数，半連続，半整数の関数に利用でき，連続領域または半整数領域の下限を設定する．関数XPRBgetbynameは，名前を指定して変数や配列を獲得する．変数に関する情報は，XPRBgetvarname, XPRBgetvartype, XPRBgetcolnum, XPRBgetbounds, XPRBgetlimなどの関数で得られる．関数.XPRBsetvartypeは変数の型を変更する．Figure 2.3 は変数と配列の生成，変更，削除などの関数の概観である．

変数の生成
```
XPRBvar y, s[4];
y = XPRBnewvar(prob,XPRB_PL,"y",1,10);
for(i=0;i<4;i++)
s[i]=XPRBnewvar(prob,XPRB_UI,"st",1,10);
```

Figure 2.3 モデル変数の様々な型，変数の作成，変更，削除，アクセスのための関数

Modeling with BCL 2

変数配列

変数の1次元配列は，多くの関数の入力としてよく利用されるので，BCLはこれのため特別な型を用意している．変数の配列は関数XPRBnewarrvar を呼んで一度に生成する方法と，定義した変数へのポインタ，を1個づつ型XPRBarrvarの配列に登録していく方法がある．

関数XPRBnewarrvarを呼んで変数配列を生成した場合，配列の中の変数は全て同じ型，同じ上下界値を持つ（これは，生成の後で変更することができる）．

変数配列の生成
```
XPRBarrvar av1, av2;
av1=XPRBnewarrvar(prob,5,XPRB_SC,"a1",0,7);
av2=XPRBnewarrvar(prob,3,"a2");
XPRBapparrvarel(prob,av2,y);
XPRBsetarrvarel(prob,av2,2,s[3]);
XPRBendarrvar(prob,av2);
```

変数へのアクセス
```
double ubd, lbd, lim;
XPRBgetvarname(prob,y);
XPRBgetvartype(prob,s[1]);
XPRBgetcolnum(prob,av2[0]);
XPRBgetbounds(prob,y,&lbd,&ubd);
XPRBgetlim(prob,y,&lim);
XPRBsetvartype(prob,av1[1],XPRB_BV);
```

配列へのアクセス
```
XPRBgetarrvarname(prob,av2);
```

```
XPRBgetarrvarsize(prob,av1);
```
変数配列の削除 `XPRBdelarrvar(prob,av2);`
名前による検索
```
XPRBvar y1; XPRBarrvar a1;
y1 = XPRBgetbyname(prob,"y",XPRB_VAR);
a1 = XPRBgetbyname(prob,"a1",XPRB_ARR);
```
分枝の方向
```
XPRBsetdir(prob,s[0],PR,1);
XPRBcleardir(prob);
```
限定の設定
```
XPRBsetlb(prob,y,4);
XPRBsetub(prob,s[0],9);
XPRBfixvar(prob,av[2],6);
XPRBsetlim(prob,y,5);
```
Figure 2.3 モデル変数の様々な型，変数の作成，変更，削除，アクセスのための関数

Modeling with BCL 2

もし変数配列を逐次的に構成していく場合，先に定義しておいた変数（変数配列でもよい）を配列に，任意の順序で追加していく．この場合，配列の定義は最初に関数`XPRBstartarrvar` でモデルと追加する変数のサイズを指定し，関数`XPRBsetarrvarel` で変数を配列内に置くか，関数`XPRBapparrvarel` で配列内の空いている位置を探してそこに置くかする．最後に関数`XPRBendarrvar` を呼んで追加の作業を終わる．例として，4つの連続変数 `s[0]`,..., `s[3]` と2値変数bがあるとしよう．新しい変数配列avを3つの要素から構成する： `av[0] = b`, `av[1] = s[2]`, `av[2] = s[0]`．このように異なる変数をまとめて1つのデータ構造をつくると，制約式の構成やモデルについての情報の獲得が明快に行えるようになる．

変数は複数の配列の要素となることができる（`XPRBsetarrvarel` や`XPRBapparrvarel`によって）．しかし，変数の生成は1回しかできない．

関数`XPRBgetbyname`によって変数配列の名前を指定して，変数配列の情報を獲得できる．また，変数配列の名前を（`XPRBgetarrvarname`によって）獲得できる．そのサイズ，つまり変数の数も（`XPRBgetarrvarsize`によって）獲得できる．

2.3 制約式

基本機能

制約式は特別な制約式関数（以下の**組み込み制約式関数**の項を参照）を呼び出すことで生成するか，または，逐次的に必要な項を制約式に追加していくことで生成する．後の方法の場合，新しい制約式の生成は，最初にXPRBnewctr を呼び制約式の型と（オプションで）名前を指定し，関係する変数と定数項を指定する．関数XPRBaddterm は指定した変数の係数を追加設定する．これに対し，関数XPRBsettermは，既に設定した係数を変更する．関数XPRBaddarrtermを使って，変数配列に対し一度にその係数を追加設定することも可能である．

Figure 2.4は制約式生成の例を示している．

```
XPRBctr ctr
```

$$\sum_{i=0}^{3} s_i$$

$$\_ \; 20 \; .$$

Figure 2.4 制約式の定義を制約式関数で行う場合（左側），係数を逐次追加する場合（右側）


```
XPRBnewsum(prob,"S1",s,XPRB_L,20); ctr=XPRBnewctr(prob,"S1",XPRB_L);
for(i=0;i<=3;i++)
XPRBaddterm(prob,ctr,s[i],1);
XPRBaddterm(prob,ctr,NULL,20);
```

:

$$\sum_{i=0}^{3} D_i s_i \; .$$

$$\_ \; 9 =$$

```
XPRBnewarrsum(prob,"S2",s,D,
XPRB_E,9);
ctr=XPRBnewctr(prob,"S2",XPRB_E);
XPRBaddarrterm(prob,ctr,s,D);
XPRBaddterm(prob,ctr,NULL,9);
```

: ( ):

$$s_0 D_0 + y \; . \; s_0 y - D - 0 \; .$$

```
XPRBnewprec(prob,"Prc",s[0],D[0],y); ctr=XPRBnewctr(prob,"Prc",XPRB_L);
XPRBaddterm(prob,ctr,s[0],1);
XPRBaddterm(prob,ctr,y,-1);
XPRBaddterm(prob,ctr,NULL,-D[0]);
```

Figure 2.4制約式の定義を制約式関数で行う場合（左側），係数を逐次追加する場合（右側）

制約式の定義のためのどの関数も，制約式をモデル名を通して認識するので，入れ子構造の制約式を構成することも可能である．

制約式の名前は64文字まで可能である．もし名前を指定しなければ，自動的に名前をつける（"CTR"で始まりそのあとに連番がつく）．もしも同じ名前の制約式を定義すると，BCLは自動的に名前の前に0.で始める連番をつける．制約式の中で使う変数および変数配列は，前もって定義されておかないといけない．制約式の中でまったく使われない変数は，後でモデルの中から削除される．

制約式が定義された後でも，関数XPRBsetrangeを呼ぶことで，その型を上下限値を設定した区間型に変更することができる．関数XPRBgetbyname によって，制約式名を指定して制約式を獲得することができる．

関数XPRBdeltermによって係数を削除することができる．1つの制約式を丸々削除することもXPRBdelctrによって可能である．更に次のようなことが可能である．制約式名の獲得(XPRBgetctrname)，行列の列番号(XPRBgetrownum)，制約式の型(XPRBgetctrtype)，区間の値(XPRBgetrange, 区間型の制約式にのみ利用可能)，右辺項の値(XPRBgetrhs)．また制約式の型の変更も可能である (XPRBsetctrtype).

制約式はモデルカットに変換することができる(XPRBsetmodcut)．また関数XPRBgetmodcutは制約式がモデルカットで定義されているかどうかを調べることができる．

組み込み制約式関数

上述した逐次的に制約式定義をするための関数のほかにも，BCL は，いくつかの組み込み制約式関数を提供している．関数 XPRBnewarrsum は標準的な線形制約を指定された係数に対して作成する．関数 XPRBnewsum は変数の重みなしの和を作成する．関数 XPRBnewprec はいわゆる優先度制約を作成する．これは，最初の変数に定数を足したものが２番目の変数以下になるという制約である（この制約は，スケジューリング問題でスタート時刻を示す変数間の関係を表すためによく使われるので，この名がある）

目的関数

目的関数(Figure 2.5)は制約式の特別な場合と見なされる．目的関数は他の制約式と同じように定義されるが，通常，制約式タイプをXPRB_Nに選ぶ．しかし，他のタイプにすることも可能である．このように制約式の形で目的関数を与えた場合，制約式中の変数項は目的関数であると同時に，制約としての）等式や不等式は問題の一部として有効となる．目的関数はXRBsetobjで定義される．以前に定義されていた目的関数は，この関数を呼ぶと新しい目的関数と入れ替わる．

目的関数を最小化（こちらがデフォルト）するか最大化するかは，関数XPRBsetsenseで設定する．関数XPRBgetsenseでどちらの設定になっているか確認できる．

目的関数の設定
```
XPRBctr c;
XPRBsetobj(prob,c);
```
目的関数の最小化・最小化の設定
```
XPRBsetsense(prob,XPRB_MAXIM);
```
目的関数の方向の検出
```
int dir;
dir = XPRBgetsense(prob);
```
制約式の取り出し
```
XPRBctr c;
c = XPRBgetbyname(prob,"Sum1",XPRB_CTR);
```
範囲制約の定義
```
XPRBsetrange(prob,c,1,5,15);
```
制約式の削除
```
XPRBdelctr(prob,c);
```
制約式中の項の削除
```
XPRBvar y;
XPRBdelterm(prob,c,y);
```
制約式へのアクセス
```
double bdl, bdu;
XPRBgetctrname(prob,c);
XPRBgetrange(prob,c,&bdl,&bdu);
XPRBgetrownum(prob,c);
XPRBgetctrtype(prob,c);
XPRBsetctrtype(prob,c,XPRB_L);
XPRBgetmcut(prob,c);
XPRBsetmcut(prob,c,1);
```

Figure 2.5 目的関数の定義，制約式の変更とアクセス

全ての解関数(XPRBsolve, XPRBminim, XPRBmaxim)と問題出力関数XPRBexportprobは，目的関数が定義済みであることを要求する．もし目的関数の最小化・最大化が特に設定されていないときは，最小化問題として扱われる．

2.4 問題の解についての情報

モデルの作成と同時に，BCLは問題を解くための関数と解の情報を得るための関数を提供している．Figure 2.6.にそれらをまとめた．より進んだ作業をするためには，（関数XPRBloadmatによって）問題をOptimizerに移してから，利用者はOptimizer ライブラリの関数を利用してほしい．ただし，BCLモデルの中での解情報の参照には，BCLの関数で十分である．.

```
問題を解く XPRBsolve(prob,"dg");
XPRBminim(prob,"pl");
XPRBmaxim(prob,"");
現在の状況 XPRBgetprobstat(prob);
XPRBgetlpstat(prob);
XPRBgetmipstat(prob);
目的関数値 XPRBgetobjval(prob);
解についての情報 XPRBvar y; XPRBctr c;
XPRBgetsol(prob,y); XPRBgetdual(prob,c);
XPRBgetrcost(prob,y); XPRBgetslack(prob,c);
```

Figure 2.6 問題の解法，解の情報

Xpress-BCL Reference Manual Solving and Solution Information 14

Modeling with BCL 2

解を求める関数を呼ぶ前には， XPRBsetobj**によって，必ず目的関数を決めておく．関数** XPRBsolve**は目的関数の方向（最大化か最小化か）が決められていることを要求する．関数** XPRBsolve, XPRBminim, XPRBmaxim**は解法のアルゴリズムをパラメータとして与えることができる．問題が解かれた後，以下のような解の情報を得ることができる；目的関数の最適値** (XPRBgetobjval)**，最適解における変数の値** (XPRBgetsol)**, スラック変数の値**(XPRBgetslack)**, , 被約費用**(XPRBgetrcost)**，双対変数の値**(XPRBgetdual)**.**
問題を解く前や，解いた後の情報を見る前に, 問題や解の状態をチェックすることは有用である（関数 XPRBgetprobstat，XPRBgetlpstat，XPRBgetmipstat**などによる）．モデルの中では定義されているのに制約式の中には出てこない変数とか，係数が全部０の制約式があるかもしれない．こういう場合，問題を**Optimizer**に渡すと，そういう変数や制約を無視して計算する．当該の行や列は負の番号が振られ，いかなる解情報も得られない．**
BCL**では，現在の基底解を保存し，問題を少し変えた（変数や制約を削ったり，追加したり）後で，それを再び読み込むことができる．**
Optimizer**ライブラリ関数を使ったより高度の機能については，**Optimizer Reference Manual**を参照**
Xpress-BCL Reference Manual Further Modeling Topics 15
Further Modeling

Topics 3

3 進んだ使い方

3.1 データ入力とインデックス集合

BCLの利用者は，Cの標準関数を使って，外部データファイルにアクセスし，データを読み込まなければならない．関数XPRBreadarrlineやXPRBreadlineは，Xpress-Modeler使われているのと同じフォーマットでデータを読み込む．関数XPRBreadarrlineは，全て同じフォーマットで書かれた（密な）データテーブルを読み込むのに使う．関数XPRBreadlineは，異なる型のデータ（文字列と数値など）を含むテーブルを読み込む．特に，XPRBreadlineはインデックス集合で番号付けされた疎なデータテーブルを読み込むのに適している．インデックス集合は，大まかに言って，モデルで使うデータを，（番号を使うより）文字列を使って分かりやすい名前をつけるために使われる．（Modeler Reference Manualを参照）．

Data input from file
```
FILE *datafile;
char name[50];
double dval, dvals[5];
XPRBreadline(datafile,200,"T,d",name,&dval);
XPRBreadarrline(datafile,200,"d;",dvals,5);
```
Create a new index set
```
XPRBidx set1;
set1 = XPRBnewidxset(prob,"Set1",100);
```
Add index to a set
```
XPRBaddidxel(prob,set1,"Prob1");
```
Accessing index sets
```
int size, ind;
ind = XPRBgetidxel(prob,set1,"Prod1");
name = XPRBgetidxelname(prob,set1,14);
name = XPRBgetidxsetname(prob,set1);
size = XPRBgetidxsetsize(prob,set1);
```
Figure 3.1 データのファイルからの読み込みとインデックス集合へのアクセス：インデックス集合の作成，要素の追加，要素の獲得，インデックス集合のサイズの獲得．

新しいインデックス集合は関数XPRBnewidxsetによって作成される．集合の要素は，XPRBaddidxelによって追加される．集合の要素は，その名前(XPRBgetidxelを使う)か，その集合内での位置(XPRBgetidxelnameを使う)で獲得できる．データ項目は，複数のインデックス集合の要素になることができる．関数XPRBgetidxsetsizeは現時点でのインデックス集合のサイズ（要素の個数）を返す．

Topics 3

インデックス集合の定義は入れ子にすることができる。つまり，データファイルを読み込む一方で，同時に，複数のインデックス集合に入れることができる．インデックス集合のサイズは，必要に応じ自動的に大きくなる．利用者は，最初にサイズを指定して集合を作成するが，もし，そのサイズに満たない要素しか加えられなかったならば，XPRBgetidxsetsizeは実際の要素数を返す．もしもとより多い要素が加えられたならばサイズはそれに応じて増やされる

3.2 Special Ordered Sets

基本機能

型*n* (*n=1,2*) のSpecial Ordered Sets(SOS)は，そのうちの高々n個が，実行可能整数解において非零であるような変数の集合である．各要素には，実数値（重み）が関連付けられている．その重みが要素間の順序を決める．型2のSOSでは，正となる変数はこの順序でみて隣接していなければならない．

```
XPRBsos set1, set2;
XPRBarrvar s;
Immediate (ref. constraint) XPRBctr c;
set1=XPRBnewsosrc(prob,"sA",XPRB_S2,s,c);
Immediate (coefficients) double C[] = {1,2,3,4};
set2=XPRBnewsosw(prob,"sB",XPRB_S1,s,C);
Consecutive definition set2=XPRBnewsos(prob,"sB",XPRB_S1);
XPRBaddsosarrel(prob,set2,s,C);
Delete set definition XPRBdelsos(prob,set2);
Accessing sets XPRBaddsosel(prob,set2,s[2],4,5);
XPRBdelsosel(prob,set1,s[0]);
XPRBgetsosname(prob,set1);
XPRBgetsostype(prob,set2);
```

Figure 3. SOSの定義とそれへのアクセス: 制約式への参照による即時定義; 逐次追加による定義

BCLでは，SOSはFigure 3.2にあるように，いくつかの方法で定義することができる．配列や制約式の場合と同様に，ある関数を呼んで（下のSOS配列の定義参照）一度の定義することも可能だし，係数を逐次的に追加しながら定義することも可能である．逐次的な定義では， 関数XPRBnewsosが定義の最初に呼ばれる．要素が1つずつ，各時点で対応する係数を指定しながら，関数XPRBaddsoselによって配列に追加され，配列が関数XPRBaddsosarrelによって追加される．

1つの要素，または全集合の削除は，関数XPRBdelsosel，XPRBdelsosによってそれぞれ行われる．BCL はSOSの名前と型を獲得する関数 (XPRBgetsosnameとXPRBgetsostype)も提供している．また，分枝の方向として，優先度や擬コストなどの設定（関数XPRBsetsosdir）を行うことも可能である．

SOS配列の定義

BCL はSOSの生成のために2種類の関数を用意している：XPRBnewsosrc と XPRBnewsosw．どちらの関数でも，新しいSOS は型（1，2）と変数と対応する重み係数（要素間の順序の決定に使われる）を指定して，生成される．関数XPRBnewsosrcを使う場合は，これらの係数は，指定した制約式中で変数がもつ係数の値からとられる。関数XPRBnewsoswを使う場合は，利用者が直接指定する．

3.3 出力と印刷

BCLは，変数，制約式，SOS，インデックス集合それぞれをプリントする関数(XPRBprintvar, XPRBprintarrvar, XPRBprintctr, XPRBprintsos, XPRBprintidxset)を提供し，また，モデル定義をプリントする関数(XPRBprintprob)も提供している．どのプログラムの出力も関数XPRBprintfによって，C言語のprintfとよく似た方法で，プリントされる． 以上の全ての関数の出力は，コールバックXPRBdefcbmsgが定義されていれば，この関数により割り込みを受ける．

拡張LP形式や拡張MPS形式の行列形式で問題をファイルに出力すること（XPRBexportprob）も可能である．注意することは，標準LP形式と違い，拡張LP形式は，SOSやその他の非標準的変数の型）準連続，半整数，部分整数）をサポートしていることである．標準LP形式と同様に，目的関数の方向を指定することが必要である．

ファイル出力
```
XPRBexportprob(prob,XPRB_MPS,"expl2");
```
モデルの各要素のプリント
```
XPRBvar y;
XPRBprintvar(prob,y);
XPRBarrvar av;
XPRBprintarrvar(prob,av);
XPRBctr c;
XPRBprintctr(prob,c);
XPRBsos s;
XPRBprintsos(prob,s);
XPRBidxset i;
XPRBprintidxset(prob,i);
```
問題のプリント
```
XPRBprintprob(prob);
```
プリント出力
```
XPRBprintf("Print this text");
```
名前の文字列の作成
```
int i = 3;
XPRBnewname(prob,"abc%d",i);
```

Figure 3.3 ファイル出力とプリント

Topics 3

3.4 BCLによる2次計画問題

LPやMIPの拡張として，BCLは2次計画問題 (QP) の定式化および解法の提供をしている．2次計画問題は，線形の制約式と，2次形式の目的関数

$c^T x + x^T Q x$

をもつ．xは決定変数，cはコストベクトル，Qは2次のコスト係数の行列であり，対称でなければならない．さらに，問題が最小化であれば半正定値，最大化であれば半負定値でなければならない．これは，Xpress-Optimizerは凸2次計画問題を解くようにできているからである．もし問題が凸でない場合は，アルゴリズムは収束しないか，局所最適解に収束するかである．

2次項の追加
```
XPRBvar x1;
XPRBaddqterm(prob,x1,x1,3);
```
2次項の作成
```
XPRBvar x2;
XPRBsetqterm(prob,x1,x2,-7.2);
```
2次項の削除
```
XPRBdelqobj(prob);
```
Figure 3.4 目的関数の定義とアクセス.

Further Modeling

Topics 3

BCLでは，制約式の定義と同様に，目的関数の2次項は項別に定義される．2次項の係数は，指定した値に設定されるか(XPRBsetqtermを使う)， 指定した値だけ増加される(XPRBaddqterm)．目的関数の2次項は1次項とは別に扱われるので，目的関数が削除されたり変更されたりするときは，関数XPRBdelqobjを呼んで削除しなければならない．目的関数の2次項を定義する前に，変数のほうの定義を先にしておくこと．
BCLが学生版で使われていないならば， 関数XPRBprintprob, XPRBexportprob, and XPRBprintctrによって，問題の完全な定義（2次目的関数を含む）を出力することができる．

Example

l以下の例は，次の2次目的関数を定義する．

$-6x0 + 2x0^2 - 2x0x1 + x^2$

ここでは，変数xiすでに定義されているものとする．

```
XPRBctr cobj;
XPRBvar x[2];
...
cobj = XPRBnewctr(prob,"OBJ",XPRB_N); /* create the objective
constraint */
XPRBaddterm(prob,cobj, x[0],-6); /* add the linear term */
XPRBsetobj(prob,cobj); /* choose the objective function */
XPRBaddqterm(prob,x[0],x[0],2); /* add quadratic terms */
XPRBaddqterm(prob,x[0],x[1],-2);
XPRBaddqterm(prob,x[1],x[1],1);
```

Xpress-BCL Reference Manual Quadratic Programming with BCL 20

Further Modeling

Topics 3

Xpress-BCL Reference Manual BCL Library Functions 21

4 BCL ライブラリ関数

Xpress-MP Builder Component Library (BCL)では，多くのルーチンが利用可能である．それらは，問題の定義や求解のための簡単なルーチンから，Xpress-Optimizer ライブラリと連携するための関数まで幅広く用意されている．
BCLにはモデル識別のため以下のような型が用意されている．

4.1 関数記述のための表記

この章で説明する関数は，以下のようなヘッダーをつけて記述されている．

関数名
各ルーチンは，ページの最初にその名称を記述している．

目的
各ルーチンの目的を簡単に記述している．

概要
使い方の概要を示す．オプションの引数やフラグは，もし必要なければ，NULLとして指定される．
オプションの引数やフラグについては，以下の**引数**や**更なる情報**の項で記述される．
`XPRBprob` 問題の定義;
`XPRBvar` 変数;
`XPRBarrvar` 一次元の配列，要素は`XPRBvar`;
`XPRBctr` 制約式;
`XPRBsos` a Special Ordered Set (SOS1 of SOS2);
`XPRBidxset` an index set;
`XPRBbasis` 基底.

引数
引数のリストが与えられる．引数がとりうる値についても記述される．.

例
１，２の例題が与えられ，具体的な使い方の説明をする．

追加情報
ルーチンについての追加情報が与えれる．

関連事項
関連するルーチンや情報が与えらる．

XPRBaddarrterm

目的

LPの制約式に複数の項を追加する．

概要

```
int XPRBaddarrterm(XPRBprob prob, XPRBctr ctr,
XPRBarrvar av, double *coeff);
```

引数

prob 問題

ctr 制約式

av 変数の配列

coeff 各変数への係数の値（係数の数は対応する変数の配列のサイズと一致しなければならない）

例

以下の例は

```
2.0*arry1000 + 13.0*arry1001 + 15.0*arry1002 +
6.0*arry1003 +8.5*arry1004
```

を制約式 ctr1 に追加する．

```
XPRBctr ctr1;
XPRBarrvar ty1;
...
double cr[] = {2.0, 13.0, 15.0, 6.0, 8.5};
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
XPRBaddarrterm(prob, ctr1, ty1, cr);
```

追加情報

この関数は複数の項を制約式に追加する．追加される項は，変数の配列 av と対応する係数coeff から作成する．制約式が，既にこれから追加しようとする変数を含んでいる場合，その変数に対応するcoeff で指定される係数は，もとの係数に加算される．

関連事項

XPRBaddterm, XPRBdelctr, XPRBdelterm, XPRBnewctr.

XPRBaddidxel 4

XPRBaddidxel

目的

インデックス集合にインデックスを追加する.

概要

```
int XPRBaddidxel(XPRBprob prob, XPRBidxset idx,
char *name);
```

引数

prob 問題

idx BCL インデックス集合

name 追加するインデックス名

例

以下の例では，100のサイズをもつインデックス集合を定義し，新しいインデックスを追加している．追加したインデックスのインデックス集合中での連番を獲得している

```
XPRBidxset iset;
int val;
...
iset = XPRBnewidxset(prob,"Set",100);
val = XPRBaddidxel(prob, iset, "first");
```

追加情報

この関数は，定義済みのインデックス集合にインデックスを追加する．新しいインデックスは，同じインデックスが既にインデックス集合に定義されていると，追加されない．インデックスが追加された場合でも，既に定義されているため追加されない場合でも，返り値は，指定したインデックスのインデックス集合上での連番である．通常のCの規約と同様，インデックスの連番は0から始まる.

関連事項

XPRBgetidxel, XPRBnewidxset.

XPRBaddqterm

目的

目的関数に2次の項を追加する.

概要

```
int XPRBaddqterm(XPRBprob prob, XPRBvar var1, XPRBvar var2,
double coeff);
```

引数

prob 問題

var1 変数1

var2 変数2(var1と同じでもよい).

coeff var1 * var2の係数

例

以下の例は -2*x2*x4 を目的関数に追加する.

```
XPRBvar x2,x4;
...
x2 = XPRBnewvar(prob,XPRB_PL,"abc1",0,XPRB_INFINITY);
x4 = XPRBnewvar(prob,XPRB_PL,"abc5",0,XPRB_INFINITY);
XPRBaddqterm(prob,x2,x4,-2);
```

追加情報

この関数は, 係数をcoeff* var1 * var2なる2次の項を目的関数に追加する. 目的関数が既にvar1 * var2なる2次の項を含む場合, その係数にあらたに指定した係数coeffを加算する.

関連事項

XPRBdelqobj, XPRBsetqterm.

XPRBaddsosarrel

目的

SOSに複数の要素を追加する.

概要

```
int XPRBaddsosarrel(XPRBprob prob, XPRBsos sos,
XPRBarrvar av, double *weight);
```

引数

例

以下では，SOS set1に重みcrをつけた配列ty1を追加する．

```
XPRBsos set1;
XPRBarrvar ty1;
double cr[] = {2.0, 13.0, 15.0, 6.0, 8.5};
...
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
set1 = XPRBnewsos(prob, "sos1", XPRB_S1);
XPRBaddsosarrel(prob, set1, ty1, cr);
```

追加情報

この関数は，変数の配列と対応する重みをSOSに追加する．もし変数が既にその集合に存在していれば，指定した重みは既存の重みに足しこまれる．全ての重みは，0以外の値を設定しなければならないことに注意する．

関連事項

XPRBaddsosel, XPRBdelsos, XPRBdelsosel, XPRBnewsos.

prob **対象としている問題**

sos A SOS of type 1 or 2.

av **変数の配列**

weight **重みの配列．重みの数は対応する変数配列の数と一致しなければいけない**

XPRBaddsosel

目的

1つの要素をSOSに追加する.

概要

```
int XPRBaddsosel(XPRBprob prob, XPRBsos sos, XPRBvar var,
double *weight);
```

引数

例

```
XPRBsos set1;
XPRBvar x2;
...
x2 = XPRBnewvar(prob,XPRB_PL,"abc1",0,XPRB_INFINITY);
set1 = XPRBnewsos(prob,"sos1",XPRB_S1);
XPRBaddsosel(prob,set1,x2,9.0);
```

This adds a variable `x2` with weight `9.0` to the SOS `set1`.

追加情報

この関数は，1つの変数と対応する重みをSOSに追加する．もし変数が既にその集合に存在していれば，指定した重みは既存の重みに足しこまれる．重みは，0以外の値を設定しなければならないことに注意する．

関連事項

`XPRBaddsosarrel`, `XPRBdelsos`, `XPRBdelsosel`, `XPRBnewsos`.

`prob` **現在の問題**

`sos` A SOS of type 1 or 2.

`var` **変数**

`weight` **変数に対応する重み.**

XPRBaddterm

目的

項を制約式に追加する．

概要

```
int XPRBaddterm(XPRBprob prob, XPRBctr ctr, XPRBvar var,
double coeff);
```

引数

例

```
XPRBctr ctr1;
XPRBvar x1;
...
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
XPRBaddterm(prob, ctr1, x1, 5.4);
```

This adds the term `5.4*x1` to the constraint `ctr1`.

追加情報

この関数は，新しい項を制約式に追加する．項は，変数`var`**と係数**`coeff`**.で指定する．もし制約式中にすでに変数**`var`**があれば，係数**`coeff`** はもとの係数に足しこまれる．変数**`var`**を**`NULL`**に設定した場合，係数**`coeff`** は制約式の右辺に足される．**

関連事項

`XPRBaddarrterm`, `XPRBdelctr`, `XPRBdelterm`, `XPRBnewctr`, `XPRBsetterm`.

`Prob` **現在の問題.**

`ctr` **制約式への参照，**`XPRBnewctr`**の返り値.**

`var` **変数への参照．**`NULL`**の場合もある．**

`coeff` **変数**`var`**に対する係数.**

XPRBapparrvarel

目的

要素を変数配列に追加する.

概要

```
int XPRBapparrvarel(XPRBprob prob, XPRBarrvar av,
XPRBvar var);
```

引数

例

次の例は変数x1を配列av2の最初の空き位置に挿入する.

```
XPRBarrvar av2;
XPRBvar x1;
...
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
av2 = XPRBstartarrvar(prob, 5, "arr2");
XPRBapparrvarel(prob, av2, x1);
```

追加情報

この関数は変数を配列の最初の空いている位置に挿入する.

関連事項

XPRBdelarrvar, XPRBendarrvar, XPRBnewarrvar, XPRBsetarrvarel, XPRBstartarrvar.

prob **現在の問題**

av **配列への参照**

var **追加する変数T**

XPRBcleardir

目的

設定されたすべての探索方向を解除する.

概要

```
int XPRBcleardir(XPRBprob prob);
```

引数

例

```
XPRBprob expl2;
expl2 = XPRBnewprob("example2");
XPRBcleardir(expl2);
```

以下では，問題expl2に設定された全ての探索方向を解除する.

関連事項

XPRBsetdir, XPRBsetsosdir.

prob **現在の問題**.

XPRBdefcberr 4

XPRBdefcberr

目的

利用者の定義したエラー処理のコールバック

概要

```
int XPRBdefcberr(XPRBprob prob,
void (*usererr)(XPRBprob my_prob, void *my_object,
int errnum, int type, char *errtext), void *object);
```

引数

例

この例では, エラーを表示した後, もし重大なエラーならば終了するというエラー処理の関数myerr**を定義した後，エラー処理のコールバックとして設定している.**

```
void myerr(XPRBprob my_prob, void *my_object, int num,
int type, char *t)
{
printf("BCL error %d: %s\n", num, t);
if(type == XPRB_ERR) exit(0);
}
...
XPRBdefcberr(prob,myerror,NULL);
```

追加情報

この関数は，BCL**の返すエラー番号とエラーメッセージを返すエラー処理のコールバックを定義する．**BCL**のエラーの一覧は**5**章BCL Error Messages**を参照．もしエラーメッセージや警告メッセージの出力が可能ならば（**XPRBsetmsglevel **を参照），この関数が呼ばれたのち，それらが出力される.**

prob **現在の問題**

usererr **利用者定義のエラー処理関数**

my_prob **コールバック関数に渡される問題のポインタ**.

my_object **コールバックに渡される利用者定義のオブジェクト**.

errnum **エラー番号**

type **エラーのタイプ．以下のうちのどれか.**

XPRB_ERR fatal error;

XPRB_WAR warning.

errtext **エラーメッセージのテキスト文**.

object **コールバックに渡される利用者定義のオブジェクト**..

XPRBdefcberr 4

関連事項

XPRBdefcbmsg, XPRBgetversion, XPRBseterrctrl.

XPRBdefcbmsg 4

XPRBdefcbmsg

目的

プリント出力のためコールバック

概要

```
int XPRBdefcbmsg(XPRBprob prob,
void (*userprint)(XPRBprob my_prob, void *my_object,
char *msgtext), void *object);
```

引数

例

以下の例は，プリント出力関数をコールバックに定義する．.

```
void myprint(XPRBprob prob, void *my_object, char *msg);
{
printf("BCL output: %s\n", msg);
}
...
XPRBdefcbmsg(prob,myprint,NULL);
```

追加情報

この関数はXPRBsetmsglevelの設定による全てのメッセージを返すコールバック関数を定義する．メッセージには，警告やエラーなどBCLとOptimiezerが返す全てのメッセージを含む．このコールバックは通常のメッセージプリント出力とは独立に，XPRBprintfを利用したユーザのプログラムプリント出力も返すことができる．もしこのコールバックが定義されなければ，すべてのプログラム出力は標準出力にプリントされる．ただし，警告とエラーは，標準エラー出力のほうに出力される．

prob **現在の問題.**

userprint **利用者定義のメッセージ処理関数**

my_prob **コールバック関数にわたされる問題**

my_object **コールバック関数に渡される利用者定義のオブジェクト**

msgtext **メッセージ文.**

object **コールバック関数に渡される利用者定義のオブジェクト**

XPRBdefcbmsg 4

関連事項

`XPRBdefcberr`, `XPRBsetmsglevel`.

XPRBdelarrvar 4

XPRBdelarrvar

目的

変数配列の削除する.

概要

```
int XPRBdelarrvar(XPRBprob prob, XPRBarrvar av);
```

引数

例

```
XPRBarrvar av2;
...
av2 = XPRBstartarrvar(prob,5,"arr2");
XPRBendarrvar(prob,av2);
XPRBdelarrvar(prob,av2);
```

この例は配列`av2`**を削除する．ただし，その配列に入っていた変数そのものは削除しない.**

追加情報

この関数は配列への参照を削除する．配列は制約式を定義するため補助的に作成されることがあるので，そういうときは，配列をずっと持っている必要はない．もし配列がそのモデルの中だけで使われているならば，この関数を呼んで削除し，使用していたメモリーを解放できる．配列が，もし `XPRBstartarrvar` **で作成されていたならば，配列への参照のみが削除され，そこに属していた変数は削除されない.**

関連事項

`XPRBapparrvarel`, `XPRBendarrvar`, `XPRBnewarrvar`, `XPRBsetarrvarel`, `XPRBstartarrvar`.

`prob` **現在の問題**

`av` **モデル中での配列への参照**

XPRBdelbasis 4

XPRBdelbasis

目的

以前にセーブした基底解を削除する．

概要

```
void XPRBdelbasis(XPRBprob prob, XPRBbasis basis);
```

引数

例

以下の例は行列を変更する前に基底解を保存することを例示したものである．その後，古い基底解はもう一度読み込まれてから削除される．

```
XPRBprob expl2;
XPRBbasis basis;
expl2 = XPRBnewprob("example2");
...
XPRBsolve(expl2,"l");
basis = XPRBsavebasis(expl2);
...
XPRBloadmat(expl2);
XPRBloadbasis(expl2,basis);
XPRBdelbasis(expl2,basis);
XPRBsolve(expl2,"l");
```

追加情報

この関数は，以前にXPRBsavebasis**を使ってセーブした基底解を削除する．通常は使わなくなった基底は削除しておくべきである．**

関連事項

XPRBloadbasis, XPRBsavebasis.

prob **現在の問題**

basis **以前にセーブされた基底解**.

XPRBdelctr

例

制約式を削除する．

概要

```
int XPRBdelctr(XPRBprob prob, XPRBctr ctr);
```

引数

例

```
XPRBctr ctr1;
...
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
XPRBdelctr(ctr1);
```

この例は制約式ctr1**を削除する．**

追加情報

現在の問題から制約式を削除する．もしその削除する制約式が目的関数に選ばれていると，目的関数はNULL**になる．**

関連事項

XPRBnewctr, XPRBgetctrname, XPRBgetctrtype, XPRBsetctrtype.

prob **現在の問題**.

ctr **制約式への参照**.

XPRBdelprob

目的

問題を削除する．

概要

```
int XPRBdelprob(XPRBprob prob);
```

引数

例

この例では問題`expl2` **が削除される．**

```
XPRBprob expl2;
expl2 = XPRBnewprob("example2");
XPRBdelprob(expl2);
```

追加情報

この関数はBCL上の問題を削除する．問題の削除後もOptimizerに読み込まれた行列は削除されたない．削除後も全てのパラメータの設定は有効なままである．新しい問題を作成するときは`XPRBnewprob`**を使う．**

関連事項

`XPRBgetXPRSprob`, `XPRBnewprob`, `XPRSdestroyprob` (see Optimizer Reference Manual).

`prob` **現在の問題**

XPRBdelqobj 4

XPRBdelqobj

目的

目的関数中の2次項を削除する．

概要

```
int XPRBdelqobj(XPRBprob prob);
```

引数

例

この例では，目的関数中の全ての2次項が削除される．

```
XPRBvar x2;
.
x2 = XPRBnewvar(prob,XPRB_PL,"abc1",0,XPRB_INFINITY);
XPRBaddqterm(prob,x2,x2,5.2);
XPRBdelqobj(prob);
```

追加情報

この関数は，現問題の目的関数中の２次項を削除する．もし（XPRBdelctr **や** XPRBsetobj**を用いて）既に目的関数が削除されていたり変更されていた場合は，2次項は削除されないことに注意せよ．**

関連事項

XPRBaddqterm, XPRBsetqterm.

prob **現在の問題**

XPRBdelsos

目的

SOSを削除する.

概要

```
int XPRBdelsos(XPRBprob prob, XPRBsos sos);
```

引数

例

The following deletes the SOS `set1`.

```
XPRBsos set1;
...
set1 = XPRBnewsos(prob, "sos1", XPRB_S1);
XPRBdelsos(prob, set1);
```

追加情報

この関数はSOSを削除する．SOSを構成する変数は削除しない.

関連事項

`XPRBaddsosarrel`, `XPRBaddsosel`, `XPRBdelsosel`, `XPRBnewsos`.

`prob` 現在の問題

`sos` 既定義のタイプ１または２のSOS への参照.

XPRBdelsosel 4

XPRBdelsosel

目的

SOSから，要素を一つ削除する．

概要

```
int XPRBdelsosel(XPRBprob prob, XPRBsos sos, XPRBvar var);
```

引数

例

以下の例は変数 x2 を SOS set1から削除する．

```
XPRBsos set1;
XPRBvar x2;
...
x2 = XPRBnewvar(prob,XPRB_PL,"abc1",0,XPRB_INFINITY);
set1 = XPRBnewsos(prob,"sos1",XPRB_S1);
XPRBaddsosel(prob,set1,x2,9.0);
XPRBdelsosel(prob,set1,x2);
```

追加情報

この関数は，SOS（Special Ordered Set）から，要素を一つ削除する．

関連事項

XPRBaddsosarrel, XPRBaddsosel, XPRBdelsos, XPRBnewsos.

prob 現在の問題.

sos A SOS of type 1 or 2.

var 変数への参照

XPRBdelterm 4

XPRBdelterm

目的

制約式から項を削除する.

概要

```
int XPRBdelterm(XPRBprob prob, XPRBctr ctr, XPRBvar var);
```

引数

例

以下の例は変数 x1を制約式から削除する.

```
XPRBctr ctr1;
XPRBvar x1;
...
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
XPRBaddterm(prob, ctr1, x1, 5.4);
XPRBdelterm(prob, ctr1, x1);
```

追加情報

この関数は，指定した制約式から項を削除する.

関連事項

XPRBaddarrterm, XPRBaddterm, XPRBdelctr, XPRBnewctr, XPRBsetterm.

prob 現在の問題

.

ctr 既定義の制約式への参照

var 変数への参照

XPRBendarrvar 4

XPRBendarrvar

目的

変数配列の定義を終了する.

概要

```
int XPRBendarrvar(XPRBprob prob, XPRBarrvar av);
```

引数

例

```
XPRBarrvar av2;
...
av2 = XPRBstartarrvar(prob,5,"arr2");
XPRBendarrvar(prob,av2);
```

この例では，配列 av2**の定義を終了する.**

追加情報

この関数は変数配列の定義を終了する．この関数では，他の配列を追加的に定義する関数と同様に，配列への参照を引数に必要とするので，同時にいくつかの配列を定義することも可能である.

関連事項

XPRBdelarrvar, XPRBnewarrvar, XPRBstartarrvar.

prob **現在の問題**

av **配列への参照**

XPRBexportprob

目的

問題の行列をファイル出力する．

概要

```
int XPRBexportprob(XPRBprob prob, int format,
char *filename);
```

引数

例

```
XPRBprob expl2;
expl2 = XPRBnewprob("example2");
XPRBexportprob(expl2, XPRB_MPS, "ex2");
```

これは，問題の行列をMPS形式でファイルex2.matに出力する．

追加情報

この関数は，問題の行列を拡張LP形式，または拡張MPS形式にしてファイルに出力する．拡張子は，LP形式に対しては.lpが，MPS形式に対しては.matが自動的に当てられるので，利用者は，拡張子を指定してはいけない．この関数はstudent versionでは利用できない．

関連事項

XPRBprintprob, XPRBprintf.

prob **現在の問題**

format **行列の出力フォーマット形式．以下のどちらかを指定する．**

XPRB_LP **LPファイル形式**;

XPRB_MPS **MPSファイル形式**.

filename **出力ファイル名．拡張子は書かない。**

XPRBfixvar

目的

変数の値を固定する.

概要

```
int XPRBfixvar(XPRBprob prob, XPRBvar var, double val)
```

引数

例

```
XPRBvar x1;
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 1, 100);
XPRBfixvar(prob, x1, 20.0);
```

これは変数x1**を**20**に設定する**.

追加情報

この関数は変数の値を指定値に固定する．この関数の呼び出しは，関数XPRBsetub **と** XPRBsetlb.
の呼び出しで置き換えられる．値 val **は当該変数のもともとの上下界から外れていてもよい．**

関連事項

XPRBgetbounds, XPRBgetlim, XPRBsetlb, XPRBsetlim, XPRBsetub.

prob **現在の問題**
var **変数への参照**
val **変数の固定値**

XPRBfree 4

XPRBfree

目的

開いていたファイルを全部クローズする．

概要

```
int XPRBfree(void);
```

引数

なし

例

以下は，BCL**のセッションの最後で行う例である．**

```
XPRBdestroyprob(prob);
XPRBfree();
return 0;
```

追加情報

重要な点．XPRBfree**は問題に付随するメモリーの解放は行わない．メモリーの解放は，**XPRBdestroyprob **により行う．**

関連事項

XPRBdestroyprob, XPRBinit, XPRSfree (Optimizer Reference Manual**を参照せよ**).

XPRBgetarrvarname 4

XPRBgetarrvarname

目的

変数の配列の名前を獲得する.

概要

```
char *XPRBgetarrvarname(XPRBprob prob, XPRBarrvar av);
```

引数

prob **現在の問題**

av **変数の配列への参照.**

例

```
XPRBarrvar ty1;
...
ty1 = XPRBnewarrvar(prob, 10, XPRB_PL, "arry1", 0, 500);
printf("%s\n",XPRBgetarrvarname(ty1));
```

この例は配列の名前arry1**をプリント出力する.**

追加情報

この関数は変数の配列の名前を返す．もし名前が定義されていないときは，返り値は NULL**である.**

関連事項

XPRBdelarrvar, XPRBgetarrvarsize, XPRBnewarrvar.

XPRBgetarrvarsize 4

XPRBgetarrvarsize

目的

変数の配列のサイズを獲得する.

概要

```
int XPRBgetarrvarsize(XPRBprob prob, XPRBarrvar av);
```

引数

prob **現在の問題**

av **変数の配列への参照**

例

```
XPRBarrvar ty1;
int tsize;
...
ty1 = XPRBnewarrvar(prob, 10, XPRB_PL, "arry1", 0, 500);
tsize = XPRBgetarrvarname(prob, ty1);
```

配列 ty1**のサイズを返す.**

追加情報

この関数は変数の配列のサイズ（要素の個数）を返す. もし配列に変数が逐次的に追加されていた場合は, 返り値は配列の定義時の最大サイズより小さいこともある. 返り値は実際に配列に追加された変数の数を表している.

関連事項

XPRBdelarrvar, XPRBgetarrvarname, XPRBnewarrvar.

XPRBgetbounds

目的

変数の上下限値を獲得する.

概要

```
int XPRBgetbounds(XPRBprob prob, XPRBvar var, double *bdl,
double *bdu);
```

引数

var 変数への参照

bdl 下限値. NULLを与えると，なにも値を返さない.

Bdu 上限値. NULLを与えると，なにも値を返さない.

例

```
XPRBvar x1;
double ubound;
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
XPRBgetbounds(prob, x1, NULL, &ubound);
```

この例は変数x1の上限値を獲得する.

追加情報

この関数は，変数の現時点での上下限値を返す. bdlまたはbdu をNULLに設定したときは，**値を返さない.**

関連事項

XPRBfixvar, XPRBgetlim, XPRBsetlb, XPRBsetlim, XPRBsetub.

prob 現在の問題

XPRBgetbyname 4

XPRBgetbyname

目的

名前で，対象を獲得する．

概要

```
void *XPRBgetbyname(XPRBprob prob, char *name, int type);
```

引数

prob **現在の問題**

name **対象の名前**

type **対象の型．以下のうちのどれか．**

XPRB_VAR **変数**;

XPRB_ARR **変数の配列**;

XPRB_CTR **制約式**;

XPRB_SOS SOS;

XPRB_IDX **インデックス集合**

例

この例はabc3**という名前の変数を獲得する．**

```
XPRBvar x1;
x1 = XPRBgetbyname(prob,"abc3",XPRB_VAR);
```

追加情報

この関数は，指定された型の対象への参照を返す．もし該当する対象がなければNULL**を返す． 一つの問題の中で，異なる型の対象が同じ名前をもつことは可能である．**

関連事項

XPRBgetXPRSprob.

XPRBgetcolnum

目的

変数の対応する列番号を獲得する.

概要

```
int XPRBgetcolnum(XPRBprob prob, XPRBvar var);
```

引数

prob **現在の問題.**

var **変数への参照**

例

```
XPRBvar x1;
int vindex;
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
vindex = XPRBgetcolnum(prob, x1);
```

この例は変数x1**の対応する列番号を返す.**

追加情報

この関数は，現時点で Xpress-Optimizer**にロードされている行列中で，指定した変数に対応する列番号を返す．指定した変数の対応列が行列中にない場合や，行列がまだ生成されていない場合は，負の番号が返される．行番号は**0**から始まる.**

関連事項

XPRBgetvarname, XPRBgetvartype, XPRBsetvartype.

XPRBgetctrname 4

XPRBgetctrname

目的

制約式の名前を返す.

概要

```
char *XPRBgetctrname(XPRBprob prob, XPRBctr ctr);
```

引数

`prob` **現在の問題**

`ctr` **制約式への参照**

例

```
XPRBctr ctr1;
...
ctr1 = XPRBnewctr("r1", XPRB_E);
printf("%s\n",XPRBgetctrname(ctr1));
```

この例は"r1" **をプリントする.**

追加情報

この関数は制約式の名前を返す. もし利用者が名前をつけていなければな, BCLが自動生成した名前が返される.

関連事項

XPRBgetctrtype, XPRBnewctr, XPRBsetctrtype.

XPRBgetctrtype

目的

制約式の型を返す.

概要

```
int XPRBgetctrtype(XPRBprob prob, XPRBctr ctr);
```

引数

`prob` **現在の問題**

`ctr` **制約式への参照.**

例

以下の例は制約式`ctr1`**の型を返す.**

```
XPRBctr ctr1;
...
char rtype;
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
rtype = XPRBgetctrtype(prob, ctr1);
```

追加情報

この関数は以下に示すような制約式の型を返す.

`XPRB_L` **以下**;
`XPRB_G` **以上**;
`XPRB_E` **等号**;
`XPRB_N` **制約なし（目的関数）**;
`XPRB_R` **上下限制約**.

関連事項

`XPRBgetctrname`, `XPRBnewctr`, `XPRBsetctrtype`.

XPRBgetdual

目的

双対変数の値を返す.

概要

```
double XPRBgetdual(XPRBprob prob, XPRBctr ctr)
```

引数

`prob` **現在の問題**

`ctr` **制約式への参照**

例

```
XPRBprob expl2;
XPRBctr ctr2;
XPRBarrvar ty1;
double dval;
...
expl2 = XPRBnewprob("example2");
ty1 = XPRBnewarrvar(expl2, 5, XPRB_PL, "arry1", 0, 500);
ctr2 = XPRBnewsum(expl2, "r2", ty1, XPRB_E, 9);
XPRBsolve(expl2, "l");
dval = XPRBgetdual(expl2, ctr2);
```

この例は制約式`ctr2`**の双対変数の値を返す.**

追加情報

この関数は制約式の双対変数の値を返す．ある制約式が問題の中で有効になっているかどうか，利用者は，まずその行番号が非負であることによって確かめられる．この関数は最適解の探索が終了し整数解が見つかったとき（関数`XPRBgetmipstat`**が**4**または**6**を返したとき）は，その最適整数解における双対変数値を返す．整数解が見つからなかったときは，最後に解いた**LP**の双対変数値を返す．注意することは，この関数は最適化の計算の途中では使えないことである．（例えば，**Optimizer **ライブラリのコールバック関数）**

関連事項

`XPRBgetobjval`, `XPRBgetrcost`, `XPRBgetslack`, `XPRBgetsol`.

XPRBgetidxel

目的

インデックス中の番号を返す.

概要

```
int XPRBgetidxel(XPRBprob prob, XPRBidxset idx,
char *name);
```

引数

prob **現在の問題**

idx **インデックス集合**

name **インデックス集合中の要素の名前**

例

```
XPRBidxset iset;
int val;
...
iset = XPRBnewidxset(prob,"Set",100);
XPRBaddidxel(prob, iset, "first");
val = XPRBgetidxel(prob, iset, "first");
```

この例は100**個の要素を有するインデックス集合**iset**を定義し，それにインデックス**first**を追加し、その後，そのインデックス集合中の連番を獲得する.**

追加情報

インデックス集合中の要素は，その番号と名前のどちらでもアクセスできる．この関数は名前を指定することで，番号を返す.

関連事項

XPRBaddidxel, XPRBnewidxset.

XPRBgetidxelname 4

XPRBgetidxelname

目的

インデックスの名前を返す.

概要

```
char *XPRBgetidxelname(XPRBprob prob, XPRBidxset idx,
int i);
```

引数

Prob **現在の問題**

idx **インデックス集合**

i **インデックスの番号**

例

```
XPRBidxset iset;
char name[100];
...
iset = XPRBnewidxset(prob,"Set",100);
name = XPRBgetidxelname(prob, iset, 0);
```

この例はインデックス集合iset**を**100**個の要素のもたせて定義し，そのインデックス集合の中の**0**番目の要素の名前を獲得する.**

追加情報

インデックス集合中の要素は，その番号と名前のどちらでもアクセスできる．この関数は番号を指定することで，名前を返す.

関連事項

XPRBgetidxsetname, XPRBgetidxsetsize, XPRBnewidxset.

XPRBgetidxsetname 4

XPRBgetidxsetname

目的

インデックス集合の名前を獲得する.

概要

```
char *XPRBgetidxsetname(XPRBprob prob, XPRBidxset idx);
```

引数

prob **現在の問題**

idx **インデックス集合**

例

この例は100**個の要素を有するインデックス集合**iset**を定義し，その名前を獲得する.**

```
XPRBidxset iset;
char name[100];
...
iset = XPRBnewidxset(prob,"Set",100);
name = XPRBgetidxsetname(prob,iset);
```

追加情報

この関数は，インデックス集合の名前を獲得する.

関連事項

XPRBgetidxelname, XPRBgetidxsetsize, XPRBnewidxset.

XPRBgetidxsetsize 4

XPRBgetidxsetsize

目的

インデックス集合のサイズを獲得する．

概要

```
int XPRBgetidxsetsize(XPRBprob prob, XPRBidxset idx);
```

引数

prob **現在の問題**

idx **インデックス集合**

例

この例はインデックス集合iset**に**100**個分のスペースをもたせて定義し，そのサイズを獲得する．**

```
XPRBidxset iset;
int size;
...
iset = XPRBnewidxset(prob,"Set",100);
size = XPRBgetidxsetsize(prob,iset);
```

追加情報

1. **この関数は，インデックス集合中の実際の要素数を返す．返り値は必ずしも最初に指定したインデックス集合のサイズに一致しない．最初の指定したサイズより少ない要素しかインデックス集合に追加していなかったり，最初に指定したサイズよりたくさん要素を追加している（この場合サイズは自動的に増える）と，実際の要素数が返り値になるので，不一致がおこる．**
2. **この関数はインデックス集合のサイズを返す．返り値－１はエラーを意味する．**

関連事項

XPRBaddidxel, XPRBgetidxelname, XPRBgetidxsetname, XPRBnewidxset.

XPRBgetlim 4

XPRBgetlim

目的

部分整数変数の整数限界，または準連続変数の準連続整数変数の準連続限界を獲得する．

概要

```
int XPRBgetlim(XPRBprob prob, XPRBvar var, double *lim);
```

引数

prob **現在の問題.**

var **変数への参照**

lim **限界値**

例

```
XPRBvar x3;
double vlim;
...
x3 = XPRBnewvar(prob, XPRB_SC, "abc4", 0, 50);
XPRBgetlim(prob, x3, &vlim);
```

この例は変数x3**の連続部分の下限値を獲得する．**

追加情報

この関数は，部分整数変数の整数限界値または，準連続変数または準連続整数変数の下準連続限界値を返す．

関連事項

XPRBfixvar, XPRBgetbounds, XPRBsetlb, XPRBsetlim, XPRBsetub.

XPRBgetlpstat

目的

LPの状態を獲得する.

概要

```
int XPRBgetlpstat(XPRBprob prob);
```

引数

prob 現在の問題.

例

この例は現在のLPの状態を得る.

```
XPRBprob expl2;
int status;
...
expl2 = XPRBnewprob("example2");
XPRBsolve(expl2, "l");
status = XPRBgetlpstat(expl2);
```

追加情報

Xpress-OptimizerからのLPの状態についての情報を返す. 返り値は以下のような意味をもつ.

0 まだ問題がロードされていない;
1 LP は最適解;
2 LP は実行不能;
3 目的関数値は足切点より悪い;
4 LP 実行中;
5 LP 有界でない;
6 LP 双対によって足切.

関連事項

XPRBgetmipstat, XPRBgetprobstat.

XPRBgetmipstat

目的

MIPの状態を獲得する.

概要

```
int XPRBgetmipstat(XPRBprob prob);
```

引数

prob 現在の問題.

例

この例は現在のMIPの状態を返す.

```
XPRBprob expl2;
int status;
...
expl2 = XPRBnewprob("example2");
XPRBsolve(expl2,"g");
status = XPRBgetmipstat(expl2);
```

追加情報

1. この関数はXpress-Optimizer全探索(MIP)の状態を返す.
2. 返り値は以下のような意味をもつ.

0 問題はまだロードされていない;
1 LP はまだ最適化されていない;
2 LP 最適化された;
3 全探索は未終了— 整数解はひとつも見つかっていない;
4 全探索は未終了，整数解が見つかっている;
5 全探索は終了, 整数解はみつからない;
6 全探索は終了，整数解がみつかった.

関連事項

XPRBgetlpstat, XPRBgetprobstat.

XPRBgetmodcut

目的

制約式の型を獲得する.

概要

```
int XPRBgetmodcut(XPRBprob prob, XPRBctr ctr);
```

引数

`prob` **現在の問題**.

`ctr` **制約式への参照**

例

```
XPRBctr ctr1;
int mcstat;
...
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
mcstat = XPRBgetmodcut(prob, ctr1);
```

この例では，制約式`ctr1`**が通常の制約であるか**model cut**であるかを決定している.**

追加情報

1. **この関数は与えられた制約式が通常の制約であるか**model cut**であるかを決定する.**
2. **返り値は以下のような意味をもつ.** The return value is the constraint type, which will be one of:

`0` **通常の制約**;
`1` model cut.

エラーが起こったときは`-1`**を返す.**

関連事項

`XPRBsetmodcut`.

XPRBgetobjval

目的

目的関数値を獲得する.

概要

```
double XPRBgetobjval(XPRBprob prob);
```

引数

`prob` **現在の問題.**

例

以下の例は目的関数値を獲得する.

```
XPRBprob expl2;
double objval;
...
expl2 = XPRBnewprob("example2");
XPRBsolve(expl2,"l");
objval = XPRBgetobjval(expl2);
```

追加情報

この関数はXpress-Optimizerから現在の目的関数の値を返す．もし全探索が終了し整数解が見つかった場合（関数 `XPRBgetmipstat` **が4または6を返すとき）は，返り値は最適整数解である．その他の場合は，最後に解いたLPの最適値が返される.**

関連事項

`XPRBgetdual`, `XPRBgetrcost`, `XPRBgetsol`, `XPRBgetslack`.

XPRBgetprobname

目的

問題の名前を獲得する.

概要

```
char *XPRBgetprobname(XPRBprob prob);
```

引数

`prob` **現在の問題.**

例

```
XPRBprob expl2;
char *pbname;
expl2 = XPRBnewprob("example2");
pbname = XPRBgetprobname(expl2);
printf("%s",pbname);
```

この例は現在の問題の名前（`example2`**）を返す.**

関連事項

`XPRBdelprob`, `XPRBnewname`, `XPRBnewprob`.

XPRBgetprobstat

目的

問題の状態を獲得する.

概要

```
int XPRBgetprobstat(XPRBprob prob);
```

引数

prob **現在の問題**.

例

この例は，現在の問題の状態を獲得し，問題に変更が加えられていれば再度解く.

```
XPRBprob expl2;
int status;
...
expl2 = XPRBnewprob("example2");
status = XPRBgetprobstat(expl2);
if((status&XPRB_MOD)==XPRB_MOD)
XPRBsolve(expl2,"");
```

追加情報

返り値は以下のいずれかである.

XPRBgetlpstat, XPRBgetmipstat.

XPRB_GEN **行列が生成された**;

XPRB_DIR **分枝優先順位が付けれた**;

XPRB_MOD **問題に変更がなされた**;

XPRB_SOL **問題は解かれた**;

関連事項

Xpress-BCL Reference Manual BCL Library Functions 66

XPRBgetrange

目的

範囲制約の範囲値を獲得する.

概要

```
int XPRBgetrange(XPRBprob prob, XPRBctr ctr, double *bdl,
double *bdu);
```

引数

prob **現在の問題.**

ctr **範囲制約への参照**

bdl **範囲制約の下限値. 必要なければ**NULL**を指定する.**

Bdu **範囲制約の上限値. 必要なければ**NULL**を指定する.**

例

```
XPRBctr ctr2;
XPRBarrvar ty1;
double bdl, bdu;
...
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
ctr2 = XPRBnewsum(prob, "r2", ty1, XPRB_E, 9);
XPRBgetrange(prob, ctr2, &bdl, &bdu);
```

この例では制約式ctr2**の上下限値を獲得する.**

追加情報

この関数は制約式の上下限値を返す. もしbdl**または** bdu**に**NULL**が設定されると, その引数のほうには返り値はない.**

関連事項

XPRBsetrange.

XPRBgetrcost 4

XPRBgetrcost

目的

変数のリデューストコストを獲得する.

概要

```
double XPRBgetrcost(XPRBprob prob, XPRBvar var);
```

引数

prob **現在の問題.**

var **変数への参照**

例

```
XPRBprob expl2;
XPRBvar x1;
double rcval;
...
expl2 = XPRBnewprob("example2");
x1 = XPRBnewvar(expl2, XPRB_UI, "abc3", 1, 100);
XPRBsolve(expl2, "l");
rcval = XPRBgetrcost(expl2, x1);
```

この例は変数x1**のリデューストコストを獲得する.**

追加情報

この関数は指定した変数のリデューストコストを返す．利用者は，ある変数が問題の中で有効となっているかどうかを調べるとき，まずその列番号が非負であることによって調べることができる．全探索が終了し整数解が見つかった場合は，この関数は最適整数解に対する値を返す．もしそうでないきは，最後に解かれたLP**のリデューストコストを返す．注意することは，この関数は最適化の実行途中では使えないことである.**

関連事項

XPRBgetdual, XPRBgetobjval, XPRBgetslack, XPRBgetsol.

XPRBgetrhs 4

XPRBgetrhs

目的

制約式の右辺の値を与える.

概要

```
double XPRBgetrhs(XPRBprob prob, XPRBctr ctr);
```

引数

prob 現在の問題.

ctr 制約式への参照

例

この例は，制約式ctr1の右辺の値を与える.

```
XPRBctr ctr1;
double rhs;
...
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
rhs = XPRBgetrhs(prob, ctr1);
```

追加情報

この関数は，制約式の右辺の値を与える．右辺のデフォルト値は0である．指定した制約が範囲制約の場合は，その上限値を返す.

関連事項

XPRBaddterm, XPRBgetctrtype, XPRBsetctrtype, XPRBsetterm.

XPRBgetrownum

目的

制約式の行番号を与える.

概要

```
int XPRBgetrownum(XPRBprob prob, XPRBctr ctr);
```

引数

prob 現在の問題.

ctr 制約式への参照

例

この例は制約式ctr1の行番号を与える.

```
XPRBctr ctr1;
...
int rindex;
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
rindex = XPRBgetrownum(prob, ctr1);
```

追加情報

この関数は，行列中で，制約式の対応する行番号を返す．行列がまだ生成されていない場合や，制約式が行列に含まれていない場合（これは制約式の型がXPRB_Nであるか，あるいは制約式に非零項がない場合）は，負の値を返す．行番号は，0から始まる.

関連事項

XPRBdelctr, XPRBnewctr.

XPRBgetsense

目的

目的関数の最適化の方向を与える．

概要

```
int XPRBgetsense(XPRBprob prob);
```

引数

prob **現在の問題**.

XPRB_MAXIM **目的関数は最大化される**;

XPRB_MININ **目的関数は最小化される**.

例

この例は，問題expl2**の最適化の方向を与える．**

```
XPRBprob prob;
int dir;
...
prob = XPRBnewprob("example2");
dir = XPRBgetsense(prob);
```

追加情報

この関数は目的関数の最適化の方向（最大化，最小化）を返す．この関数が正常に終了したときの返り値は上記の通り．返り値が－1**のときは，エラーである．最適化の方向はデフォルトでは最小化に設定されるが，関数**XPRBsetsense, XPRBminim, XPRBmaxim.**によって変更できる．**

関連事項

XPRBmaxim, XPRBminim, XPRBsetsense, XPRBsolve.

XPRBgetslack 4

XPRBgetslack

目的

制約式のスラック値を与える.

概要

```
double XPRBgetslack(XPRBprob prob, XPRBctr ctr);
```

引数

prob **現在の問題.**

ctr **制約式への参照**

例

```
XPRBprob expl2;
XPRBctr ctr2;
XPRBarrvar ty1;
double slack;
...
expl2 = XPRBnewprob("example2");
ty1 = XPRBnewarrvar(expl2, 5, XPRB_PL, "arry1", 0, 500);
ctr2 = XPRBnewsum(expl2, "r2", ty1, XPRB_E, 9);
XPRBsolve(expl2, "l");
slack = XPRBgetslack(expl2, ctr2);
```

この例は制約式ctr2**のスラック値を与える.**

追加情報

この関数は制約式のスラック値を与える．利用者は，ある制約式が問題の中で有効となっているかどうかを調べるとき，まずその行番号が非負であることによって調べることができる．全探索が終了し整数解が見つかった場合は，この関数は最適整数解に対する値を返す．もしそうでないときは，最後に解かれたLP**でのスラック値を返す．注意することは，この関数は最適化の実行途中では使えないことである.**

関連事項

XPRBgetdual, XPRBgetobjval, XPRBgetrcost, XPRBgetsol.

XPRBgetsol 4

XPRBgetsol

目的

変数の値を与える．

概要

```
double XPRBgetsol(XPRBprob prob, XPRBvar var);
```

引数

prob **現在の問題.**

var **変数への参照**

例

```
XPRBprob expl2;
XPRBvar x1;
double solval;
...
expl2 = XPRBnewprob("example2");
x1 = XPRBnewvar(expl2, XPRB_UI, "abc3", 1, 100);
XPRBsolve(expl2, "l");
solval = XPRBgetsol(expl2, x1);
```

The retrieves the solution value for the variable x1.

追加情報

この関数は，現時点の解での変数の値を返す．利用者は，ある変数が問題の中で有効となっているかどうかを調べるとき，まずその列番号が非負であることによって調べることができる．全探索が終了し整数解が見つかった場合は，この関数は最適整数解に対する値を返す．もしそうでないきは，最後に解かれたLPでの値を返す．「整数解」とは「実行可能整数解」の意味であることに注意する．つまり，解の値を比較するときには，現在のOptimizerの許容値の設定を考慮しないといけない．また，整数変数の解の値を扱うときも注意が必要である．例えば，数値を丸めるとき，0.999998が0に丸められてしまうこともあるので，注意が必要である．もっとも近い整数への丸めをするよう注意が必要である．なお，この関数は最適化の実行途中では使えないこと．

XPRBgetsol 4

関連事項

XPRBgetdual, XPRBgetobjval, XPRBgetrcost, XPRBgetslack.

Xpress-BCL Reference Manual BCL Library Functions 74

XPRBgetsosname 4

XPRBgetsosname

目的

SOSの名前を与える.

概要

```
char *XPRBgetsosname(XPRBprob prob, XPRBsos sos);
```

引数

prob 現在の問題.

sos SOSへの参照

例

```
XPRBsos set1;
...
set1 = XPRBnewsos(prob, "sos1", XPRB_S1);
printf("%s\n",XPRBgetsosname(set1));
```

この例は"sos1"と出力する.

追加情報

この関数は，SOSの名前を与える．利用者が名前を定義していないときは，BCLが自動的に生成した名前を返す.

関連事項

XPRBdelsos, XPRBgetsostype, XPRBnewsos.

Xpress-BCL Reference Manual BCL Library Functions 75

XPRBgetsostype 4

XPRBgetsostype

目的

SOSの型を与える.

概要

```
int XPRBgetsostype(XPRBprob prob, XPRBsos sos);
```

引数

prob 現在の問題.

sos SOSへの参照

例

```
XPRBsos set1;
char stype;
...
set1 = XPRBnewsos(prob, "sos1", XPRB_S1);
stype = XPRBgetsostype(prob, set1);
```

この例はSOS set1の型を返す.

追加情報

この関数は，SOSの型を返す．型は以下のどれかである.

XPRB_S1 型 1;

XPRB_S2 型 2.

返り値が1のときはエラーである.

関連事項

XPRBdelsos, XPRBgetsosname, XPRBnewsos.

XPRBgettime

目的

時刻を与える.

概要

```
int XPRBgettime(void);
```

引数

None.

例

この例はプログラムの経過時間を出力する.

```
int starttime;
starttime = XPRBgettime();
(...)
printf("Time: \%g sec",(XPRBgettime()-starttime)/1000);
```

追加情報

この関数は，ミリ秒単位で計ったシステム時刻を返す．時刻は計算機システムに依存する．プログラムの実行時間を計測するためには，この関数によって，開始時刻と計測ポイントでの時刻の差をとる.

関連事項

`XPRBgetversion`.

XPRBgetvarname 4

XPRBgetvarname

目的

変数の名前を与える.

概要

```
char *XPRBgetvarname(XPRBprob prob, XPRBvar var);
```

引数

prob **現在の問題.**

var **変数への参照**

例

この例は，変数の名前をプリントする.

```
XPRBvar x1;
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
printf("%s\n",XPRBgetvarname(prob, x1));
```

追加情報

この関数は変数の名前を返す．利用者が変数名を定義していないときは，BCL**が自動的につけた名前を返す.**

関連事項

XPRBgetarrvarname, XPRBgetvartype, XPRBnewvar, XPRBsetvartype.

XPRBgetvartype

目的

変数の型を返す.

概要

```
int XPRBgetvartype(XPRBprob prob, XPRBvar var);
```

引数

prob **現在の問題**.

var **変数への参照**

例

```
XPRBvar x1;
char vtype;
...
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
vtype = XPRBgetvartype(prob, x1);
```

この例は変数x1**の型を返す.**

追加情報

この関数が正常に終了したとき，以下のいずれかを返す.:

関連事項

XPRBnewvar, XPRBsetvartype.

XPRB_PL **連続**;

XPRB_BV 2**値**;

XPRB_UI **整数**;

XPRB_PI **部分整数**;

XPRB_SC **準連続**;

XPRB_SI **準連続整数**.

XPRBgetversion

目的

BCLのバージョンを与える.

概要

```
const char *XPRBgetversion(void);
```

引数

None.

例

この例はBCLのバージョンを獲得し，1.1.0. という風に表示する.

```
char *version;
version = XPRBgetversion();
printf("%s",version);
```

追加情報

この関数はBCLのバージョンを与える．この番号は利用者が問題の問合わせなどするときに必要である.

関連事項

XPRBgettime.

XPRBgetXPRSprob

目的

BCL で作成され，Xpress-Optimizer へとロードされた問題の，XPRSprob 型のポインタを返す.

概要

```
XPRSprob XPRBgetXPRSprob(XPRBprob prob);
```

引数

prob 現在のBCL問題

例

以下はこの関数の標準的な使い方である:

```
XPRBprob bcl_prob;
XPRSprob opt_prob;
XPRBinit("");
bcl_prob = XPRBnewprob("MyProb");
...
XPRBloadmat(bcl_prob);
opt_prob = (XPRSprob)XPRBgetXPRSprob(bcl_prob);
XPRSmaxim(opt_prob,"");
```

関連事項

XPRBloadmat, XPRBnewprob, XPRScreateprob (参照先 Optimizer Reference Manual), Appendix A, “Using BCL with the Optimizer Library”.

XPRBinit 4

XPRBinit

目的

BCLを初期化する．

概要

```
int XPRBinit(char *xpress);
```

引数

xpress これはXpress-MPのセキュリティファイルを探索パスを指定するために使われる．指定しなければデフォルトの探索パスが使われる．

例

```
XPRBseterrctrl(0);
if(XPRBinit(""))
printf("BCL has not been initialized correctly.
Please check your Xpress-MP licenses.");
```

この例はエラー処理を利用者に移し，BCLを初期化する．（ライセンスのテストでもある）

追加情報

1. BCLが学生版で稼動しているときは，32を返す．
2. この関数は，BCLの初期化を明示的に行う．すなわち，ソフトウェアのライセンスの有無を確認する．この初期化は関数XPRBnewprobによっても行われるので，通常はこの関数を呼ぶ必要はない． BCL を他のより大きなプログラムに組み込んで使うときなど，実際にモデルを作成する前に，ライセンスの確認をするのに利用できる．この関数で，有効なライセンスが見つからず，初期化ができないときは，学生版としてプログラムは稼動する．このときは，利用可能な機能の制限や，問題サイズの制限が加えられる．

関連事項

XPRBfree, XPRBnewprob, XPRSinit (Optimizer Reference Manualを参照されたい).

XPRBloadbasis 4

XPRBloadbasis

目的

以前に保存した基底解をロードする．

概要

```
int XPRBloadbasis(XPRBprob prob, XPRBbasis basis);
```

引数

prob **現在の問題.**

basis **以前に保存した基底解への参照**

例

この例は，行列の変更の前に現在の基底解を保存しておき，その後，線形緩和を解くために，その保存しておいた基底解を再びロードしている．

```
XPRBprob expl2;
XPRBbasis basis;
...
expl2 = XPRBnewprob("example2");
XPRBsolve(expl2,"l");
basis = XPRBsavebasis(expl2);
...
XPRBloadmat(expl2);
XPRBloadbasis(expl2,basis);
XPRBdelbasis(expl2,basis);
XPRBsolve(expl2,"l");
```

追加情報

この関数は，現在の問題の基底解をロードする．基底解XPRBsavebasis**で保存されていないといけない．似たような問題だからといって，現在の問題以外の基底解をロードすることはできない．この関数は，問題が変更されている（変数や制約式が追加／削除されている）ことを考慮にいれている．ファイルから基底解を読み込むためには，** Optimizer library **の**XPRSreadbas**が利用できる．基底解が**XPRBloadbasis**で再ロードされる前には，前もって(**XPRBloadmat**で) 問題がロードされていなければならない．さらに，基底解への参照は，以後利用しないのであれば,** XPRBdelbasis **で削除しておくべきである．**

関連事項

XPRBdelbasis, XPRBsavebasis, XPRSreadbasis (Optimizer Reference Manualを参照されたい), XPRSwritebasis (Optimizer Reference Manualを参照されたい).

XPRBloadmat

目的

問題をXpress-Optimizerへロードする.

概要

```
int XPRBloadmat(XPRBprob prob);
```

引数

prob 現在の問題.

例

問題expl2の行列が生成される.

```
XPRBprob expl2;
expl2 = XPRBnewprob("example2");
...
XPRBloadmat(expl2);
```

追加情報

この関数は, Optimizer libraryの関数 XPRSloadlp, XPRSloadqp, XPRSloadglobal, またはXPRSloadqglobalを呼んで，現在のBCL の問題をXpress-Optimizerの行列へと変換する．空の行や列は，生成のまえに削除される．問題で定義されているが，行列中には現れない変数には，負の番号が付けられる．通常はこの関数を明示的に呼ぶ必要はない．BCLが必要に応じて自動的にこの変換を行う.

関連事項

XPRBgetXPRSprob, XPRSloadglobal (Optimizer Reference Manualを参照されたい), XPRSloadlp (Optimizer Reference Manualを参照されたい), XPRSloadqglobal (Optimizer Reference Manualを参照されたい), XPRSloadqp (Optimizer Reference Manualを参照されたい), Appendix A, “Using BCL with the Optimizer Library”.

XPRBmaxim

目的

目的関数の最大化を行う．

概要

```
int XPRBmaxim(XPRBprob prob, char *flags);
```

引数

`prob` **現在の問題**.

`flags` **解法のアルゴリズムの選択を行う．以下のいずれかを選択する**:

`p` **主単体法**;

`d` **双対単体法**;

`b` **ニュートンバリア法**;

`l` **（緩和）線形問題のみを解く**;

`g` **大域解を求める**;

no parameter — **デフォルト設定を使う**

例

ニュートンバリア法を使いLPの最大化をする．

```
XPRBprob expl2;
...
expl2 = XPRBnewprob("example2");
XPRBmaxim(expl2,"b");
```

追加情報

この関数はXpress-Optimizerのアルゴリズムを選択し実行する．アルゴリズムを選択するときの文字は，その組み合わせが意味をもつときは，"dg"**のように組み合わせて使うことができる．もし現在Optimizerにロードされている行列が，現在の指定された問題の状態と対応しない場合は，アルゴリズムを開始する前に生成し直す．問題を解く前には**`XPRBsetobj`**で目的関数を指定しておかなければならない．もし，全探索を途中でやめるときは，**Optimizer library **の関数**`XPRSinitglobal`**を呼んで，保存されている探索木の情報を削除しておかなければならない．さもないと，あとでそのプログラムを動かすことができなくなるかもしれない．**

関連事項

`XPRBgetobjval`, `XPRBgetsol`, `XPRBminim`, `XPRBsetsense`, `XPRBsolve`, `XPRSmaxim` (Optimizer Reference Manual**を参照されたい**).

XPRBminim 4

XPRBminim

目的

目的関数の最小化を行う．

概要

```
int XPRBminim(XPRBprob prob, char *flags);
```

引数

prob **現在の問題**.

flags **解法のアルゴリズムの選択を行う．以下のいずれかを選択する**:

p **主単体法**;

d **双対単体法**;

b **ニュートンバリア法**;

l **（緩和）線形問題のみを解く**;

g **大域解を求める**;

no parameter — **デフォルト設定を使う**

例

ニュートンバリア法を使いLP**の最小化をする．**

```
expl2 = XPRBnewprob("example2");
...
XPRBminim(prob,"b");
```

追加情報

この関数はXpress-Optimizer**のアルゴリズムを選択し実行する．アルゴリズムを選択するときの文字は，その組み合わせが意味をもつときは，**"dg"**のように組み合わせて使うことができる．もし現在**Optimizer**にロードされている行列が，現在の指定された問題の状態と対応しない場合は，アルゴリズムを開始する前に生成し直す．問題を解く前には**XPRBsetobj**で目的関数を指定しておかなければならない．もし，全探索を途中でやめるときは，**Optimizer library** の関数**XPRSinitglobal**を呼んで，保存されている探索木の情報を削除しておかなければならない．さもないと，あとでそのプログラムを動かすことができなくなるかもしれない．**

関連事項

XPRBgetobjval, XPRBgetsol, XPRBmaxim, XPRBsetsense, XPRBsolve, XPRSminim (Optimizer Reference Manual**を参照されたい**).

XPRBnewarrsum

目的

個々の係数を与えて，新たに和の形の制約式を生成する．

概要

```
XPRBctr XPRBnewarrsum(XPRBprob prob, char *name,
XPRBarrvar av, double *cof, char qrtype, double rhs);
```

引数

prob **現在の問題**.

name **制約式名．**64**文字まで，**NULL**でもよい．**

av **変数の配列への参照**

cof av**の各要素への係数を入れる配列．この配列すくなくともav**と**同じサイズであること．**

qrtype **制約式の型．以下のいずれか**:

XPRB_L **以下**;

XPRB_G **以上**;

XPRB_E **等号**;

XPRB_N **制約なし (目的関数)**.

rhs **制約式の右辺の値**

例

この例は sum(i=0:4) c[i]*ty1[i] >= 7.0 **という制約式を生成する．**

```
XPRBctr ctr4;
XPRBarrvar ty1;
double c[] = {2.5, 4.0, 7.2, 3.0, 1.8};
...
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
ctr4 = XPRBnewarrsum(prob, "r4", ty1, c, XPRB_G, 7.0);
```

追加情報

この関数は，変数の配列の各要素にcof**で指定された係数を乗じて和を作成し、制約式を生成する．この関数は**XPRBnewctr**と**XPRBaddterm**で代替することもできる．もし，指定した名前が既に使われていれば，その名前の後に番号をつけて使われる．もし名前が与えられなければデフォルト名**CTR**が付けられる。**

XPRBnewarrsum 4

関連事項

XPRBdelctr, XPRBnewctr, XPRBnewprec, XPRBnewsum.

XPRBnewarrvar

目的

１次元の変数配列を作る．

概要

```
XPRBarrvar XPRBnewarrvar(XPRBprob prob, int nbvar,
int type, char *name, double bdl, double bdu);
```

引数

prob **現在の問題**.

nbvar **変数の配列のサイズ**

type **変数の型．以下のいずれか**:

XPRB_PL **連続**;

XPRB_BV **２値**;

XPRB_UI **整数**;

XPRB_PI **部分整数**;

XPRB_SC **準連続**;

XPRB_SI **準連続整数**.

name **配列の名前．**NULL**でもよい．**

bdl **変数の下限値**

bdu **変数の上限値**

例

この例は，0**から**500**の間の値をとる連続変数１０個からなる配列を生成する．配列の名前は**arry1**.**

```
XPRBarrvar ty1;
...
ty1 = XPRBnewarrvar(prob, 10, XPRB_PL, "arry1", 0, 500);
```

追加情報

1. **この関数は，１次元の変数配列を生成する．各変数の上下限は**XPRBsetlb**や**XPRBsetub**によって変更できる．また，変数の型は**XPRBsetvartype**によって変更できる．関数は変数配列への参照を返す．もし指定した名前が既に使われているときは，その名前の後に番号をつけて使われる．もし名前が与えられなければデフォルト名**AV**が付けられる。**

2. **正または負の無限大を指定するには，**XPRB_INFINITY **や** -XPRB_INFINITY**を使う．**

関連事項

XPRBdelarrvar, XPRBendarrvar, XPRBstartarrvar.

XPRBnewctr 4

XPRBnewctr

目的

新しい制約式を生成する．

概要

```
XPRBctr XPRBnewctr(XPRBprob prob, char *name, char qrtype);
```

引数

prob 現在の問題.

name 制約式名．64文字まで，NULLでもよい．

qrtype 制約式の型．以下のいずれか:

XPRB_L 以下;

XPRB_G 以上;

XPRB_E 等号;

XPRB_N 制約なし (目的関数).

例

この例は，新しい等式制約式を生成する．

```
XPRBctr ctr1;
...
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
```

追加情報

この関数は新しい制約式を生成し，その参照を返す．関数XPRBaddtermを使い項を追加する前には，この関数を呼んでおかなければならない．範囲制約は，最初任意の型の制約として生成しておき，関数XPRBsetrangeで型を変換してつくる．もし指定した名前が既に使われているときは，その名前の後に番号をつけて使われる．もし名前が与えられなければデフォルト名CTRが付けられる。

関連事項

XPRBaddterm, XPRBdelctr, XPRBdelterm.

XPRBnewidxset

目的

新しいインデックス集合を生成する．

概要

```
XPRBidxset XPRBnewidxset(XPRBprob prob, char *name,
int maxsize);
```

引数

prob **現在の問題**.

name **インデックス集合の名前．**NULL**でもよい**

maxsize **インデックス集合のサイズ**

例

この例は，100**のスペースを持ったインデックス集合を生成する．**

```
XPRBidxset iset;
...
iset = XPRBnewidxset(prob,"Set",100);
```

追加情報

この関数はインデックス集合を生成する．引数maxsize**で指定されるサイズは，最初に割り当てられるスペースである．これは，あとで増やすこともできる．もし指定した名前が既に使われているときは，その名前の後に番号をつけて使われる．もし名前が与えられなければデフォルト名**IDX**が付けられる。**

関連事項

XPRBaddidxel, XPRBgetidxel, XPRBgetidxsetname, XPRBgetidxsetsize.

XPRBnewname 4

XPRBnewname

目的

名前の文字列を作成する.

概要

```
char *XPRBnewname(XPRBprob prob, char *format, ...);
```

引数

prob **現在の問題.**

format **C言語で利用するフォーマット文(詳しくは，C言語のマニュアルの項のフォーマットオプションを参照せよ). よく使われるのは：**

%c **１文字**;

%d **整数**;

%g **倍精度**;

%s **文字列**

... **フォーマット文の中で使われる文字数字など．カンマで区切る.**

例

この例は，xab15**という名前の変数を獲得する.**

```
char a[] = "ab";
int i = 15;
XPRBvar x1;
x1 = XPRBgetbyname(prob,
XPRBnewname(prob,"x%s%d",a,i),XPRB_VAR);
```

追加情報

この関数は，BCL**のオブジェクトの名前を簡易に作成するのに用いる．これは，他の関数などで，名前をパラメータとして使う場合を想定している．標準のC**の文字列関数と違い，この関数は利用者にメモリーの割り当てを要求しない.**

関連事項

XPRBdelprob, XPRBgetprobname, XPRBnewprob.

XPRBnewprec 4

XPRBnewprec

目的

新たに優先度制約v1 + dur <= v2を生成する．

概要

```
XPRBctr XPRBnewprec(XPRBprob prob, char *name, XPRBvar v1,
double dur, XPRBvar v2);
```

引数

prob 現在の問題.

name 制約式の名前．64文字まで．NULLでもよい

v1 変数への参照

dur 倍制度または整数の定数

v2 変数への参照

例

この例は不等式制約ty1[2] + 5.4 <= ty1[4]を生成する．

```
XPRBctr ctr5;
XPRBarrvar ty1;
...
ty1 = XPRBnewarrvar(5, XPRB_PL, "arry1", 0, 500);
ctr5 = XPRBnewprec("r5", ty1[2], 5.4, ty1[4]);
```

追加情報

この関数は，いわゆる優先度制約（ある変数に定数を加えたものが２番目の変数以下であるというような）を生成する．この関数はXPRBnewctr と XPRBaddtermで代替可能である．もし指定した名前が既に使われているときは，その名前の後に番号をつけて使われる．もし名前が与えられなければデフォルト名CTRが付けられる。

関連事項

XPRBnewarrsum, XPRBnewsum.

XPRBnewprob

目的

新しい問題を初期化する．

概要

```
XPRBprob XPRBnewprob(char *probname);
```

引数

probname 問題名．NULLでもよい

例

この例は，example2という名前の新しい問題を定義している．

```
XPRBprob expl2;
expl2 = XPRBnewprob("example2");
```

追加情報

この関数は，新しい問題を入力したときは，初期化と以前の問題の中間結果の消去のために呼び出す必要がある．もっとも重要なことは，いくつかの問題を次々に入力したときには，この関数を呼び出さなければならないことである．この関数はXpress-MPを初期化するので，XPRSinitを呼ぶ必要はない．初期化の過程で，有効なライセンスファイルが見つからなければ，BCLは学生版として稼動する．このモードでは利用可能な機能と問題サイズに制限がある． 関数XPRBnewprobを呼ぶと，BCL内での現在の問題は変更されるが，Optimizerのほうにロードされている行列や解の情報は更新されない．もし問題の名前が指定されないと，デフォルト名PROBが付けられる．

関連事項

XPRBdelprob, XPRBgetprobname, XPRBnewname XPRScreateprob (see Optimizer Reference Manual).

XPRBnewsos 4

XPRBnewsos

目的

新たにSOSを作成する．

概要

```
XPRBsos XPRBnewsos(XPRBprob prob, char *name, char type);
```

引数

name 集合の名前

type 集合の型．以下のいずれか：

XPRB_S1 Special Ordered Set 型 1;

XPRB_S2 Special Ordered Set 型 2.

例

この例はsos1という名前のSOS 型 1を生成する．

```
XPRBsos set1;
...
set1 = XPRBnewsos(prob, "sos1", XPRB_S1);
```

追加情報

この関数はSpecial Ordered Set (SOS) 型1 または型2 (省略名SOS1 と SOS2)を生成する．この関数は，生成したSOSの参照を返す．これは，以下のような関数に引数に使われる．XPRBaddsoselは要素の追加．XPRBdelsoselは（１個の）要素の削除，XPRBdelsosは全要素の削除．もし指定した名前が既に使われているときは，その名前の後に番号をつけて使われる．もし名前が与えられなければデフォルト名SOSられる。

関連事項

XPRBdelsos, XPRBgetsosname, XPRBgetsostype, XPRBnewsosrc, XPRBnewsosw.

prob 現在の問題.

XPRBnewsosrc

目的

指定した制約式を使ってSOSを生成する．

概要

```
XPRBsos XPRBnewsosrc(XPRBprob prob, char *name, char type,
XPRBarrvar av, XPRBctr ctr);
```

引数

例

この例は型２のSOSを，変数配列ty1と係数の配列からなる制約式ctr4を使って生成する．

```
XPRBsos set2;
XPRBctr ctr4;
XPRBarrvar ty1;
double c[] = {2.5, 4.0, 7.2, 3.0, 1.8};
...
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
ctr4 = XPRBnewarrsum(prob, "r4", ty1, c, XPRB_G, 7.0);
set2 = XPRBnewsosrc(prob, "sos2", XPRB_S2, ty1, ctr4);
```

追加情報

この関数は，SOS を段階的に定義する代わりに変数配列が利用できて，その係数がいずれも非零である制約式が定義されているときに，用いられる．制約式を指定しないと，全ての係数が1であるように設定される．もし指定した名前が既に使われているときは，その名前の後に番号をつけて使われる．もし名前が与えられなければデフォルト名SOSが付けられる。

prob **現在の問題.**

name **SOSの名前**

type **SOSの型．以下のいずれかである**:

XPRB_S1 Special Ordered Set **型** 1;

XPRB_S2 Special Ordered Set **型** 2.

av **変数の配列．**NULL **でも可**

ctr **制約式への参照．**NULL **でも可**

XPRBnewsosrc 4

関連事項

XPRBdelsos, XPRBgetsosname, XPRBgetsostype, XPRBnewsos, XPRBnewsosw.

XPRBnewsosw 4

XPRBnewsosw

目的

重み係数を使ってSOSを生成する．

概要

```
XPRBsos XPRBnewsosw(XPRBprob prob, char *name, char type,
XPRBarrvar av, double *weight);
```

引数

prob **現在の問題**.

name **SOSの名前**

type **SOSの型．以下のいずれかである**:

XPRB_S1 Special Ordered Set **型** 1;

XPRB_S2 Special Ordered Set **型** 2.

av **変数の配列．**

weight **係数の配列．**NULL**でも可**

例

この例は変数配列ty1 **と係数配列**cr**を使って，**SOS**型**1**を生成する．**

```
XPRBsos set1;
XPRBarrvar ty1;
double cr[] = {2.0, 13.0, 15.0, 6.0, 8.5};
...
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
set1 = XPRBnewsosw(prob, "sos1", XPRB_S1, ty1, cr);
```

追加情報

この関数は，変数とその係数が配列で与えれられているとき，XPRBnewsos**や**XPRBaddsosarrel **によって** SOS **を段階的に定義する代わりに利用される．係数を指定しないと，全ての係数が**1**であるように設定される．もし指定した名前が既に使われているときは，その名前の後に番号をつけて使われる．もし名前が与えられなければ、デフォルト名**SOS**が付けられる。**

関連事項

XPRBdelsos, XPRBgetsosname, XPRBnewsos, XPRBnewsosrc.

XPRBnewsum

目的

和の形の制約式を生成する．

概要

```
XPRBctr XPRBnewsum(XPRBprob prob, char *name,
XPRBarrvar av, char qrtype, double rhs);
```

引数

prob 現在の問題.

name 制約式の名前．64文字まで．NULLでもよい

av 変数の配列への参照

qrtype 制約式の型．以下のいずれかである:

XPRB_L 以下;

XPRB_G 以上;

XPRB_E 等式;

XPRB_N 制約なし(目的関数).

rhs 制約式の右辺の値

例

この例は制約式ctr2をsum(i=0:4) ty1[i]= 9によって作成する．

```
XPRBctr ctr2;
XPRBarrvar ty1;
...
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
ctr2 = XPRBnewsum(prob, "r2", ty1, XPRB_E, 9);
```

追加情報

この関数は，与えられた変数配列の要素の単純和による制約式を生成する．この関数は，XPRBnewctr と XPRBaddtermで置き換えることもできる．もし指定した名前が既に使われているときは，その名前の後に番号をつけて使われる．もし名前が与えられなければ、デフォルト名CTRが付けられる。

関連事項

XPRBnewarrsum, XPRBnewctr, XPRBnewprec.

XPRBnewvar

目的

変数を宣言する.

概要

```
XPRBvar XPRBnewvar(XPRBprob prob, int type, char *name,
double bdl, double bdu);
```

引数

prob **現在の問題**.

type The variable type, which may be one of:

XPRB_PL continuous;

XPRB_BV binary;

XPRB_UI general integer;

XPRB_PI partial integer;

XPRB_SC semi-continuous;

XPRB_SI semi-continuous integer.

name The variable name, of up to 64 characters. May be NULL if not required.

bdl The variable's lower bound

bdu The variable's upper bound

例

```
XPRBvar x1;
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 1, 100);
```

この例は，abc3**という名前の**1**から**100**の間の値をとる整数変数を定義している.**

追加情報

1. BCL**で変数を生成するときは，その名前だけでなく，その型ととる範囲（無限大でもよい）を指定する．この関数は変数への参照** (i.e. **モデル変数)を返す．．もし指定した名前が既に使われているときは,その名前の後に番号をつけて使われる.もし名前が与えられなければデフォルト名**VAR **が付けられる。部分整数変数，半連続変数，半連続整数変数が定義されるときは，整数または半連続の限界値（整数変数や半連続変数の連続部分の下限値，半連続整数変数の半連続整数部分の下限値）は**1**と**bdl**の大きい方に設定される．この限界値は後で関数** XPRBsetlim**によって変更できる.**

2. **上下限値は，**XPRB_INFINITY**または**XPRB_INFINITY**を設定することで，無限大にすることもできる.**

関連事項

XPRBnewarrvar, XPRBsetvartype, XPRBstartarrvar.

XPRBprintarrvar

目的

変数の配列をプリントする.

概要

```
int XPRBprintarrvar(XPRBprob prob, XPRBarrvar av);
```

引数

prob **現在の問題.**

av **変数の配列への参照**

例

```
XPRBarrvar ty1;
...
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
XPRBprintarrvar(prob, ty1);
```

この例は変数配列ty1**に含まれる全変数の名前と上下限値をプリントする.**

追加情報

この関数は配列中の全変数をプリントする(名前，上下限値，解).

関連事項

XPRBexportprob, XPRBprintctr, XPRBprintprob, XPRBprintvar.

XPRBprintctr 4

XPRBprintctr

目的

制約式をプリントする

概要

```
int XPRBprintctr(XPRBprob prob, XPRBctr ctr);
```

引数

prob 現在の問題.

ctr 制約式への参照

例

この例は制約式ctr2**をプリントする**

```
XPRBctr ctr2;
XPRBarrvar ty1;
...
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
ctr2 = XPRBnewsum(prob, "r2", ty1, XPRB_E, 9);
XPRBprintctr(prob, ctr2);
```

追加情報

この関数は制約式をLP**形式でプリントする．学生版では利用できない.**

関連事項

XPRBexportprob, XPRBprintprob, XPRBprintarrvar, XPRBprintvar.

XPRBprintf 4

XPRBprintf

目的

テキストをプリントする．

概要

```
int XPRBprintf(const *format, ...);
```

引数

format テキストをプリントするときのフォーマットを示す文字列．C言語のprintf関数と共通

... 上のフォーマット形式でプリントされる項目．カンマで区切る．

例

この例は，最初に"New variable: abc3"とプリントし，次行に"A real number: 1.3, an integer: 5"とプリントする．

```
XPRBvar x1;
double a=1.3;
int i=5;
...
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 1, 100);
XPRBprintf("New variable: %s\n",
XPRBgetvarname(prob,x1));
XPRBprintf("A real number: %g, an integer: %d", a, i);
```

追加情報

この関数は，テキストやデータをプリントする．機能はC言語のprintf関数と同様であるが，追加機能として，XPRBdefcbmsgでコールバックが定義されているときは，標準出力にプリントする代わりにコールバックが実行される．

関連事項

XPRBprintprob, XPRBreadline.

XPRBprintidxset

目的

インデックス集合をプリントする.

概要

```
int XPRBprintidxset(XPRBprob prob, XPRBidxset idx);
```

引数

例

```
XPRBidxset iset;
...
iset = XPRBnewidxset(prob,"Set",100);
XPRBprintidxset(prob,iset);
```

この例はインデックス集合isetをプリントする.

追加情報

この関数はインデックス集合をプリントする．学生版では利用できない.

関連事項

XPRBprintctr, XPRBprintf, XPRBprintsos, XPRBprintvar.

prob **現在の問題**.

idx Reference to an index set.

XPRBprintprob

目的

指定した問題をプリントする.

概要

```
int XPRBprintprob(XPRBprob prob);
```

引数

prob **現在の問題**.

例

これ例は現在の問題の定義を印刷する.

```
XPRBprob expl2;
...
expl2 = XPRBnewprob("example2");
XPRBprintprob(expl2);
```

追加情報

この関数は，現在の問題の全定義をプリントする．すなわち，制約式のリスト，SOS，目的関数である．学生版では利用できない.

関連事項

XPRBexportprob, XPRBprintf.

XPRBprintsos

目的

SOSをプリントする.

概要

```
int XPRBprintsos(XPRBprob prob, XPRBsos sos);
```

引数

`prob` 現在の問題.

`sos` SOSへの参照

例

```
XPRBsos set1;
...
set1 = XPRBnewsos(prob, "sos1", XPRB_S1);
XPRBprintsos(prob, set1);
```

この例はSOS `set1`をプリントする

追加情報

この関数はSOSをプリントする. 学生版では利用できない.

関連事項

`XPRBprintctr`, `XPRBprintidxset`, `XPRBprintprob`, `XPRBprintvar`.

XPRBprintvar 4

XPRBprintvar

目的

変数を印刷する.

概要

```
int XPRBprintvar(XPRBprob prob, XPRBvar var);
```

引数

prob **現在の問題.**

var **変数への参照**

例

この例は変数abc3[1.000,100.000]**をプリントし，次行に変数**abc4[0.000,5.000,50.000]**をプリントする.**

```
XPRBvar x1, x3;
...
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 1, 100);
XPRBprintvar(prob, x1);
x3 = XPRBnewvar(prob, XPRB_SC, "abc4", 0, 50);
XPRBsetlim(prob, x3, 5);
XPRBprintvar(prob, x3);
```

追加情報

この関数は，変数をプリントする．連続変数，２値変数，整数変数については，名前と上下限値.
部分整数変数，半連続変数，半連続整数変数については，名前と整数限界値または下半連続限界値.
解が計算されているときは，解の値もプリントされる.

関連事項

XPRBprintctr, XPRBprintidxset, XPRBprintprob, XPRBprintsos.

XPRBreadarrline

目的

データファイルから配列を読み込む.

概要

```
int XPRBreadarrline(FILE *file, int length,
const char *format, void *arrc, int size);
```

引数

file データファイルへのポインタ

length 読み込む文字列の最大長さ

format 読み込むデータファイルのフォーマットを指定する文字列，以下のいずれか1つの後に区切り記号をつける.

t[num] テキストを区切り記号または空白（ブランク／タブ／ラインブレイク）まで読む．後ろに数字をつけて，最大読み込み文字数を指定することができる;

s[num] シングルクォート(′ ′)で囲まれたテキスト．後ろに数字をつけて，最大読み込み文字数を指定することができる;

S[num] ダブルクォート (" ")で囲まれたテキスト．後ろに数字をつけて，最大読み込み文字数を指定することができる;

T[num] テキスト．t, sやSに関して，読み込んだ最初の文字に依存する．後ろに数字をつけて，最大読み込み文字数を指定することができる;

ｉ整数値;

g 実数（浮動小数）値.

arrc 読み込んだデータを入れる配列．少なくともより大きいこと．.

size 読み込むデータの最大数.

例

```
double vlist[10];
FILE *datafile;
...
datafile=fopen("filename","r");
XPRBreadline(datafile,120,"g ",vlist,6);
fclose(datafile);
```


この例は，データファイルを開き，スペースで区切られた6個の倍精度データを読み込み，ファイルをクローズする.

追加情報

この関数は，モデル作成者が指定したフォーマット形式のデータファイルからデータを読み込む．データ行は，継続記号&を行末につけることで，複数行にわたって記述可能である．入力ファイルはコメント行（!で始まる）や空白行を含むことができる．これらの行は読みとばされる．データファイルは標準的なC言語の関数(fopen, fclose)によってアクセスされる．この関数は指定されたフォーマットに基づき，最大size個のデータを読み込む．フォーマットを指定する文字の後には最大読み込み文字数をオプションでつけることができる．この指定がないときは，lengthが最大読み込み文字数として使われる．区切り文字(e.g. , ; :)をフォーマットを指定する文字の後に指定することが必要である．配列arrcは読み込むデータと同じ型 (int *, char *, or double *)であり，かつ少なくともsizeだけのスペースをもたないといけない．関数XPRBsetdecsignを利用すると，読み込みデータ中の小数点の変換（カンマを小数点につかっているデータなど）を行う

ことができる.

関連事項

XPRBprintf, XPRBreadline, XPRBsetdecsign.

Xpress-BCL Reference Manual BCL Library Functions 113

XPRBreadline

目的

固定フォーマット形式でのデータファイルからの読み込み

概要

```
int XPRBreadline(FILE *file, int length,
const char *format, ...);
```

引数

file データファイルへのポインタ

length 読み込む文字列の最大長さ

format 読み込むデータファイルのフォーマットを指定する文字列，以下のいずれか1つの後に区切り記号をつける．

t[num] テキストを区切り記号または空白（ブランク／タブ／ラインブレイク）まで読む．後ろに数字をつけて，最大読み込み文字数を指定することができる;

s[num] シングルクォート(′ ′)で囲まれたテキスト．後ろに数字をつけて，最大読み込み文字数を指定することができる;

S[num] ダブルクォート (" ")で囲まれたテキスト．後ろに数字をつけて，最大読み込み文字数を指定することができる;

T[num] テキスト．t, sやSに関して，読み込んだ最初の文字に依存する．後ろに数字をつけて，最大読み込み文字数を指定することができる;

ｉ整数値;

g 実数（浮動小数）値.

これらのフォーマットパラメータはいくつでも続けて書くことができる．

... 読み込んだデータをいれる変数などをカンマで区切っていれる．

例

この例は，データファイルをオープンし，セミコロン（;）で区切られたテキストと倍精度数値を読み込み，次に，ブランクで区切られた整数を２つと最大10文字のシングルクォートで囲まれた文字列を読み込み，ファイルを閉じる．

```
double value;
FILE *datafile;
char name[100];
int i[2];
...
```



```
datafile=fopen("filename","r");
XPRBreadline(datafile,99,"T;g",name,&value);
XPRBreadline(datafile,50,"i i s[10]",&i[0],&i[1],name);
fclose(datafile);
```

追加情報

この関数は，モデル作成者が指定したフォーマット形式のデータファイルからデータを読み込む．データ行は，継続記号&を行末につけることで，複数行にわたって記述可能である．入力ファイルはコメント行（!で始まる）や空白行を含むことができる．これらの行は読みとばされる．データファイルは標準的なC言語の関数(fopen, fclose)によってアクセスされる．フォーマットを指定する文字の後には最大読み込み文字数をオプションでつけることができる．この指定がないときは，lengthが最大読み込み文字数として使われる．区切り文字(e.g. , ; :)をフォーマットを指定す

る文字の間に指定することが必要である．C言語の関数の printf **やscanf のように，フォーマット文の後に，データを格納する変数などを指定する．関数XPRBsetdecsignを利用すると，読み込みデータ中の小数点の変換（カンマを小数点につかっているデータなど）を行うことができる．**

関連事項

XPRBprintf, XPRBreadarrline, XPRBsetdecsign.

Xpress-BCL Reference Manual BCL Library Functions 115

XPRBsavebasis

目的

現在の基底解を保存する

概要

```
XPRBbasis XPRBsavebasis(XPRBprob prob);
```

引数

`prob` **現在の問題.**

例

```
XPRBprob expl2;
XPRBbasis basis;
expl2 = XPRBnewprob("example2");
XPRBsolve(expl2,"l");
basis = XPRBsavebasis(expl2);
```

この例は，現在の基底解を保存する

追加情報

この関数は現在の基底解を保存する．保存した基底解は関数`XPRBloadbasis`** によって，再読み込みできる．これら2つの関数は，メモリー中で基底を保存しておくためのものだが，基底解をファイルに書き出すためには，**Optimizer library**の**`XPRSwritebasis`**を利用する．基底の保存のためメモリーを割り当てる必要はないが，保存した基底が必要なくなったら，**`XPRBdelbasis`**で削除しておくことに注意する．基底を保存したり再読み込みするときは，正常に稼動させるため，線形計画法の事前解法や整数の前処理をオフにしておくこと.**

関連事項

`XPRBdelbasis`, `XPRBloadbasis`, `XPRSreadbasis` (see Optimizer Reference Manual), `XPRSwritebasis` (see Optimizer Reference Manual).

XPRBsetarrvarel 4

XPRBsetarrvarel

目的

変数の配列の指定した位置に要素を追加する.

概要

```
int XPRBsetarrvarel(XPRBprob prob, XPRBarrvar av, int ndx,
XPRBvar var);
```

引数

prob **現在の問題.**

av **配列への参照**

ndx **配列中のインデックス**

var **追加する変数**

例

```
XPRBarrvar av2;
XPRBvar x1;
...
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
av2 = XPRBstartarrvar(prob,5,"arr2");
XPRBsetarrvarel(prob,av2,3,x1);
```

この例は変数x1**を配列**av2**の**4**番目に追加する.（配列の番号は**0**から始まる）**

追加情報

この関数は変数の配列の指定した位置に要素を追加する.指定した位置に既に変数がある場合は,上書きする.

関連事項

XPRBapparrvarel, XPRBdelarrvar, XPRBendarrvar, XPRBnewarrvar, XPRBstartarrvar.

XPRBsetctrtype 4

XPRBsetctrtype

目的

制約式の型を指定する．

概要

```
int XPRBsetctrtype(XPRBprob prob, XPRBctr ctr,
char qrtype);
```

引数

prob **現在の問題**.

ctr **制約式への参照**

qrtype **制約式の型．以下のいずれか**:

XPRB_L **以下**;

XPRB_G **以上**;

XPRB_E **等式**;

XPRB_N **制約なし(目的関数)**.

例

```
XPRBctr ctr1;
...
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
XPRBsetctrtype(prob, ctr1, XPRB_L);
```

この例は制約式ctr1**の型を '以下'に変更する．**

追加情報

この関数は既定義の制約式の型を変更する．関数XPRBsetrange**は範囲制約への変更に使われる．**
もし，範囲制約がこの関数によって型を変更されると，もともとの上限が右辺項になる．

関連事項

XPRBgetctrtype, XPRBnewctr, XPRBsetrange, XPRBsetterm.

XPRBsetdecsign

目的

入力データの小数点の記号を選択する．

概要

```
int XPRBsetdecsign(char sign);
```

引数

`sign` **小数点に使う記号．通常は** '.' (default), **または** ',' .

例

```
XPRBsetdecsign(',');
```

この例は，カンマを小数点として使うように変更する．

追加情報

外部ファイルからデータを読み込むとき，BCL**はデフォルトでは，アングロ－アメリカン小数点を利用する．この関数は，データ入力を**`XPRBreadline`**や**`XPRBreadarrline`**などの関数によって行うとき，小数点の記号をカンマや他の**ASCII**文字に切り替えるとき利用する．出力のほうは，常にデフォルトの小数点を使う．**

関連事項

`XPRBprintf`, `XPRBreadarrline`, `XPRBreadline`.

XPRBsetdir 4

XPRBsetdir

目的

変数の分枝の方向を設定する．

概要

```
int XPRBsetvardir(XPRBprob prob, XPRBvar var, int type,
double c);
```

引数

prob **現在の問題**.

var **変数への参照**.

type **分枝方向の型．以下のいずれか**:

XPRB_PR **優先度付け**;

XPRB_UP **上方向へ分枝**;

XPRB_DN **下方向へ分枝**;

XPRB_PU **上方向分枝の擬コストを設定する**;

XPRB_PD **下方向分枝の擬コストを設定する**.

c **上の型に応じて決まる引数．以下のいずれか**:

XPRB_PR **優先度—** 1 **(最高)から**1000 **(最低)の間の整数, デフォルトは**1000;

XPRB_UP **入力の必要なし— 任意の値，例えば**0**に設定**;

XPRB_DN **入力の必要なし— 任意の値，例えば**0**に設定**;

XPRB_PU **上方向分枝の擬コスト**;

XPRB_PD **下方向分枝の擬コスト**.

例

以下の例は，変数x1**に優先度**10**を与え，上方向の分枝を設定している．**

```
XPRBvar x1;
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
XPRBsetvardir(prob, x1, XPRB_PR, 10);
XPRBsetvardir(prob, x1, XPRB_UP, 0);
```

追加情報

1. **この関数は**Xpress-MP**で利用できる全ての分枝の型を設定する．すなわち，変数の分枝の優先度 (型** XPRB_PR**)，優先的に行う分枝の方向(型** XPRB_UP, XPRB_DN**)，変数の分枝によるコストの**

推定値 (型 XPRB_PU, XPRB_PD**)である．1つの変数に対し，複数のことなる分枝の型を設定できる．**

2. **分枝の方向を設定できるのは離散変数のみである(準連続変数，部分整数変数をふくむ)．関数** XPRBsetsosdir **はSOSの方向を設定するのに使う．**

関連事項

XPRBcleardir, XPRBsetsosdir.

XPRBseterrctrl 4

XPRBseterrctrl

目的

エラー時の挙動を選択する.

概要

```
int XPRBseterrctrl(int flag)
```

引数

`flag` **エラー処理のための指示値. 以下のいずれか:**

`0` **BCLではエラー処理しない;**

`1` **エラー時, プログラムは終了する（デフォルト）.**

例

以下の例は, エラー処理を利用者自身のプログラムで処理するように変更する.

```
XPRBseterrctrl(0);
```

追加情報

この関数は, BCLがエラー処理を行うかどうかを制御する. デフォルトでは, エラーが起こった時点で処理は終了する. もしBCLによるエラー処理をしない場合は, 利用者は自身のプログラム中で各関数の返り値を調べて, 起こりうるエラーのチェックをしなければならない. BCLプログラムが他の大きいアプリケーションに埋め込まれて使われるときには, この設定をしておくほうがよい. コールバック関数XPRBdefcberrはエラーメッセージを獲得するために定義される.

関連事項

`XPRBdefcberr`, `XPRBgetversion`.

XPRBsetlb 4

XPRBsetlb

目的

下限値を設定する.

概要

```
int XPRBsetlb(XPRBprob prob, XPRBvar var, double bdl);
```

引数

prob 現在の問題.

var 変数への参照

bdl 新しい下限値

例

この例は変数x1の下限値を3に設定する.

```
XPRBvar x1;
...
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 1, 100);
XPRBsetlb(prob, x1, 3.0);
```

追加情報

この関数は変数の下限値を設定する.

関連事項

XPRBfixvar, XPRBgetbounds, XPRBgetlim, XPRBsetlim, XPRBsetub.

XPRBsetlim 4

XPRBsetlim

目的

半整数変数の整数限界値を設定する．または，準連続変数や準連続整数変数の下側半連続限界値を設定する．

概要

```
int XPRBsetlim(XPRBprob prob, XPRBvar var, double c);
```

引数

prob **現在の問題.**

var **変数への参照**

c **限界値**

例

```
XPRBvar x3;
...
x3 = XPRBnewvar(prob, XPRB_SC, "abc4", 0, 50);
XPRBsetlim(prob, x3, 5);
```

この例は変数x3**の限界値を**5**に設定する．従って，** x3**のとりうる値は，** x3 = 0 , or 5 <= x3 <= 50.

追加情報

この関数は部分整数変数の整数限界値 (i.e. **連続部分の下限値**) **を設定する．または，準連続変数や準連続整数変数の準連続下限値を設定する．**

関連事項

XPRBfixvar, XPRBgetbounds, XPRBgetlim, XPRBsetlb, XPRBsetub.

XPRBsetmodcut

目的

制約式の型を設定する.

概要

```
int XPRBsetmodcut(XPRBprob prob, XPRBctr ctr, int mcstat);
```

引数

`prob` **現在の問題**.

`ctr` **制約式への参照**.

`mcstat` **制約式の型．以下のいずれか**:

`0` **（通常の）制約**;

`1` **モデルカット**.

例

この例は制約式 `ctr1` **の型をモデルカットにする.**

```
XPRBctr ctr1;
...
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
XPRBsetmodcut(prob,ctr1,1);
```

追加情報

この関数は，制約式の型を，通常の制約からモデルカットに変更する.

関連事項

`XPRBdelctr`, `XPRBgetmodcut`, `XPRBnewctr`.

XPRBsetmsglevel 4

XPRBsetmsglevel

目的

メッセージのプリントレベルを設定する．

概要

```
int XPRBsetmsglevel(XPRBprob prob, int level);
```

引数

`prob` **現在の問題.**

`level` **メッセージレベル．以下のいずれか：**

`0` **メッセージをプリントしない**;

`1` **エラーメッセージのみをプリントする**;

`2` **警告とエラーメッセージをプリントする（デフォルト）**;

`3` **全てのメッセージをプリントする．**

例

この例は，エラーメッセージのみをプリントするように切り替える．

```
XPRBsetmsglevel(prob,1);
```

追加情報

1. BCL **は異なる型のメッセージをプリントする：状態情報，警告，エラーなどである．この関数は出力するメッセージを制御する．**
2. 1**以上に設定すると，**Optimizer **の出力も表示される．**

関連事項

`XPRBdefcbmsg`, `XPRBsetdecsign`.

XPRBsetobj

目的

目的関数を選択する.

概要

```
int XPRBsetobj(XPRBprob prob, XPRBctr ctr);
```

引数

prob **現在の問題.**

ctr **制約式への参照**

例

```
XPRBctr ctr3;
XPRBarrvar tobj;
...
tobj = XPRBnewarrvar(prob, 10, XPRB_PL, "tabo", 0, 800);
ctr3 = XPRBnewsum(prob, "r3", tobj, XPRB_N, 0);
XPRBsetobj(prob,ctr3);
```

この例は制約式ctr3**を目的関数に設定する.**

追加情報

この関数は，目的関数を設定する．この設定は，全ての最適化計算が終わっているときになされなければならない．通常は，目的関数になる制約式は，型XPRB_N**をもつが，他の型の制約式でもよい．その場合は，その制約式の満たす等式や不等式は問題の一部として取り扱われる.**

関連事項

XPRBgetsense, XPRBsetsense.

XPRBsetqterm 4

XPRBsetqterm

目的

目的関数中に２次項を設定する.

概要

```
int XPRBsetqterm(XPRBprob prob, XPRBvar var1, XPRBvar var2,
double coeff);
```

引数

prob **現在の問題**.

var1 **変数への参照**

var2 **変数への参照** (**上と同じでもよい**).

coeff var1 * var2**への係数**

例

```
XPRBvar x2;
x2 = XPRBnewvar(prob,XPRB_PL,"abc1",0,XPRB_INFINITY);
XPRBaddqterm(prob,x2,x2,5.2);
```

この例は目的関数に係数を5.2**の項**x2*x2**を設定する.**

追加情報

この関数は目的関数中の2**次項の係数を**coeff**に設定する.**

関連事項

XPRBaddqterm, XPRBdelqobj.

XPRBsetrange

目的

範囲制約を設定する.

概要

```
int XPRBsetrange(XPRBprob prob, XPRBctr ctr, double bdl,
double bdu);
```

引数

`prob` **現在の問題.**

`ctr` **制約式への参照**

`bdl` **下限値**

`bdu` **上限値**

例

この例は等式制約だった `ctr2` **を範囲制約に変更する.**

constraint `4.0 <= sum(i=0:4) ty1[i] <= 15.5`.

```
XPRBctr ctr2;
XPRBarrvar ty1;
...
ty1 = XPRBnewarrvar(prob, 5, XPRB_PL, "arry1", 0, 500);
ctr2 = XPRBnewsum(prob, "r2", ty1, XPRB_E, 9);
XPRBsetrange(prob, ctr2, 4.0, 15.5);
```

追加情報

この関数は定義済みの制約式の型を, `bdl` **と** `bdu`**で決まる範囲制約に変更する. 制約式の型は** `XPRB_R`**になり, 右辺項は2つの値で置き換わる.**

関連事項

`XPRBgetctrtype`, `XPRBgetrange`, `XPRBsetctrtype`.

XPRBsetsense

目的

目的関数の最大化，最小化を決める．

概要

```
int XPRBsetsense(XPRBprob prob, int dir);
```

引数

prob **現在の問題**.

dir **目的関数の方向．以下のいずれかである**:

XPRB_MAXIM **最大化する**;

XPRB_MINIM **最小化する**

例

```
XPRBprob expl2;
...
expl2 = XPRBnewprob("example2");
XPRBsetsense(expl2,XPRB_MAXIM);
```

この例はexpl2 **を最大化問題に設定する．**

追加情報

この関数は目的関数の最大化，最小化を決める．デフォルトでは，問題は最小化に設定される．

関連事項

XPRBgetsense, XPRBsetobj.

XPRBsetsosdir

目的

SOSの分枝方向を設定する．

概要

```
int XPRBsetsosdir(XPRBprob prob, XPRBsos sos, int type,
double val);
```

引数

prob 現在の問題.

sos SOSへの参照

type 方向の型．以下のいずれか:

XPRB_PR 優先度付け;

XPRB_UP 上方向へ分枝;

XPRB_DN 下方向へ分枝;

XPRB_PU 上方向分枝の擬コストを設定する;

XPRB_PD 下方向分枝の擬コストを設定する.

val 上の型に応じて決まる引数．以下のいずれか:

XPRB_PR 優先度— 1 (最高)から1000 (最低)の間の整数, デフォルトは1000;

XPRB_UP 入力の必要なし— 任意の値，例えば0に設定;

XPRB_DN 入力の必要なし— 任意の値，例えば0に設定;

XPRB_PU 上方向分枝の擬コスト;

XPRB_PD 下方向分枝の擬コスト.

例

```
XPRBsos set1;
...
set1 = XPRBnewsos(prob, "sos1", XPRB_S1);
XPRBsetsosdir(prob, set1, 5);
XPRBsetsosdir(prob, set1, XPRB_DN, 0);
```

この例では，SOS set1に優先度5を与え，set1に下方向の分枝を優先するように設定する．

追加情報

この関数はXpress-MPで利用できる全ての分枝の型を設定する．すなわち，SOSの分枝の優先度 (型 XPRB_PR)，優先的に行う分枝の方向(型 XPRB_UP, XPRB_DN)，変数の分枝によるコストの推定値 (型 XPRB_PU, XPRB_PD)である．1つの変数に対し，複数のことなる分枝の型を設定できる．

関連事項

XPRBcleardir, XPRBsetdir.

XPRBsetterm 4

XPRBsetterm

目的

制約式の項を設定する.

概要

```
int XPRBsetterm(XPRBprob prob, XPRBctr ctr, XPRBvar var,
double coeff);
```

引数

prob **現在の問題.**

ctr **制約式への参照**

var **変数への参照. 必要なければ**NULL**でもよい.**

coeff **変数**var**に対する係数の値.**

例

```
XPRBctr ctr1;
...
ctr1 = XPRBnewctr(prob, "r1", XPRB_E);
XPRBsetterm(prob, ctr1, NULL, 7.0);
```

この例は制約式ctr1**の右辺項を**7.0**に設定する.**

追加情報

この関数は, 変数の係数をcoeff**に設定する. もし**var**が**NULL**に設定されれば,** coeff**は右辺項として設定される.**

関連事項

XPRBaddterm, XPRBdelctr, XPRBnewctr.

XPRBsetub

目的

上限値を設定する.

概要

```
int XPRBsetub(XPRBprob prob, XPRBvar var, double bdu);
```

引数

prob 現在の問題.

var 変数への参照

bdu 変数への新しい上限値.

例

この例は，変数x1の上限値を200に設定する.

```
XPRBvar x1;
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 1, 100);
XPRBsetub(prob, x1, 200.0);
```

追加情報

この関数は，変数の上限値を設定する.

関連事項

XPRBfixvar, XPRBgetbounds, XPRBgetlim, XPRBsetlb, XPRBsetlim.

XPRBsetvartype

目的

変数の型を設定する.

概要

```
int XPRBsetvartype(XPRBprob prob, XPRBvar var, int type);
```

引数

prob **現在の問題**.

var **変数への参照**

type **変数の型. 以下のいずれか**:

XPRB_PL **連続**;

XPRB_BV **2値**;

XPRB_UI **整数**;

XPRB_PI **部分整数**;

XPRB_SC **準連続**;

XPRB_SI **準連続整数**.

例

この例は変数x1**の型を整数から**2**値に変更する. この結果, この変数の上限値は**1**になる.**

```
XPRBvar x1;
x1 = XPRBnewvar(prob, XPRB_UI, "abc3", 0, 100);
XPRBsetvartype(prob, x1, XPRB_BV);
```

追加情報

この関数は, 既にある変数の型を変更する.

関連事項

XPRBgetvarname, XPRBgetvartype, XPRBnewvar.

XPRBsolve

目的

Xpress-Optimizerを呼び出す．

概要

```
int XPRBsolve(XPRBprob prob, char *alg);
```

引数

`prob` 現在の問題.

`alg` 解法のアルゴリズムを以下から選択する：

`p` 主単体法;

`d` 双対単体法;

`b` ニュートンバリア法;

`l` （緩和）線形問題のみを解く;

`g` 大域最適問題 (MIP) を解く;

なし — デフォルトの設定を使う．

例

この例はは，問題`expl2`を主単体法でMIPとして解く．大域解があるものと仮定している．

```
 XPRBprob expl2;
...
expl2 = XPRBnewprob("example2");
XPRBsolve(expl2,"pg");
```

追加情報

この関数は，Xpress-Optimizerのアルゴリズムを選択して実行する． アルゴリズムを指定する文字は組み合わせてつかうことができる，e.g. "dg"．もし，Optimizerにロードされている行列が，現在の問題と一致しないときは，行列が最初に作り直される．最適化の方向（デフォルトは最小化）は関数`XPRBsetsense`で変更できる．問題を解く前に，関数`XPRBsetobj`によって目的関数を指定しておかねばならない．もし不完全全域探索を行ったときは，そのプログラムを終了するとき，関数`XPRSinitglobal`を呼んで，探索木の情報を全て削除しておくことに注意する．そうしておかないと，その後そのプログラムを動かすことができなくなる．

関連事項

`XPRBgetsense`, `XPRBmaxim`, `XPRBminim`, `XPRBsetsense`.

XPRBstartarrvar 4

XPRBstartarrvar

目的

変数の配列の定義を開始する．

概要

```
XPRBarrvar XPRBstartarrvar(XPRBprob prob, int nbvar,
char *name);
```

引数

prob **現在の問題.**

nbvar **配列中の変数の最大個数**

name **配列の名前．**NULL**でもよい．**

例

```
XPRBarrvar av2;
...
av2 = XPRBstartarrvar(prob,5,"arr2");
```

この例は，arr2**という名の5個の要素をもつ変数配列の定義をする．**

追加情報

この関数は，変数配列の定義を開始する．この関数の返り値は配列への参照であり，それは，制約式の定義などで使われる．作成する配列に属する変数は既定義TのLP**変数である．変数の型は違っていてもよい，順番も任意である．一つの変数が，いくつもの配列に属してもよい．ただし，変数の定義は一度だけ行う（関数**XPRBnewvar**や**XPRBnewarrvar**）．もし，指定した名前が既に使われていれば，**BCLはその名の後に自動的に連番をつける． **名前を省略したときは**AV**で始まる連番を付けた名前を**BCL**がつける．**

関連事項

XPRBdelarrvar, XPRBendarrvar, XPRBnewarrvar.

XPRBstartarrvar 4
Xpress-BCL Reference Manual BCL Error Messages 139
Error Codes

1502 – 1514 5

5 BCL Error Messages

BCLのエラーには2種類ある．‘Error’とマークされているものは，プログラムの実行を中断する．‘Warning’とマークされているものは，プログラムを中断しない．‘fct’ という表示は，エラーが起こった関数の名前がそこに表示されることを示す．

1502 Error *Not enough memory*.
メモリー不足で，BCLのオブジェクトにメモリーを割り当てられない．

1505 Error *(fct) No variable given*.
関数 *fct* は引数として型XPRBvarの変数を要求する． 変数が既に生成されているかどうかチェックする． (functions XPRBnewvar or XPRBnewarrvar).

1506 Error *(fct) No array of variables given*.
関数 *fct* は引数に変数の配列を要求する．配列が既に生成されているかチェックする．(function XPRBnewarrvarまたはXPRBstartarrvarまたはXPRBendarrvar).

1507 Error *(fct) No constraint given*.
関数 *fct* は引数に制約式を要求する．制約式が既に生成されているかチェックする．(functions XPRBnewctr,XPRBnewsum, XPRBnewarrsum or XPRBnewprec).

1508 Error *(fct) No SOS given*.
関数 *fct* は引数に SOS を要求する．既に生成されているかチェックする (functions XPRBnewsos, XPRBnewsosrc,or XPRBnewsosw).

1512 Error *(fct) No array of constants given*.
関数 *fct* は引数に制約式の配列を要求する．

1513 Warning *(fct) No variable given*.
関数 *fct* は引数に型XPRBvarを要求する．コマンドは無視される．

1514 Warning *(fct) No constraint given*.
関数 *fct* は引数に制約式を要求する．コマンドは無視される．

Xpress-BCL Reference Manual BCL Error Messages 140
Error Codes

1515 – 1535 5

1515 Error *(fct) Problem has no* ‘name’.
問題の定義が不完全 (少なくとも，一つの変数か，一つの制約式と，非零の目的関数が定義されなければならない).

1516 Warning *(fct) Problem has no* ‘name’.
問題の定義が不完全 (少なくとも，一つの変数か，一つの制約式と，非零の目的関数が定義されなければならない).

1518 Warning *(fct) No SOS given*.
関数 *fct* は引数に Special Ordered Set を要求する．コマンドは無視される．

1520 Warning *(fct) No solution available or problem modified since last solved*.
関数 *fct* は解の情報を得ることができない．

1521 Error *Xpress-Optimizer error getting objective function value*.
目的関数値をXpress-Optimizerが獲得できない．

1522 Error *Xpress-Optimizer error getting* ‘name’ *status*.
Xpress-Optimizer による解の情報が得られない．

1523 Error *Unknown solution option* ‘char’.

XPRBsolve に ‘b’, ‘d’, ‘g’, ‘l’, ‘p’**以外のオプションが設定された．**
1524 Error *(fct) Xpress-Optimizer error* num *during* ‘name’.
Xpress-Optimizer**が関数***name***を実行中にエラーが起こった．**
Optimizer Reference **のエラー番号***num***を参照**
1525 Warning *(fct) Different problem loaded in the Xpress-Optimizer*.
Xpress-Optimizer **でまだロードされていない問題の（解）情報を，得ようとした．この情報にアクセスするためには，問題を解く必要がある．**
1530 Error *(fct) Inconsistent bounds for variable* ‘name’ (bdl,bdu).
変数に対し，上限値より大きいか現地を設定しようとした．
1531 Error *(fct) Incorrect type for variable* ‘name’.
変数に間違った型を設定しようとした，または，型を設定することをしなかった．
1535 Error *(fct) Incorrect type for constraint* ‘name’.
制約式に，間違った型を設定しようとした，または，型を設定することをしなかった．

Error Codes

1536 – 1562 5

1536 Error *(fct) Inconsistent range for constraint* ‘name’ (bdl, bdu).
制約式の下限値が上限値より大きく設定された．
1540 Error *(fct) Trying to modify a closed array of variables* (‘name’).
変数配列に対する変更は，XPRBendarrvar**で定義を終わったあとでは，できない．**
1541 Error *(fct) Index num1 out of range for an array of variables* (‘name’ *max* = num2)
配列の容量を越えて変数を格納しようとした，または，参照しようとした．
1542 Error *(fct) Trying to add an entry* (‘name’) *to a complete array of variables* (‘name’).
変数の配列の容量より多くの変数を追加することはできない．
1543 Error *(fct) Trying to close an incomplete array of variables* (‘name’).
関数XPRBendarrvar**で配列定義を終わろうとしたとき，まだ変数定義がされていない変数が含まれていた．**
1545 Error *(fct) Wrong type for SOS* ‘name’.
SOS**に対し，間違った方を指定した，または，型を指定しなかった．**
1547 Error *(fct) Wrong directive type*.
探索方向の型を間違ってしてした，または型を指定しなかった．
1550 Error *(fct) No index set given*.
関数 *fct* **は引き数にインデックス集合を要求している．** (function XPRBnewidxset).
1551 Warning *(fct) No name given for an element of an index set*.
関数 *fct* **は引き数にインデックスの名前を要求している．このコマンドは無視される．**
1552 Warning *(fct) No index set given*.
関数 *fct* **は引き数にインデックス集合を要求している．このコマンドは無視される．**
1561 Error *(fct)* ‘XPRSinit’ *failed* (*value:* num).
Xpress-Optimizer **を初期化できなかった．** (error code *num*).
1562 Warning *(fct) Switching to Student Edition*.
初期化時に，有効なライセンスを見つけられなかった．BCL **は学生版として稼動する．**

Error Codes

1563 – 1593 5

1563 Error *(fct) Inconsistency during matrix generation*.
行列生成のときに，内部でエラーが起こった．

1565 Error *(fct) Internal error.*
行列生成のときに，内部でエラーが起こった．
1566 Error *Name too long:* ‘name’.
利用者定義，あるいはBCLが自動作成した名前が最大長さを超えた．
1567 Error *(fct) Size limits of the Student Edition exceeded*.
指定したモデルは，学生版BCLでは大きすぎて解けない．
1568 Warning *Operation* fct *not allowed in Student Edition*.
関数 *fct* は学生版ではりようできない．
1570 Warning *XOSL:* text.
Optimizer Reference Manual**の指定した番号のエラーを参照．**
1580 Warning *Unknown output file format* format.
関数`XPRBexportprob`**の記述を参照．**
1582 Error *Internal error writing MPS file*.
1591 Error *(fct) Non-quadratic term*.
目的関数が2次以上の項を含む．
1593 Warning *Ignoring quadratic terms in MIP objective*.
BCL**は現時点では，2次計画問題は連続問題のみ解くことができる．**

付録 A — OptimizerライブラリによるBCLの利用

BCLはモデリングの機能（これはXpress-Mosel,やXpress-Modelerの機能に対応）と基本的な最適化計算の機能（Xpress-Optimizerライブラリのコンソールモードでの機能に対応）を提供する．もし利用者が，より進んだOptimizerの機能を利用するとか，もっと情報を得たいとか，アルゴリズムの設定を変えたい，などの要求をもったときは，Optimizerライブラリ関数を直接使うべきである．

以下の節では，BCLプログラムの中からOptimizerライブラリ関数を使う方法を詳細に説明する．最初の節ではBCLと互換性のある関数の一覧を挙げる．次に初期化や行列の読み込みやインデックスの使用についての一般的な注意をし，最後の節ではBCLのプログラムの中からOptimizerの関数を利用する例をいくつか挙げる．

重要事項: Optimizerライブラリ関数を利用するときは，Optimizerのヘッダファイルを，BCLのヘッダファイルと共にインクルードしなければならない．プログラムの最初に以下の2行を付け加える．

```
#include "xprb.h"
#include "xprs.h"
```

A.1 ライブラリ間の切り替え

Optimizer ライブラリの関数は2種類に分かれる：問題についての情報を得るためのものまたは探索アルゴリズムの設定を変えるものと，問題そのものの定義を変更するものである．前者のタイプの関数は，BCLのプログラムから何の問題もなく利用できる．後者のタイプを使うときは，利用者は完全にOptimizerのほうに切り替えをしなければならない（例えば，問題の定義をBCLの方で完全に終え，行列をOptimizerに読み込ませた後でなければ，これらの関数を利用できない）．

BCL互換の Optimizer 関数

以下のOptimizer ライブラリ関数は，BCLで利用可能である（ただし，行や列のインデックスを引数にもつ関数については，若干の注意が必要である．これについては A.4 行列のインデックス，を参照．更に，BCL内では，解の情報は解探索の終了後のみに更新される．）

- ***問題や制御パラメータの設定，情報獲得のための関数***: `XPRSsetintcontrol`, `XPRSgetintcontrol`, `XPRSgetintattrib` etc.;
- ***出力や保存***: `XPRSsave`, `XPRSwritebasis`, `XPRSrange`, `XPRSiis`, `XPRSwriteprtsol`, `XPRSwritesol`, `XPRSwriteprtrange`, `XPRSwriterange`, `XPRSgetsol`, `XPRSwriteomni`, `XPRSwriteprob`, 全てのログ出力関数とコールバック;
- ***情報の獲得***: all functions `XPRSget...`, `XPRSgetrowrange`, `XPRSgetcolrange`;
- ***アルゴリズムの設定***: `XPRSreaddirs`, `XPRSloaddirs`, `XPRSreadbasis`, `XPRSloadbasis`, `XPRSloadsecurevecs`, `XPRSscale`, `XPRSftran`, `XPRSbtran`, all global callbacks;
- *cut manager.*

非互換のOptimizer 関数

以下のOptimizer ライブラリ関数はBCLの入力が終ったあとに使われる:

- ***行列要素の変更追加削除***: `XPRSadd...`, `XPRSchg...`, `XPRSdel...`;
- ***解法のアルゴリズム***: `XPRSminim`, `XPRSmaxim`, `XPRSglobal`;
- ***データ（または問題）の入力***: `XPRSreadprob`, `XPRSloadlp`, `XPRSloadglobal`, `XPRSloadqglobal`, `XPRSloadqp`, `XPRSalter`, `XPRSsetprobname`;
- ***行列の操作***: `XPRSrestore`;

上記の関数が一旦使われると，BCLで保持している問題の情報と，Optimizerが保持している情報に違いが生じてしまい，この違いをBCLのほうで更新することはできない．従って，当該問題の処理は以後Optimizer ライブラリ関数のみを使って行わなければならない．つまり，処理がOptimizer

のほうに完全に切り替わる．このBCLからOptimizer への切り替えは単一の問題についてのみ当てはまる．もしも，他の問題を平行して扱っている場合，この切り替えが行われていない他の問題については，利用者は引き続きBCLの関数を使って処理することができる．
Xpress-BCL Reference Manual Using BCL with the Optimizer Library 145

A.2 初期化と終了

Optimizerの初期化は，BCLと同時に行われるので，プログラムでOptimizerの初期化関数XPRSinitを呼び出す必要はない(実際, BCLはOptimizer を再初期化する．この再初期化によってそれ以前に行われたOptimizer の設定は無効になる)．Optimizer の初期化はXPRBinitを呼ぶことで自動的になされる．BCL の問題ポインタは通常どおりXPRBnewprobによって定義される．
プログラムの終了時，BCLの正常終了ルーチンが適用される．最初にXPRBdelprobによって問題に使われた全てのメモリーが解放され，次にXPRBfreeによって全てのメモリーが解放される．これらのルーチンはOptimizer で使われたメモリーも解放するので，XPRSdestroyprobやXPRSfreeなどの関数を使う必要はない．
以下は標準的なBCLの利用法である．

```
XPRBprob prob;
...
XPRBinit("");
prob = XPRBnewprob("Example1");
... /* define and solve the problem */
XPRBdelprob(prob);
XPRBfree();
```

A.3 行列の読み込み

Optimizer 側で何らかの操作を行うとき，BCLは行列をOptimizer に読み込ませる．つまり，もし利用者がOptimizer のコマンドに切り替えて操作を行いたいとき（例えば，Optimizerを使って最適化計算を行いたいとき），BCL の問題をOptimizer に(関数 XPRBloadmatを使って)読み込ませなければならない．BCL と Optimizer はそれぞれ別々の問題へのポインタを持っているので，BCLから読み込んでできたOptimizer側の問題ポインタをどのようにして獲得するかという問題が生じる．
しかしこれは，関数XPRBgetXPRSprobを呼ぶことで解決する．この関数は要求された（Optimizer側の）問題ポインタを返す．Optimizer側での問題作成XPRScreateprob を呼ぶ必要はない．これは最初にOptimizerへ問題が渡されたとき，自動的に問題が作成されるからである．
Xpress-BCL Reference Manual Loading the Matrix 146
標準的なBCLの利用法:

```
XPRBarrvar x;
...
x = XPRBnewarrvar(prob, 10, XPRB_PL, "x", 0, 100);
... /* (define the rest of the problem) */
XPRBmaxim(prob,""); /* load matrix and maximize LP problem */
for(i=0; i<10; i++) /* print solution values for variables */
printf("%s: %d, ",XPRBgetvarname(prob,x[i]),
XPRBgetsol(prob,x[i]));
```

BCLで問題を入力後，Optimizer に利用を切り替える:

```
XPRBprob bcl;
XPRSprob opt;
XPRBarrvar x;
int i, cols, len;
double sol;
char names;
XPRBinit(""); /* initialize BCL (and the Optimizer library) */
bcl = XPRBnewprob("Example1"); /* create a new problem */
```

```
x = XPRBnewarrvar(bcl, 10, XPRB_PL, "x", 0, 100);
... /* define the rest of the problem */
XPRBloadmat(bcl); /* load the matrix into the Optimizer */
/* get an Optimizer problem pointer */
opt = (XPRSprob)XPRBgetXPRSprob(bcl);
XPRSmaxim(opt,""); /* maximize the LP problem */
XPRSgetintattrib(opt,XPRS_COLS, &cols); /* get the number of
columns */
sol = (double *)malloc(cols * sizeof(double));
XPRSgetsol(opt,sol,NULL,NULL,NULL); /* get entire primal
solution */
XPRSgetintattrib(opt,XPRS_NAMELENGTH,&len); /* get the
length of names (divided by 8) */
names = (char *)malloc((8*len+1)*cols*sizeof(char));
XPRSgetnames(opt,2, names, 0, cols-1); /* get variable names */
for(i=0; i<cols; i++) /* print all solution values */
printf("%s: %g, ", names+((8*len+1)*i), sol[i]);
```


A.4 行列のインデックス

BCLの関数XPRBgetrownumやXPRBgetcolnumによって返される行や列のインデックスは，前処理が行われる前の（ただし空白行や列は取り除かれた）行列における変数ないし制約式の位置を示している．行列の行や列の位置は，前処理のアルゴリズムで変更されるので，もしこれらの前処理アルゴリズムが使用不可になっていなければ（これは制御パラメータPRESOLVEとMIPPRESOLVEによりなされる），BCLのほうで保持している変数や制約式へのインデックスはOptimizerライブラリ関数では決して使われない．同様な状況が他の変数や制約式に関する情報，解や双対変数値に関する情報，について当てはまる．この問題はBCLの内部（つまりBCLの関数のみを使う場合）では起こらない．なぜなら，解の情報は，最適計算が行われ、後処理が行われた後にのみ，利用可能だからである．

この規則の例外は，Optimizer 関数 XPRSgetsolである：この関数は，例えば，Optimizer のコールバック関数の中で，大域的探索の途中で現在の解の値を得るために，BCLの保持している変数や制約式インデックスと組合わせて使うことができる．これは，XPRSgetsolが後処理が済んだ解を返すために可能なのである．

A.5 BCL互換な関数の利用

BCLのプログラムの中で，最も頻繁に使われるOptimizer 関数は，制御パラメータや問題のパラメータを設定したり読み込んだりする関数である．以下に例を示している．制御パラメータは初期化後いつでも設定をすることができる．問題のパラメータは，一旦Optimizerに問題が読み込まれた後問題に関する値を返す．全てのパラメータは初期設定値を持つ．この初期値は，１つのプログラム中でいくつかの問題が解かれるとき, いちいちもとの初期設定値に再設定されることはないので, 注意されたい．

制御パラメータの設定:

```
XPRBinit(""); /* initialize BCL (and the Optimizer library) */
bcl = XPRBnewprob("Example1"); /* create a new problem */
... /* define the problem */
XPRBloadmat(bcl);
opt = (XPRSprob)XPRBgetXPRSprob(bcl);
XPRSsetintcontrol(opt,XPRS_MAXTIME, 60); /* set a time limit of
60 seconds */
```



```
XPRSsetdblcontrol(opt,XPRS_ADDCUTOFF, 0.999); /* set an
```

```
ADDCUTTOFF value */
XPRBmaxim(bcl,""); /* load matrix and maximize LP problem */
```
問題パラメータの獲得:
```
int rows;
...
XPRBinit(""); /* initialize BCL (and the Optimizer library) */
bcl = XPRBnewprob("Example1"); /* create a new problem */
... /* define the problem */
XPRBloadmat(bcl); /* load the matrix into the Optimizer */
opt = (XPRSprob)XPRBgetXPRSprob(bcl);
XPRSgetintattrib(opt,XPRS_ROWS, &rows);
/* get number of rows */
XPRBmaxim(opt,""); /* maximize the LP problem */
```
この他に，よく使われる Optimizer **関数は，解をプリントアウトするためのコールバック関数と，分枝限定のための探索法の設定の関数であろう．**（A.4, “**行列のインデックス**”**を参照**):
```
XPRSprob bcl
XPRSprob opt;
XPRBvar x;
double sol;
void printobjective(XPRSprob my_opt,void *my_object)
{
int num;
XPRSgetintattrib(my_opt,XPRS_INTSOL,&num);/* get number of
the solution */
XPRSgetsol(my_opt,sol,NULL,NULL,NULL); /* get the
solution values */
XPRBprintf("Solution %d: Objective value: %g\n",
num, XPRBgetobjval(bcl));
XPRBprintf("%s: %g\n", XPRBgetvarname(bcl,x),
sol[XPRBgetcolnum(bcl,x)]);
}
int main(int argc, char **argv)
{
int cols;
```

```
XPRBinit(""); /* initialize BCL (and Optimizer library) */
bcl = XPRBnewprob("Example1"); /* create a new problem */
x = XPRBnewvar(bcl,XPRB_BV,"x_1",0,1);/* define variable */
... /* define the rest of the problem */
XPRSsetcbintsol(opt,printobjective,NULL); /* define an
integer solution callback */
XPRBloadmat(bcl); /* load the matrix into the Optimizer */
opt = (XPRSprob)XPRBgetXPRSprob(bcl);
XPRSgetintattrib(opt,XPRS_COLS,&cols); /* get the number of
columns */
/* create the solution array */
sol = (double *)malloc(cols * sizeof(double));
XPRBmaxim(bcl,"g"); /* maximize the global problem */
}
```

付録 B — BCL in C++とJava

B.1 C++でのBCLの利用の概要

BCLのC++ インターフェースは，データ入出力，エラー処理を除く全ての機能を提供する（除外された機能は対応するC言語の関数を利用することができる）．C言語を用いたモデリングにおける対象（変数や，制約式など）はクラスに変換され，対象に作用する関数は，対応するクラスの解法に変換される．

C++を利用するとき，項別の制約式の定義はより簡単になる．制約式は，演算記号 '+', '-', '<=', '==' などを用いて，より普通の式に近い形で記述することができる．

クラスや解法の名前のつけ方は，C++の命名規約に合うように注意しておく．BCLのモデリングの対象 (XPRBprob, XPRBvar, XPRBctr, XPRBsos, XPRBindexSet,XPRBbasis) に対応するC++のクラスは同じ名前をもつ（例外は，XPRBindexSet）．解法の名前は，語頭のXPRBをとったものになる．例えば，関数XPRBgetvarnameはクラスXPRBvarのgetNameという名前の解法になる．

BCLの全てのC++クラスは名前空間dashoptimizationの一部になる．（ショート）クラス名を使うときは，プログラムの先頭に，以下の行を加えることが推奨される．

```
using namespace ::dashoptimization;
```

例

以下は，C++でBCLを利用した例である．ここでは， Xpress Essentials guideに記述された簡単な例を作成している:

```
#include "xprb_cpp.h"
using namespace ::dashoptimization;
int main()
{
```



```
XPRBvar a;
XPRBvar b;
XPRB::init(); // initialize library
XPRBprob p("Example"); // create new problem
a = p.newVar("a",XPRB_PL,0,200); // define variables
b = p.newVar("b",XPRB_PL,0,200);
p.newCtr("First",3*a + 2*b <= 400); // define constraints
p.newCtr("Second",a + 3*b <= 200);
p.setObj(p.newCtr("Profit",a + 2*b)); // objective function
p.print(); // print out problem definition
p.loadMat(); // generate matrix
p.exportProb(XPRB_MPS,"Simple"); // write MPS file
p.maxim(""); // solve the problem
printf("Profit: %g\n",p.getObjVal()); // output solution
printf(" a = %g, b = %g\n",a.getSol(),b.getSol());
XPRB::free();
return 0;
}
```

B.2 JavaによるBCLの利用の概要

C++ インターフェースの場合と同様に，BCLのJavaインターフェースは，データ入出力，エラー処理を除く全ての機能を提供する（除外された機能は対応するJavaの関数を利用することができる）．C言語を用いたモデリングにおける対象（変数や，制約式など）はクラスに変換され，対象に作用する関数は，対応するクラスの解法に変換される．

C++の場合は，Cの関数（プリント出力のためのprintfやXPRBprintfなｄｐ）を利用することができたが，Javaの場合は，Javaの関数のみでプログラムを書かなければならない．さらに， Java

の場合は，C++のときのように，制約式の定義などで通常の代数の演算子を利用することはできない．そのかわりJavaインターフェースはadd やeqlといったシンプルな関数を用意している．これらの関数は，様々な型の多数のパラメータをとることができる．
Javaにおけるクラスや解法の名前は，C++のときと同様なやりかたで決められる．全てのJavaのクラスは，対応するBCLのモデリングオブジェクト（XPRBprob, XPRBvar, XPRBctr, XPRBsos, XPRBindexSet, XPRBbasis) と同じ名前をもつ（XPRBindexSetを除く）．

解法の名前は，語頭のXPRBをとったものになる．例えば，関数XPRBgetvarnameはクラスXPRBvarのgetNameという名前の解法になる．
全ての Java BCL クラスは，パッケージcom.dashoptimizationに含まれる．ショート）クラス名を使うときは，プログラムの先頭に，以下の行を加えることが推奨される．

```
import com.dashoptimization.*;
```

例

以下は，C++でBCLを利用した例である．ここでは， Xpress Essentials guideに記述された簡単な例を作成している:

```
import com.dashoptimization.*;
public class Example
{
static XPRBprob p;
public static void main(String args[])
{
XPRBvar a;
XPRBvar b;
XPRB.init();
p = new XPRBprob("Example");
a = p.newVar("a",XPRB.PL,0,200);
b = p.newVar("b",XPRB.PL,0,200);
p.newCtr("First",a.mul(3).add(b.mul(2)).lEql(400));
p.newCtr("Second",a.add(b.mul(3)).lEql(200));
p.setObj(p.newCtr("Profit",a.add(b.mul(2))));
p.print();
p.exportProb(XPRB.PL,"Simple");
p.maxim("");
System.out.println("Profit: "+p.getObjVal());
System.out.println(" a = "+a.getSol());
System.out.println(" b = "+b.getSol());
```



```
XPRB.free();
}
}
```

B.3 C++ と Java のクロスレファレンス

BCL のC++ および Java インターフェースのクラスは以下にまとめたとおりである．
XPRB: 初期化，一般設定，全てのパラメータの設定
XPRBprob: 問題定義，モデリングオブジェクトの生成，削除，求解，設定変更，解情報の獲得
XPRBvar: 変数の変更，値獲得の方法
XPRBctr: 制約式の変更，値獲得，生成のための方法
XPRBsos: SOS変数の変更，値獲得，生成のための方法
XPRBindexSet: インデックス集合へのアクセス，要素の追加，獲得の方法
XPRBbasis: 基底解へのアクセスの方法

XPRBlinExp: 線形式の生成の方法
XPRBquadExp: 2次式の生成の方法
XPRBlinRel: 線形式から線形関係を生成する方法
方法isValidは少し説明を要する：この方法はgetVarByNameやgetCtrByNameなどと組み合わせて使われる．これらの方法は，通常の他の方法と違い（通常の方法は，オブジェクトが見つからないときはNULLポインタを返す），要求された型のオブジェクトを返す．方法isValidによってのみ，そのオブジェクトが有効なオブジェクトかどうか（問題定義にふくまれているかどうか）を調べることができる
この節の一覧にあるクラスのほとんどの方法は，標準のBCLのCの関数を呼び出し，その結果を返す．これらの方法は，C++とJavaに共通である．リスト上ではC++での記法のみが与えられている（C++の型charまたはchar * はJavaではStringに置き換わる．また，型boolはboolean になる．さらに，Javaではポインタは存在しない）

クラス XPRBprob

構成:

```
XPRBprob()
calls XPRBnewprob
XPRBprob(char *name)
calls XPRBnewprob
```

方法:

```
XPRBprob *Cref() (C++ only)
XPRBvar newVar(char *name, char type, double lob, double upb)
calls XPRBnewvar
XPRBvar newVar(char *name, char type)
calls XPRBnewvar
XPRBvar newVar(char *name)
calls XPRBnewvar
XPRBvar newVar()
calls XPRBnewvar
XPRBctr newCtr(char *name, XPRBlinRel& ac)
calls XPRBnewctr
XPRBctr newCtr(char *name)
calls XPRBnewctr
XPRBctr newCtr()
calls XPRBnewctr
XPRBctr newCtr(XPRBlinRel& ac)
calls XPRBnewctr
void delCtr(XPRBctr& ctr)
calls XPRBdelctr
int setObj(XPRBctr ctr)
calls XPRBsetobj
int setObj(XPRBquadExp q)
int setObj(XPRBlinExp l)
int setObj(XPRBvar v)
void delQObj()
calls XPRBdelqobj
XPRBsos newSos(char type)
calls XPRBnewsos
XPRBsos newSos(char *name, char type)
```

calls XPRBnewsos
XPRBsos newSos(char type, XPRBlinExp& l)

calls XPRBnewsos
XPRBsos newSos(char *name, char type, XPRBlinExp& l)
calls XPRBnewsos
void delSos(XPRBsos& sos)
calls XPRBdelsos
XPRBindexSet newIndexSet()
calls XPRBnewidxset
XPRBindexSet newIndexSet(char *name)
calls XPRBnewidxset
XPRBindexSet newIndexSet(char *name, int maxsize)
calls XPRBnewidxset
XPRBbasis saveBasis()
calls XPRBsavebasis
int loadBasis(const XPRBbasis& b)
calls XPRBloadbasis
void clearDir()
calls XPRBcleardir
int setSense(int dir)
calls XPRBsetsense
int getSense()
calls XPRBgetsense
char *getName()
calls XPRBgetprobname
int exportProb(int format, char *filename)
calls XPRBexportprob
int loadMat()
calls XPRBloadmat
int print()
calls XPRBprintprob (only from C++)
int solve(char *alg)
calls XPRBsolve
int minim(char *alg)
calls XPRBminim
int maxim(char *alg)
calls XPRBmaxim
int getProbStat()
calls XPRBgetprobstat
int getLPStat()
calls XPRBgetlpstat
int getMIPStat()
calls XPRBgetmipstat
double getObjVal()

calls XPRBgetobjval
XPRBvar getVarByName(char *name)
calls XPRBgetbyname
XPRBctr getCtrByName(char *name)
calls XPRBgetbyname

`XPRBsos getSosByName(char *name)`
calls `XPRBgetbyname`
`XPRBindexSet getIndexSetByName(char *name)`
calls `XPRBgetbyname`

クラス XPRBvar

構成:

`XPRBvar()`
`XPRBvar(XPRBvar *v)`

方法:

`XPRBvar *Cref()` (C++ only)
`bool isValid()`
`int print()`
calls `XPRBprintvar` (only from C++)
`int setType(char type)`
calls `XPRBsetvartype`
`int setUB(double val)`
calls `XPRBsetub`
`int setLB(double val)`
calls `XPRBsetlb`
`int setLim(double val)`
calls `XPRBsetlim`
`int fix(double val)`
calls `XPRBfixvar`
`int setDir(int type, double val)`
calls `XPRBsetdir`
`int setDir(int type)`
calls `XPRBsetdir`
`char *getName()`
calls `XPRBgetvarname`
`int getColNum()`
calls `XPRBgetcolnum`
`char getType()`
calls `XPRBgetvartype`
`double getLB()`
calls `XPRBgetbounds`
`double getUB()`
calls `XPRBgetbounds`
`int getBounds(double *lb, double *ub)` (C++ only)
calls `XPRBgetbounds`
`int getLim(double *val)`
calls `XPRBgetlim`
`double getSol()`

calls `XPRBgetsol`
`double getRCost()`
calls `XPRBgetrcost`

クラス XPRBctr

構成:

`XPRBctr()`
`XPRBctr(XPRBctr *c)`
`XPRBctr(XPRBctr *c, XPRBlinRel& ctr)`

方法:

`XPRBctr *Cref()` (C++ only)
`bool isValid()`
`int print()`
calls `XPRBprintctr` (only from C++)
`int setType(char type)`
calls `XPRBsetctrtype`
`int setRange(double low, double up)`
calls `XPRBsetrange`
`int setModCut(bool mstat)`
calls `XPRBsetmodcut`
`bool isModCut()`
calls `XPRBgetmodcut`
`char *getName()`
calls `XPRBgetctrname`
`double getRangeL()`
calls `XPRBgetrange`
`double getRangeU()`
calls `XPRBgetrange`
`int getRange(double *lw, double *up)` (C++ only)
calls `XPRBgetrange`
`double getRHS()`
calls `XPRBgetrhs`
`int getRowNum()`
calls `XPRBgetrownum`
`char getType()`
calls `XPRBgetctrtype`
`double getSlack()`
calls `XPRBgetslack`
`double getDual()`
calls `XPRBgetdual`
`int setTerm(XPRBvar& var, double val)`
calls `XPRBsetterm`

`int setTerm(double val)`
calls `XPRBsetterm`
`int addTerm(XPRBvar& var, double val)`
calls `XPRBaddterm`
`int addTerm(double val)`
calls `XPRBaddterm`
`int delTerm(XPRBvar& var)`
calls `XPRBdelterm`
`void add(XPRBlinExp& l)`

C++ only: Operators

Assigning constraints and adding linear expressions:

ctr = linrel
ctr += linexp
ctr -= linexp
Xpress-BCL Reference Manual C++ and Java Class Reference 162

クラス XPRBsos

構成:

```
XPRBsos()
XPRBsos(XPRBsos *s)
XPRBsos(XPRBsos *s, XPRBlinExp& l)
```

方法:

`XPRBsos *Cref()` (C++ only)
`bool isValid()`
`int print()`
calls `XPRBprintsos` (only from C++)
`char *getName()`
calls `XPRBgetsosname`
`char getType()`
calls `XPRBgetsostype`
`int setDir(int type, double val)`
calls `XPRBsetsosdir`
`int setDir(int type)`
calls `XPRBsetsosdir`
`int addElement(XPRBvar& var, double val)`
calls `XPRBaddsosel`
`int delElement(XPRBvar& var)`
calls `XPRBdelsosel`

C++ only: operators

Assigning and adding linear expressions to Special Ordered Sets:
set = linexp
set += linexp
Xpress-BCL Reference Manual BCL in C++ and Java 163

クラス XPRBindexSet

構成:

```
XPRBindexSet()
XPRBindexSet(XPRBindexSet *i)
```

方法:

`XPRBindexSet *Cref()` (C++ only)
`bool isValid()`
`int print()`
calls `XPRBprintidxset` (only from C++)
`char *getName()`
calls `XPRBgetidxsetname`
`int getSize()`
calls `XPRBgetidxsetsize`
`int addElement(const char *text)`
calls `XPRBaddidxel`
`int getIndex(const char *text)`
calls `XPRBgetidxel`
`char *getIndexName(int i)`

calls `XPRBgetidxelname`

C++ only: Operators

Adding an element to an index set:
iset += text

Accessing index set elements by their name or index number:
int iset[text]
*char *iset[val]*

クラス XPRBbasis

構成:

```
XPRBbasis()
XPRBbasis(XPRBbasis *bs)
```

方法:

`XPRBvar *Cref()` (C++ only)
`bool isValid()`

B.4 追加クラス

この節では，標準のBCLのモデリングオブジェクトに対応しないクラスの一覧を示す．クラスXPRBは初期化や，ソフトウェアの一般的な状態，全てのパラメータの定義（ここに一覧されていない），に関係する方法を含む．このことは語頭がXPRB_であるBCLのパラメータはC++ やJavaではクラスXPRBの定数として参照されることを意味する．例えば，標準的なBCLで使われるXPRB_BVはC++ではXPRB_BVに，Javaでは，XPRB.BVになる．
これ以外のクラスは，制約式の項別の定義のために導入された．Javaでは，C++の場合の代数演算子による数式定義の代わりに，簡単な方法を使った方法が用いられる．１次式（クラス XPRBlinExp）は制約式やSOSの定義のために必要である．2次式(クラス XPRBquadExp)は2次の目的関数を定義するために必要である．線形関係(クラス XPRBlinRel)は制約式定義の途中で利用する．
前節と同様に，C++とJavaに共通である場合，リスト上ではC++での記法のみが与えられている（C++の型charまたはchar * はJavaではStringに置き換わる．また，型boolはboolean になる．さらに，Javaではポインタは存在しない）

クラス XPRBlinExp

構成:

```
XPRBlinExp(double d)
XPRBlinExp(int i)
XPRBlinExp(double d, XPRBvar& v)
XPRBlinExp(XPRBvar& v)
XPRBlinExp(XPRBlinExp& l)
```

方法:

```
XPRBlinExp& neg()
XPRBlinExp neg(XPRBlinExp& )
XPRBlinExp& add(XPRBlinExp& l)
XPRBlinExp& add(XPRBvar& v)
XPRBlinExp& add(double d)
XPRBlinExp add(XPRBlinExp& l1, XPRBlinExp& l2)
XPRBlinExp add(XPRBlinExp& l, XPRBvar& v)
XPRBlinExp add(XPRBvar& v, XPRBlinExp& l)
XPRBlinExp add(XPRBlinExp& l, double d)
XPRBlinExp add(double d, XPRBlinExp& l)
XPRBlinExp& mul(double d)
XPRBlinExp mul(XPRBlinExp& l, double d)
XPRBlinExp mul(double d, XPRBlinExp& l)
XPRBlinExp& assign(XPRBlinExp& l)
```

Java only

```
XPRBquadExp mul(XPRBvar& v)
XPRBquadExp sqr()
XPRBlinRel lEql(XPRBlinExp l)
XPRBlinRel lEql(XPRBvar v)
XPRBlinRel lEql(double d)
XPRBlinRel gEql(XPRBlinExp l)
XPRBlinRel gEql(XPRBvar v)
XPRBlinRel gEql(double d)
XPRBlinRel eql(XPRBlinExp l)
XPRBlinRel eql(XPRBvar v)
XPRBlinRel eql(double d)
```

C++ only: Operators

Assigning (elements to) linear expressions:
linexp1 += *linexp2*
linexp1 -= *linexp2*
linexp1 = *linexp2*
線形式(*linexp*)，変数(*var*)，（double型）値(*val*)から，線形式を作成する．以下の演算は（最後の符号の逆転をのぞき）クラス定義の外で定義される必要がある:
- *linexp*
linexp1 + *linexp2*
linexp1 - *linexp2*
linexp * *val*
val * *linexp*
var * *val*
val * *var*
- *var*

クラス XPRBquadExp によるクラスXPRBlinExpの拡張

構成:

```
XPRBquadExp()
XPRBquadExp(double d, XPRBvar& v1, XPRBvar& v2)
XPRBquadExp(XPRBvar& v)
XPRBquadExp(XPRBlinExp& l)
XPRBquadExp(XPRBquadExp& q)
```

方法:

```
XPRBquadExp& neg()
XPRBquadExp neg(XPRBquadExp q)
XPRBquadExp& add(XPRBquadExp& q)
XPRBquadExp& add(XPRBlinExp& l)
XPRBquadExp& add(XPRBvar& v)
XPRBquadExp& add(double d)
XPRBquadExp add(XPRBquadExp& q1, XPRBquadExp& q2)
XPRBquadExp& mul(double d)
XPRBquadExp mul(XPRBquadExp& q, double d)
XPRBquadExp mul(double d, XPRBquadExp& q)
XPRBquadExp& mul(XPRBquadExp& q)
```

throws ArithmeticException ‘Non-quadratic expression’ if the result of the operation is not quadratic

```
XPRBquadExp mul(XPRBquadExp& q1, XPRBquadExp& q2)
```

throws ArithmeticException ‘Non-quadratic expression’ if the result of the operation is not quadratic

```
XPRBquadExp& mul(XPRBlinExp& l)
```

throws ArithmeticException ‘Non-quadratic expression’ if the result of the operation is not quadratic

```
XPRBquadExp mul(XPRBquadExp& q, XPRBlinExp& l)
```

throws ArithmeticException ‘Non-quadratic expression’ if the result of the operation is not quadratic

```
XPRBquadExp mul(XPRBlinExp& l, XPRBquadExp& q)
```

throws ArithmeticException ‘Non-quadratic expression’ if the result of the operation is not quadratic

```
XPRBquadExp mul(XPRBlinExp& l1, XPRBlinExp& l2)
```

throws ArithmeticException ‘Non-quadratic expression’ if the result of the

operation is not quadratic
`XPRBquadExp& assign(XPRBquadExp& q)`
Xpress-BCL Reference Manual Additional Classes 168

C++ only: Operators

Assigning (elements to) quadratic expressions:
qexp1 += qexp2
qexp1 -= qexp2
qexp1 = qexp2
2次式(*qexp*)，変数(*var*)，（double型）値(*val*)から，2次式を作成する．以下の演算は（最後の符号の逆転をのぞき）クラス定義の外で定義される必要がある:
- qexp
qexp1 + qexp2
qexp1 - qexp2
*var1 * var2*
*linexp * var*
*var * linexp*
*qexp * val*
*val * qexp*
*qexp1 * qexp2*
throws the exception 'Non-quadratic expression' if the result of the operation is not quadratic
Functions outside any class definition that generate quadratic expressions:
`XPRBquadExp sqr(XPRBquadExp& Q)`
`XPRBquadExp sqr(XPRBlinExp& l)`
`XPRBquadExp sqr(XPRBvar& v)`
Xpress-BCL Reference Manual BCL in C++ and Java 169

クラス XPRBlinRel によるクラス XPRBlinExpの拡張

構成

```
XPRBlinRel(const XPRBlinExp& l, char t)
XPRBlinRel(const XPRBlinExp& l)
XPRBlinRel(const XPRBvar& v)
```

C++ only: Operators

線形式の関係を作成する．以下の演算はクラス定義の外で定義する:
linexp1 <= linexp2
linexp1 >= linexp2
linexp1 == linexp2

クラス XPRB

方法:

`int init()`
calls `XPRBinit`
`int setMsgLevel(int lev)`
calls `XPRBsetmsglevel`
`char *getVersion()`
calls `XPRBgetversion`
Xpress-BCL Reference Manual Additional Classes 170
Xpress-BCL Reference Manual Index 171

索引

記号

- 166, 168
* 163, 166, 168
+ 166, 168
+= 161, 162, 163, 166, 168
<= 169
-= 161, 166, 168
= 161, 162, 166, 168
== 169
>= 169

A

add（追加）
index element（インデックス要素） 24
add 161, 165, 167
addElement 162, 163
addTerm 161
array（配列） 137
add entry（要素の追加） 116
create（生成） 9
delete（削除） 35
incremental definition（逐次定義） 9
name（名前） 10, 47
print（印刷） 17
size（サイズ） 10, 48
terminate（終了） 43
assign 165, 167

B

basis（基底）
class（クラス） 154
delete（削除） 14, 36
load（読み込み） 14, 82
save（保存） 14, 115
bound（限定）
fix（固定） 8, 45
get（獲得） 8, 49
integer limit（整数限界） 8, 59, 123
lower（下側） 8, 122
semi-continuous limit（半連続限界） 8, 59, 123
upper（上側） 8, 133
branching directive（分枝方向）
SOS 17, 130
variable（変数） 8, 119

C

callback（コールバック）
error messages（エラーメッセージ） 7, 31

messages（メッセージ） 6, 33
printing（印刷） 6, 33
change（変更）
constraint type（制約式の型） 11
variable type（変数の型） 8, 134
clearDir 156
constraint（制約式） 11
add array（配列の追加） 10, 23
add term（項の追加） 10, 23, 28
bound（限定） 133
change type（型の変更） 11, 117
class（クラス） 154
creation（生成） 10, 92
definition（定義） 10, 92
delete（削除） 11, 37
delete coefficient（係数の削除） 11
delete term（項の削除） 42
dual（双対） 14, 54
get range（範囲の獲得） 11, 66
get RHS（右辺の獲得） 68
get type（型の獲得） 11, 53
index（インデックス） 11, 69
model cut（モデルカット） 12, 62
name（名前）11, 52
precedence（優先度） 95
print（プリント） 17, 105
set coefficient（係数の設定） 10

set model cut（モデルカットの設定） 12, 124
set range（範囲の設定） 11, 128
set term（項の設定） 132
sum（総和） 101
sum with coefficients（係数による総和） 88
constraint set type（制約式集合の型） 117
create（生成）
index set（インデックス集合） 93
Cref 155, 158, 160, 162, 163, 164
cut（カット）
model（モデル） 12, 62, 124

D

data（データ）
input（入力） 15, 113
reading（読み込み） 113
decimal sign（小数点記号）
change（変更） 7, 118
delCtr 155
delElement 162

delete（削除）
array（配列） 35
constraint（制約式） 11, 37
constraint coefficient（制約式の係数） 11
constraint term（制約式の項） 42
problem（問題） 38
quadratic term（2次項） 39
set element（集合の要素） 17
set member（集合のメンバー） 41
SOS 17, 40
delQObj 155
delSos 156
delTerm 161
directive（方向）
delete（削除） 30
SOS 17, 130
variable（変数） 8, 119
dual values（双対値） 14, 54
E
eql 165
error（エラー）
exit（終了） 7, 121
error callback（エラーコールバック） 7, 31
error handling（エラー処理） 7, 121
error message（エラーメッセージ） 7, 31
exportProb 156
expression（式）
linear（線形） 164
quadratic（2次） 164
F
file（ファイル）
reading（読み込み） 15
find by name（名前による検索） 50
fix 158
G
generate matrix（行列の生成） 84
gEql 165
get RHS（右辺項の獲得） 11
getBounds 158
getColNum 158
getCtrByName 154, 157
getDual 160
getIndex 163
getIndexName 163
getIndexSetByName 157
getLB 158
getLim 158

getLPStat 156
getMIPStat 156
getName 156, 158, 160, 162, 163
getObjVal 156
getProbStat 156
getRange 160
getRangeL 160
getRangeU 160
getRCost 159
getRHS 160
getRowNum 160
getSense 156
Xpress-BCL Reference Manual Index 173
getSize 163
getSlack 160
getSol 158
getSosByName 157
getType 158, 160, 162
getUB 158
getVarByName 154, 157
getVersion 169

I

incremental definition（逐次的定義）
array（配列） 9
index（インデックス）
constraint（制約式） 11, 69
variable（変数） 8, 51
index set（インデックス集合） 15
add element（要素の追加） 15, 24
class（クラス） 154
create（生成） 15, 93
element name（要素の名前） 15, 56
find element（ファイルの要素） 15
index number（インデックス番号） 15
name（名前） 15, 57
print（印刷） 17, 107
size（サイズ） 15, 58
init 169
initialization（初期化） 81, 96, 141
input（入力）
decimal sign（小数点記号） 7, 118
file（ファイル） 113
input file（入力ファイル） 15
isModCut 160
isValid 154, 158, 160, 162, 163, 164

L

lEql 165
licence（ライセンス） 5, 81

license（ライセンス） 96, 141
linear expression（線形式） 164
class（クラス） 154
linear relation（線形関係） 164
class（クラス） 154
load matrix（行列読み込み） 6
loadBasis 156
loadMat 156

M

matrix（マトリックス）
generation（生成） 84
loading（読み込み） 6
output（出力） 17, 44
maxim 156
maximize（最大化） 14, 85, 135
message level（メッセージレベル） 6, 125
minim 156
minimize（最小化） 14, 87, 135
MIPPRESOLVE 147
model cut（モデルカット） 12, 62, 124
modeling object（モデリングオブジェクト）
finding（検索） 50
mul 165, 167

N

name（名前）
array（配列） 10, 47
composing of（組み立て） 94
constraint（制約式） 11, 52
index set（インデックス集合） 15, 57
index set element（インデックス集合要素） 15, 56
problem（問題） 64
SOS 17, 74
variable（変数） 8, 77
namespace（名前空間） 151
neg 165, 167
newCtr 155
newIndexSet 156
newSos 155, 156
newVar 155

O

objective（目的）

add quadratic term（2次項の追加） 25
delete quadratic terms（2次項の削除） 39
get sense（方向を獲得） 12, 70
quadratic（2次） 18
set quadratic term（2次項の設定） 127

set sense（方向を設定） 12, 129
value（値） 14, 63
objective function（目的関数） 126
optimize（最適化） 14, 135
output（出力）
file（ファイル） 17, 44
output level（出力レベル） 6, 125

P

package（パッケージ） 153
partial integer（部分整数）
get limit（限界の獲得） 8, 59
set limit（限界の設定） 8, 123
precedence constraint（優先度制約） 12, 95
PRESOLVE 147
print（印刷）
array（配列） 17, 104
constraint（制約式） 17, 105
index set（インデックス集合） 17, 107
problem（問題） 17, 108
program output（問題の出力） 106
SOS 17, 109
text（テキスト） 106
variable（変数） 17, 110
print 156, 158, 160, 162, 163
print flag（プリントフラグ） 6, 125
printing（出力する）
decimal sign（小数点） 7, 118
printing callback（コールバック印刷） 6, 33
priority（優先度）
delete（削除） 30
SOS 17, 130
variable（変数） 8, 119
problem（問題）
class（クラス） 154
delete（削除） 38
delete basis（基底削除） 14, 36
dual values（双対値） 14, 54
file output（ファイル出力） 17, 44
initialization（初期化） 96
load basis（基底読み込み） 14, 82
LP status（LPの状態） 14, 60
maximize（最大化） 85
minimize（最小化） 87
MIP status（MIPの状態） 14, 61
Name（名前） 64
objective value（目的関数値） 14, 63
output（出力） 108

print（印刷） 17, 108
reduced cost（被約費用） 14, 67
save basis（基底の保存） 14, 115
slack values（スラック値） 14, 71
solution（解） 72
solve（解く） 14, 135
status（状態） 14, 65
program（プログラム）
version number（バージョン番号） 7, 79
program output（プログラム出力）
print（プリント） 17, 106

Q

QP（2次計画） 18
quadratic expression（2次式） 164
class（クラス） 154
Quadratic Programming（2次計画問題） 18
quadratic term（2次項） 25, 127
delete（削除） 39

R

range（範囲） 11, 66, 128
get values（値獲得） 11
read（読み込み）
data line（データ行） 113
reduced cost value（被約費用値） 14, 67
reference constraint（参照制約） 17, 98
relation（関係）
linear（線形） 164

RHS（右辺） 11, 68
running time（稼働時間） 76

S

saveBasis 156
security system（セキュリティシステム） 5
semi-continuous（半連続）
get limit（限界値の獲得） 8, 59
set limit（限界値の設定） 8, 123
semi-continuous integer（半連続整数）
get limit（限界値の獲得） 8, 59
set limit（限界値の設定） 8, 123
sense（方向）
objective function（目的関数） 12, 70, 129
set（設定）
index（インデックス） 15
setDir 158, 162
setLB 158
setLim 158
setModCut 160

setMsgLevel 169
setObj 155
setRange 160
setSense 156
setTerm 160, 161
setType 158, 160
setUB 158
size（サイズ）
array（配列） 10, 48
index set（インデックス集合） 15, 58
size limit（サイズの限界） 142
slack values（スラック値） 14, 71
solution（解） 14, 72
objective（目的） 14, 63
solve（解く） 14, 85, 87, 135
solve 156
SOS
add array（配列の追加） 16, 26
add member（メンバーの追加） 26, 27
class（クラス） 154
creation（作成） 16, 97, 98, 100
delete（削除） 17, 40
delete element（要素の削除） 17
delete member（メンバーの削除） 41
name（名前） 17, 74
print（プリント） 17, 109
set directive（方向の設定） 17, 130
type（型） 17, 75
sos
add element（要素の追加） 16
sqr 165, 168
status（状態）
LP 14, 60
MIP 14, 61
Problem（問題） 14, 65
student version（学生版） 19, 44, 81, 96, 105, 107, 108, 109, 141
sum constraint（総和制約） 12, 101

T

table（票）
sparse（疎） 15
time(時間) 76
type(型)
constraint(制約式) 11, 53, 117
SOS 17, 75
variable（変数） 8, 78, 134

V

variable（変数）
array of（配列） 9, 90
change type（型の変更） 8, 134
class（クラス） 154
creation（作成） 8, 102
fix value（値の固定） 8, 45
get bounds（限定の獲得） 8, 49
get limit(限界の獲得) 8, 59
get type(型の獲得) 8, 78
index（インデックス） 8, 51
lower bound（下限） 8, 122
name（名前） 8, 77
number(数) 51
Xpress-BCL Reference Manual Index 176
print(印刷) 17, 110
print array（配列の印刷） 104
reduced cost（被約費用） 14, 67
set directive（方向の設定） 8, 119
set limit（限界の設定） 8, 123
set type（型の設定） 134
slack（スラック） 14, 71
solution（解） 14, 72
upper bound（上界） 8, 133
variable types(変数の型) 1
version number（バージョン番号） 7, 79

X

XPRB 164, 169
XPRB_BV 164
XPRBaddarrterm 23
XPRBaddidxel 24, 163
XPRBaddqterm 19, 25
XPRBaddsosarrel 26
XPRBaddsosel 27, 162
XPRBaddterm 28, 161
XPRBapparrvarel 29
XPRBarrvar 21, 139
XPRBbasis 21, 164
XPRBcleardir 30, 156
XPRBctr 21, 139, 160
XPRBdefcberr 31
XPRBdefcbmsg 33
XPRBdelarrvar 35
XPRBdelbasis 36
XPRBdelctr 37, 39, 155
XPRBdelprob 38
XPRBdelqobj 19, 39, 155
XPRBdelsos 40, 156
XPRBdelsosel 41, 162

XPRBdelterm 42, 161
XPRBendarrvar 43, 139, 141
XPRBexportprob 19, 44, 142, 156
XPRBfixvar 45, 158
XPRBgetarrvarname 47
XPRBgetarrvarsize 48
XPRBgetbounds 49, 158
XPRBgetbyname 50, 157
XPRBgetcolnum 51, 147, 158
XPRBgetctrname 52, 160
XPRBgetctrtype 53, 160
XPRBgetdual 54, 160
XPRBgetidxel 55, 163
XPRBgetidxelname 56, 163
XPRBgetidxsetname 57, 163
XPRBgetidxsetsize 58, 163
XPRBgetlim 59, 158
XPRBgetlpstat 60, 156
XPRBgetmipstat 61, 156
XPRBgetmodcut 62, 160
XPRBgetobjval 63, 157
XPRBgetprobname 64, 156
XPRBgetprobstat 65, 156
XPRBgetrange 66, 160
XPRBgetrcost 67, 159
XPRBgetrhs 68, 160
XPRBgetrownum 69, 147, 160
XPRBgetsense 70, 156
XPRBgetslack 71, 160
XPRBgetsol 72, 159
XPRBgetsosname 74, 162
XPRBgetsostype 75, 162
XPRBgettime 76
XPRBgetvarname 77, 158
XPRBgetvartype 78, 158
XPRBgetversion 79, 169
XPRBidxset 21, 141
XPRBindexSet 163
XPRBinit 81, 145, 169
XPRBlinExp 164, 165, 167, 169
XPRBlinRel 164, 169
XPRBloadbasis 82, 156
XPRBloadmat 84, 145, 156
XPRBmaxim 85, 156
XPRBminim 87, 156
XPRBnewarrsum 88, 139
XPRBnewarrvar 90, 139
XPRBnewctr 92, 139, 155

XPRBnewidxset 93, 141, 156
XPRBnewname 94
XPRBnewprec 95, 139
XPRBnewprob 81, 96, 155
XPRBnewsos 97, 139, 155, 156
XPRBnewsosrc 98, 139
XPRBnewsosw 100, 139
XPRBnewsum 101, 139
XPRBnewvar 102, 139, 155
XPRBprintarrvar 104
XPRBprintctr 19, 105, 160
XPRBprintf 106
XPRBprintidxset 107, 163
XPRBprintprob 19, 108, 156
XPRBprintsos 109, 162
XPRBprintvar 110, 158
XPRBprob 21, 155
XPRBquadExp 164, 167
XPRBreadarrline 111
XPRBreadline 113
XPRBsavebasis 115, 156
XPRBsetarrvarel 116
XPRBsetctrtype 160
XPRBsetdecsign 118
XPRBsetdir 158
XPRBseterrctrl 121
XPRBsetlb 122, 158
XPRBsetlim 123, 158
XPRBsetmodcut 124, 160
XPRBsetmsglevel 125, 169
XPRBsetobj 39, 126, 155
XPRBsetqterm 19, 127
XPRBsetrange 128, 160
XPRBsetsense 129, 156
XPRBsetsosdir 130, 162
XPRBsetterm 132, 160, 161
XPRBsetub 133, 158
XPRBsetvardir 119
XPRBsetvartype 134, 158
XPRBsolve 135, 140, 156
XPRBsos 21, 139, 140, 162
XPRBstartarrvar 137, 139
XPRBvar 21, 139, 158
XPRSalter 144
XPRSbtran 144
XPRSftran 144
XPRSgetcolrange 144
XPRSgetintattrib 144
XPRSgetintcontrol 144

XPRSgetrowrange 144
XPRSgetsol 144, 147
XPRSglobal 144
XPRSiis 144
XPRSinit 145
XPRSloadbasis 144
XPRSloaddirs 144
XPRSloadglobal 144
XPRSloadlp 144
XPRSloadqp 144
XPRSloadsecurevecs 144
XPRSmaxim 144
XPRSminim 144
XPRSrange 144
XPRSreadbasis 144
XPRSreaddirs 144
XPRSreadprob 144
XPRSrestore 144
XPRSsave 144
XPRSscale 144
XPRSsetintcontrol 144
XPRSsetprobname 144
XPRSwritebasis 144
XPRSwriteomni 144
XPRSwriteprob 144
XPRSwriteprtrange 144
XPRSwriteprtsol 144
XPRSwriterange 144
XPRSwritesol 144
Xpress-BCL Reference Manual Index 178