

1 概 要

1.1 Bolyaiとは

Bolyai（ボヤイ）は、サーバーを使用しない簡易型のリレーショナルデータベースソフトです。表形式のデータの操作を主体とするアプリケーション（ほとんどのビジネスアプリケーションがこれに当てはまります。）を迅速に開発し低コストで運用するためのオールインワン環境を提供します。

Bolyaiのプログラムは、サーバー上ではなくクライアント上で動作します。プログラムファイルとデータファイルをネットワーク上の共有フォルダに置いておき、必要な時にクライアント側からこのプログラムを起動してデータを操作するという使い方をします。（クライアントPCにインストールしてスタンドアロンで使用することも当然可能です。また、USBメモリに入れておいてそこから起動するという使い方もできます。）

アプリケーションは、C言語で記述します。データベースへのアクセスはC言語から専用のAPI関数を呼ぶことで行います。

データファイルは、表ごとに固定長のレコードを順編成しただけのもの（C言語の構造体の配列のメモリ内イメージをそのまま書き出したもの）です。表データの操作は、常にファイル全体をクライアントの主記憶にロードして行います。

排他制御は、ファイル単位でのオブティミスティック・ロックで行ないます。（ペシミスティック・ロックでの運用も可能です。）

1.2 配布メディアの内容

配布メディアには、以下のファイルとフォルダが含まれます。

README.TXT	簡単な説明書
BIN フォルダ	Bolyai本体とその動作に必要なファイル
BOLYAI.EXE	プログラムファイル
BOLYAI.BSA	初期データ（ユーザー管理表）
BOLYAI.PDF	このマニュアル
BOLYAI_C.H	スクリプト言語用のヘッダファイル
SAMPLE フォルダ	スクリプト言語のサンプルソースコード
SDK フォルダ	Bolyai本体のソースコード

1.3 Bolyaiの利用方法

Bolyai上でアプリケーションを開発するには次の2つの方法があり、それぞれ開発の難易度と実現可能な機能の範囲が異なります。

(1) Bolyaiがサポートするスクリプト言語（C言語のサブセット）を使用したラピッド開発

- ・開発者はC言語を知っている必要があります。
- ・Bolyai単体で開発と運用が可能です。
- ・データベース定義、データ操作、帳票印刷のコールバック関数を作成することで、最小限の工数で実際に動作するアプリケーションを構築できます。
- ・ユーザーインタフェース部品のカスタマイズはできません。

(2) Bolyai本体のソースコード（C言語）をフレームワークとして使用したネイティブ開発

- ・開発者はC言語とWin32 ネイティブAPIを知っている必要があります。
- ・別途Cコンパイラが必要です。
- ・データベースエンジン、言語処理系、ユーザーインタフェース部の全ソースコードが利用可能です。
- ・ユーザーインタフェース部品のカスタマイズを含め何でもできます。

(1)(2)ともに共通のAPIを使用した同じC言語上での開発であるため、(1)でプロトタイプを作成して動作や仕様を確認してから(2)に移行するといった使い方もできます。

1.4 Bolyaiの特性（長所と短所）

1.4.1 Bolyaiの長所

Bolyaiの長所としては、次のような点が挙げられます。

- (1) データ／プログラム共に極めて軽量で高速に動作すること
- (2) サーバーを構築する必要がなく、またクライアントごとにソフトをインストールする必要もないため、導入と維持が容易であること
- (3) データファイルのフォーマットがC言語の構造体の配列そのものであるため、C言語との親和性が高くプログラミングしやすいこと
- (4) データベース操作とフロントエンド処理に別々の言語を使い分ける必要がないこと

1.4.2 Bolyaiの短所

一方短所としては、次のような点が挙げられます。

- (1) クライアントの主記憶に入る大きさの表までしか扱えないこと
- (2) データファイルとプログラムファイルをユーザーの目に見えるフォルダに置くため、機密性と安全性に難があること
- (3) アプリケーション機能とデータベース機能が分離されていないため、データの整合性に関しては全面的にアプリケーション側の責任となること
- (4) ネットワーク上のファイル共有の不安定さに起因するトラブル（ファイル破損等）が発生する可能性があること

1.4.3 Bolyaiの流儀

一般的なリレーショナルデータベースと比較したとき、次のような点が独自仕様と言えます。

- (1) 表の定義情報にデータの物理的なレイアウト情報が含まれていること
 - ・メモリ内でのデータ項目の物理的な位置が分かるため、このことを利用したプログラミング（メモリ内にロードした表データを構造体の配列とみなしてアクセスすること）が可能です。
- (2) 表の定義情報にデータの画面上での見た目情報（列幅、文字揃え等）を含めることができること
 - ・表定義を与えるだけで、すぐに印刷可能な表ウィンドウを画面上に表示できます。
- (3) 実表とビューの統一的な扱い
 - ・表定義には、実表項目の定義の他に、実表項目から計算で得られる導出項目（計算項目）の定義を含めることができます。ビュー上で結合して表示したい他の表の項目（外部参照項目）もあらかじめ計算項目として与えます。
- (4) 非ヌル制約、一意性制約、主キー制約、外部キー制約等の扱い
 - ・NULL値の概念がないため、「値が0または空文字列である」とことと「値が設定されていない」状態とを区別できません。非ヌル制約にかえて非ゼロ制約を用いることができます。
 - ・主キーを複合キーにすることは可能ですが、外部キーは複合キーにできません。（主キーが複合キーである表は、他の表から外部キー参照できません。）
 - ・一意性制約は、単一の列項目にしか指定できません。主キーを複合キーとする場合を除き、複数の列の組み合わせに対して一意性を指定する方法はありません。
 - ・単一の列項目に主キー制約と外部キー制約を両方指定することは可能ですが、複数の参照先に対する外部キー制約を重ねて指定することはできません。
 - ・外部キーによる自己参照はできません。
 - ・外部キー制約があっても、参照先の親表の操作について何らかの制限が加わる（参照されている間は削除できない等）ことはありません。また、参照先の親表の行が削除されたとき参照元の子表の行が自動的に削除される（カスケード削除）機能もありません。

1.5 動作環境

Bolyaiは、以下のOS上で動作します。

Windows 11/Windows 10/Windows 8(8.1)/Windows 7/Windows Vista/Windows XP

配布メディアのBINフォルダ内のファイル以外にランタイムライブラリ等は一切必要ありません。

スタンドアロン利用の場合、対象PCにプログラムとデータをインストールして使用します。ネットワーク利用の場合、共有フォルダにプログラムとデータをインストールし、クライアントPC側からネットワーク越しにプログラムを起動して使用します。

クライアントPC上には、ユーザー設定を保存するためのフォルダ（デフォルトでは、C:\NFFILES）が自動的に作成されます。レジストリは使用しません。

ユーザーは、Bolyaiがインストールされたフォルダに対し、読み取り＋書き込み＋プログラム実行の権限を持っている必要があります。

2 セットアップ

2.1 インストール

本ソフトにはインストーラはありません。任意のフォルダに配布メディアの **BIN** フォルダの内容をそのままコピーすればインストール完了です。

プログラムとデータを同じフォルダ内に置いて使用するため、プログラムからのアクセスがOSによってブロックまたはリダイレクションされるような場所にインストールすると、プログラムが正しく動作しません。

2.1.1 スタンドアロン利用の場合のインストール手順

対象PCのハードディスクに適当な名前の新規フォルダを作成し、そこに配布メディアの **BIN** フォルダ内の全ファイルをコピーします。Windows Vista以降機の場合、**ProgramFiles(x86)** 等のUAC保護下のフォルダにはインストールしないでください。Windows 10以降機の場合、アクセス制御機能によって保護されているフォルダにアクセスするには、許可アプリとして登録する必要があります。

コピーした **BOLYAI.EXE** へのショートカットをデスクトップに置いて（または、スタートメニューかタスクバーに登録して）ください。

2.1.2 ネットワーク利用の場合のインストール手順

ネットワーク内の共有ストレージに適当な名前の新規フォルダを作成し、そこに配布メディアの **BIN** フォルダ内の全ファイルをコピーします。

クライアントPC側では、コピーした **BOLYAI.EXE** へのショートカットをデスクトップに置いて（または、スタートメニューかタスクバーに登録して）ください。

2.2 ユーザー登録

本ソフトは、起動時にユーザーのパスワード認証を行います。ユーザー登録は、本ソフトをインストールしたフォルダ（以下「サイト」と呼びます。）単位で行います。ユーザー登録ができるのは、「サイト管理者」権限を持つユーザーだけです。初期データには、ユーザー名「**Admin**」パスワード「**password**」という仮のサイト管理者が登録されています。初回起動時はこの仮ユーザー名でログインし、最初にユーザー登録を行ってください。

本ソフトのユーザーには、次表に示す5段階の権限レベルがあります。

権限レベル	閲覧	編集	DB定義	ユーザー管理
(0) 権限なし	×	×	×	×
(1) 閲覧ユーザー	○	×	×	×
(2) 編集ユーザー	○	○	×	×
(3) DB管理者	○	○	○	×
(4) サイト管理者	○	○	○	○

少なくとも1名のサイト管理者の登録が必須です。他は必要に応じて登録してください。

ユーザー登録は、「処理(P)」 「ユーザー管理(U)」 コマンドでユーザー管理表を開き、各行にユーザー名、パスワード、権限レベルを入力することで行います。

2.3 環境設定

サイトには、プログラムから共通して参照するためのデータとして、以下のサイト変数を設定しておくことができます。

(1) サイト名（16byte以内の英字と数字の文字列）

- ・ユーザー管理表とログイン中ユーザーリストのファイル名として使用されます。
- ・プログラムからサイトを識別する必要がある場合に用いることができます。
- ・API関数 **SiteName()** で参照できます。

(2) サイトのタイトル（32byte以内の任意文字列）

(3) サイトの説明（64byte以内の任意文字列）

(4) 消費税率（実数値）

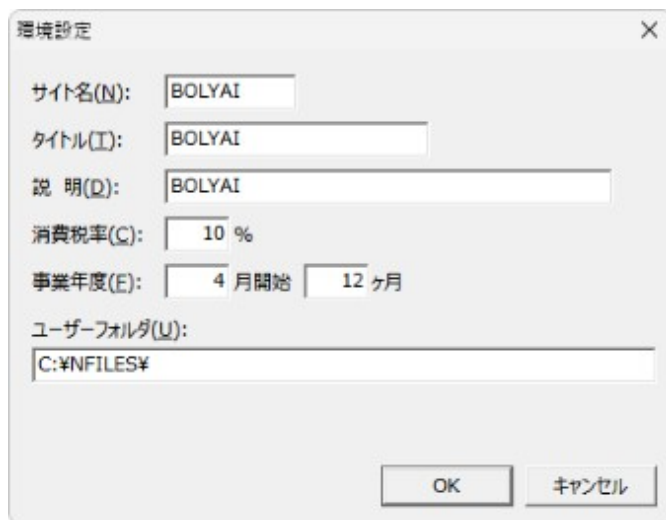
(5) 事業年度の開始月（整数値）と月数（整数値）

- ・これらには決まった用途はありません。アプリケーションで任意に使用できます。
- ・それぞれAPI関数 **SiteTitle()**、**SiteDescription()**、**TaxRate()**、**FYearMonth()**、**NMonthFYear()** で参照できます。

(6) ユーザーフォルダ名

- ・ユーザーごとの設定情報を保存するためにクライアントPC上に作成されるフォルダの位置を示します。
- ・API関数 **UserPath()** で参照できます。

環境設定ダイアログボックスは、「ツール(T)」 「環境設定(E)」 コマンドで開きます。



環境設定

サイト名(N): BOLYAI

タイトル(T): BOLYAI

説明(D): BOLYAI

消費税率(C): 10 %

事業年度(E): 4 月開始 12 ヶ月

ユーザーフォルダ(U): C:¥NFILES¥

OK キャンセル

ユーザーフォルダの位置を個別に変えたいときは、次の例のようにフォルダのパス名をプログラムのコマンドライン引数として与えてください。

BOLYAI.EXE C:¥User1¥

この指定は、環境設定よりも優先されます。

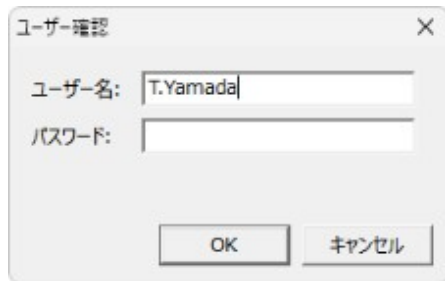
2.4 セキュリティに関する注意

本ソフトは、データファイルとプログラムファイルをユーザーの目に見えるフォルダに置いて使用するため、ソフト側のユーザー認証機能のみでは機密性と安全性を確保できません。必ずP C本体のユーザー認証機能を併用してください。ネットワーク利用の場合、インストール先の共有フォルダに適切なアクセス権設定をしておく必要があります。

3 基本操作

3.1 起動とログイン

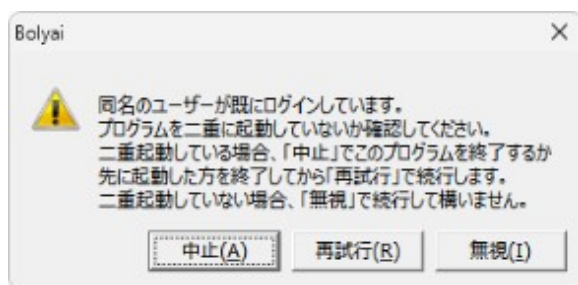
本ソフトを起動すると、最初に次のようなユーザー確認のダイアログボックスが現れます。



ユーザー名の入力欄には、そのクライアント上での最後のログインユーザー名が自動的に表示されます。（初回起動時は、空欄です。）

ダイアログボックスに正しい登録ユーザー名とパスワードを入力すると、サイトにログインすることができ、本ソフトを使えるようになります。（本ソフトは、ユーザー名とパスワードの確認時に大文字と小文字を区別しません。）

1つのサイトに対し、同じユーザー名で二重にログインすることはできません。1台のクライアント上で本ソフトを二重に起動したり、異なるクライアント上で同じユーザー名を使って同時にログインしようすると警告が出ます。



プログラムの異常終了等により、前回のログイン情報が残ったままになっている場合も、上記の警告が出ますが、その場合は「無視 (I)」で続行してください。

「ツール(T)」「オプション設定(O)」コマンドで「自動的にログインする(A)」オプションを選択すると、ユーザー確認画面をスキップすることができます。

3.2 ツールバー

ツールバー上には、現在のウィンドウ名（ウィンドウを開いていないときは、計算上の現在日付）が表示され、ウィンドウの種類に応じて以下のようなコマンドのアイコンが配置されます。



3.3 メニューバー

メニューバーには、以下のコマンドがあります。

ファイル(F)

メニュー(M)	3.4参照
新規作成(N)	3.5参照
開く(O)	3.6参照
閉じる(C)	3.7.16参照
上書保存(S)	3.7.17参照
名前を付けて保存(A)	3.9.2参照
テキスト取込(I)	3.7.18参照
テキスト書出(E)	3.7.19参照
印刷(P)	3.7.20参照
単票印刷(L)	3.7.21参照
終了(X)	

編集(E)

切り取り(T)	3.7.2参照
コピー(C)	3.7.2参照
貼り付け(P)	3.7.2参照
クリア(A)	3.7.2参照
挿入(I)	3.7.3参照
削除(D)	3.7.3参照
リンクを開く(O)	
検索(F)	3.7.4参照
次を検索(N)	3.7.4参照

表示(V)

表示形式(V)	3.7.8参照
表示項目(A)	3.7.5参照
絞り込み(F)	3.7.6参照
並べ替え(S)	3.7.7参照
全項目表示(W)	3.7.9参照
表示形式解除(B)	3.7.10参照
表示オプション(O)	3.7.11参照

データ(D)

集計(S)	3.7.12参照
チェック(C)	3.7.13参照
行編集(E)	3.7.14参照
行追加(A)	3.7.3参照
行挿入(I)	3.7.3参照
行削除(D)	3.7.3参照
物理的に並べ替え(P)	3.7.15参照

処理(P)

一括処理(X)	3.8.1参照
帳票作成(R)	3.8.2参照
DB接続状況(D)	3.8.3参照
ログイン状況(S)	3.8.3参照
ユーザー管理(U)	2.2参照
ロック解除(L)	3.8.4参照
ファイル転送(B)	3.8.5参照

ツール(T)

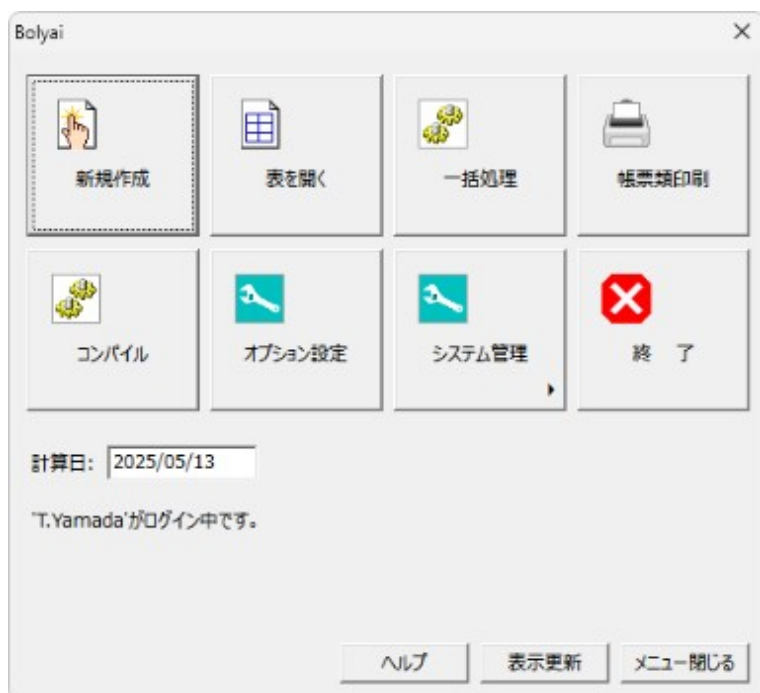
コンパイル(C)	3.9.1参照
テキスト編集(T)	3.9.2参照
環境設定(E)	2.3参照
オプション設定(O)	3.9.3参照
計算日設定(D)	3.9.4参照

ウィンドウ(W)

ヘルプ(H)

3.4 開始メニュー

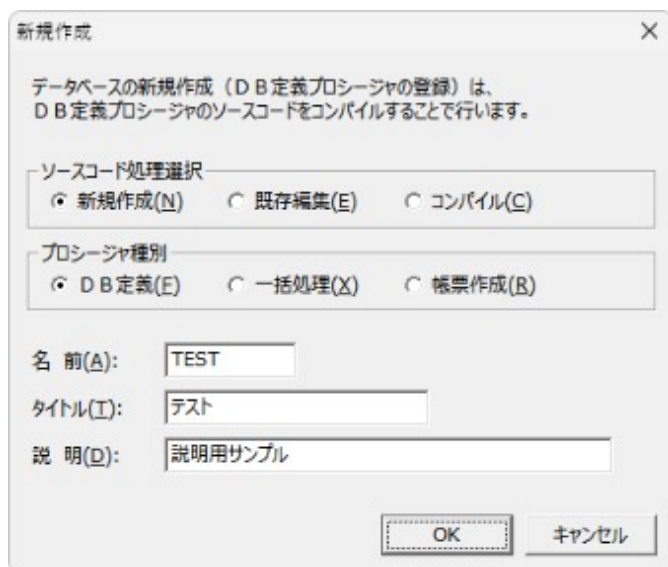
全てのコマンドは、メニューバーから実行できますが、主要なコマンドは、次のようなダイアログボックス形式の開始メニューを呼び出してそこから実行することもできます。



開始メニューは、メニューバーの「ファイル(F)」 「メニュー(M)」で呼び出します。
オプション設定で「開始メニューを使用する(M)」を選択すると、表ウィンドウが全て閉じられたタイミングで自動的に開始メニューが呼び出されます。
開始メニューの下部領域には、現在サイトにログイン中のユーザー名が表示されます。
「ヘルプ」ボタンを押すと、マニュアルが表示されます。
「表示更新」ボタンを押すと、サイトにログイン中のユーザー名の表示が更新されます。

3.5 データベースの新規作成

データベースの新規作成は、DB定義プロシージャのソースコードを用意してこれをコンパイルすることで行います。
[Ctrl]+[N]キーまたはツールバーの「新規作成」ボタンを押すと、「新規作成」ダイアログボックスが現れます。



ソースコードファイルがまだないときは、「ソースコード処理選択」を「新規作成(N)」にします。

「名前」にはプロシージャのバイトコードのファイル名（拡張子は含めない）となる16byte以内の英字と数字の文字列を入力します。「名前」として入力できる文字列は、C言語の識別子と同じ形式（最初の文字は英字で後続文字は英字か数字、下線文字は英字に属する、大文字と小文字は区別する）の文字列に限ります。（この制限は、列名、表名、DB名、サイト名に共通です。）なお、ファイル名として使用される「名前」は、大文字と小文字の区別なしでサイト内で一意でなければなりません。「タイトル」と「説明」は、任意の文字列です。

名前を付けて「OK」ボタンを押すと、テキスト編集画面になり、テンプレートとなる次のようなソースコードが呼び出されます。

```
// info  nam="TEST"  tit="テスト"  dsc="説明用サンプル"  pro=1

COLI ciTMP[1] = {
    { STR(64), 0, "cnam", "列タイトル", "列説明", CF_KEY, },
};
COLS csTMP = { ciTMP, NELEM(ciTMP) };

TBLI tiTMP[1] = {
    { &csTMP, NULL, "TNAM", "表タイトル", "表説明", },
};
TBLS tsTMP = { tiTMP, NELEM(tiTMP) };

DBBI dbTMP = { &tsTMP, "TEST", "テスト", "説明用サンプル" };

BOOL ConnectMain(void)
{ // サンプル
    int re;

    re = ConnectDBX(&dbTMP);
    if (re) NewDBFiles();
    return re;
}
```

テンプレートは、このままでもコンパイル可能な最小限のデータベース定義を記述しています。1行目のコメントは、プログラムの動作には影響しませんが、この書式で記述しておく、と、「コンパイル」ダイアログボックスを開いたとき、プロシージャの種別、名前、タイトル、説明が自動的に補完されるため、再入力の手間が省けます。データベースの設計仕様にに基づきソースコードを完成させ、コンパイルが成功するとデータベースが登録され「開く」コマンドで表ウィンドウを開くことができるようになります。

「プロシージャ種別」を切り替えることで、DB定義の他に、一括処理、帳票作成プロシージャもこの画面から同様に登録することができます。

完成したソースコードファイルがすでにあり、これをコンパイルしたいときは、「ソースコード処理選択」を「コンパイル(C)」にします。これは、「ツール(T)」 「コンパイル(C)」コマンドを選択するのと同じです。（3.9.1参照）ソースコードファイルがすでにあり、これを編集したいときは、「ソースコード処理選択」を「既存編集(E)」にします。これは、「ツール(T)」 「テキスト編集(T)」コマンドを選択するのと同じです。（3.9.2参照）

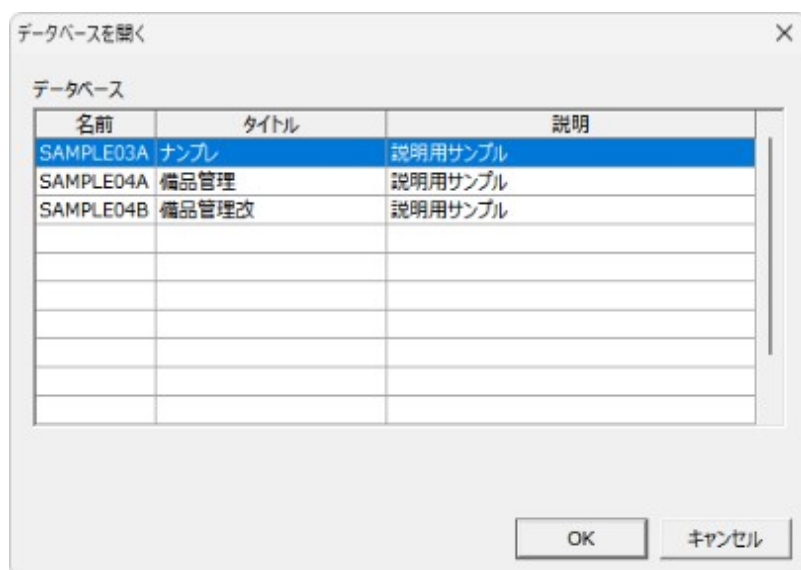
※注意※ 新規作成したソースコードを「上書保存」すると、ソースコードファイルはサイト内に保存されます。スタンドアロンで使用する場合を除き、サイト内にソースコードを置くのは安全ではありません。「名前を付けて保存」で開発者のクライアントPCに保存してください。（実運用サイトとは別にスタンドアロンの開発テスト用サイトを開発者のPCに置くのが正解です。）

Bolyai内蔵のテキストエディタは、最小限の機能しか持たないため、ソースコードの編集には、それに適したエディタを使用してください。

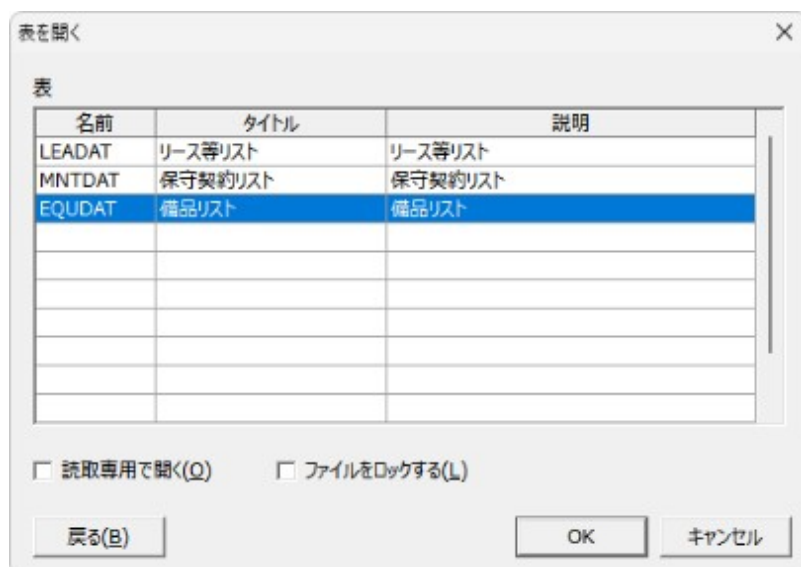
ストアドプロシージャプログラミングについては、このマニュアルの5章を、プロシージャ中で使用できるBolyaiのAPI関数については、このマニュアルの6章を、それぞれ参照してください。

3.6 表を開く

表データを閲覧／編集するときは、[Ctrl]+[O]キーまたはツールバーの「開く」ボタンを押します。他に表ウィンドウを開いていない状態（データベースに未接続の状態）で「開く」コマンドを選択すると、「データベースを開く」ダイアログボックスが現れます。



「データベースを開く」ダイアログボックス上でデータベースを選択して「OK」を押すと、次に「表を開く」ダイアログボックスが現れます。



「表を開く」ダイアログボックス上で表を選択して「OK」を押すと、表ウィンドウが開きます。表ウィンドウを開いた状態（データベースに接続した状態）で「開く」コマンドを選択すると、「データベースを開く」ダイアログボックスは現れず、直接「表を開く」ダイアログボックスが現れます。一度に複数のデータベースに接続することはできないため、同時に開くことができるのは、同じデータベース内の表のみです。

「読取専用で開く(O)」オプションを選択すると、表ウィンドウが読取専用モードとなり、編集コマンドが使用できなくなります。

「ファイルをロックする(L)」オプションを選択すると、表ウィンドウを閉じるまでファイルがロックされ、その間他のユーザーが同ファイルを編集できなくなります。これがペシミスティック・ロックです。ファイルがロックされた状態でも、他のユーザーが読取専用モードで表を参照することは可能です。

3.7 表の操作

3.7.1 現在セル

セルをクリックするとそのセルが現在セルになります。キー操作では、[↑][↓][←][→]で上下左右、[Enter]または[Tab]で次、[Shift]+[Tab]で前、[Home]で行頭、[End]で行末、[Ctrl]+[Home]で先頭行の行頭、[Ctrl]+[End]で最終行の行末、[Page Up]で一画面分前方、[Page Down]で一画面分後方のセルに移動します。スクロールバーを使用すると、現在セルは変わらず表示領域のみ移動します。

3.7.2 セルの編集

セルをダブルクリックするか、スペースバー、[F2]キーまたは任意の文字キーを押すと、現在セルが編集モードになり、エディットコントロールまたはドロップダウンリストが開きます。[F4]キーを押すと、1つ上のセルの内容をコピーして編集モードに入ります。[Enter]キー等で他のセルに移動すると、編集内容が確定します。他のセルに移動する前に[Esc]キーを押すと、編集内容がキャンセルされます。

セルを右クリックすると、次のようなポップアップメニューが現れます。

	切り取り(T)	Ctrl+X
	コピー(C)	Ctrl+C
	貼り付け(P)	Ctrl+V
クリア(A)		Del

- 「切り取り(T)」 現在セルの内容をクリップボードに移動します。
- 「コピー(C)」 現在セルの内容をクリップボードにコピーします。
- 「貼り付け(P)」 クリップボードの内容を現在セルにコピーします。
- 「クリア(A)」 現在セルを空にします。

3.7.3 行編集

行単位の編集は、行範囲を選択して行ないます。行番号セルをクリックするとその行が選択状態になり、そのままドラッグするとその範囲の行ブロックが選択状態になります。キー操作では、[Shift]+[Enter]で現在行が選択され、[Shift]を押しながら[↑][↓][Page Up][Page Down]等で移動するとその範囲の行ブロックが選択されます。

行番号セルを右クリックすると、次のようなポップアップメニューが現れます。

	切り取り(T)	Ctrl+X
	コピー(C)	Ctrl+C
	貼り付け(P)	Ctrl+V
挿入(I)		Ctrl+E
削除(D)		Ctrl+D
クリア(A)		Del

- 「切り取り(T)」 選択行範囲をクリップボードに移動します。
- 「コピー(C)」 選択行範囲をクリップボードにコピーします。
- 「貼り付け(P)」 クリップボード内の行ブロックを選択位置に挿入します。
- 「挿入(I)」 選択行範囲に空行を挿入します。
- 「削除(D)」 選択行範囲を削除します。
- 「クリア(A)」 選択行範囲を空行にします。

貼り付け、挿入、クリア等のコマンドを実行した結果、表が主キー制約違反や一意性制約違反の状態になる場合があります。制約違反状態のまま表データをファイルに保存することはできません。

3.7.4 文字列検索

[Ctrl]+[F]キーまたはツールバーの「検索」ボタンを押すと、「検索」ダイアログボックスが現れます。

検索する文字列を入力し、「OK」ボタンを押すと、現在列内で現在セル位置から下方向に文字列を検索し、見つければその行に現在セルを移動します。

[F3]キーを押すと、同じ文字列を続けて検索します。

3.7.5 表示項目

ツールバーの「表示項目」ボタンを押すと、「表示形式：表示項目」ダイアログボックスが現れます。

表示形式

表示項目 絞り込み 並べ替え

表示項目(列タイトル)

備品ID
品目
事業所
状態
メーカー名
製品名/型番
製造番号等
スペック等
主要用途等
リース等ID
保守ID
個体保守コード
IPアドレス
個体撤去日

<追加(A)
>除外(D)

非表示項目

リース開始
リース満期
リース終了
リース状態
保守開始
保守満期
保守終了
保守状態

固定列数(F): 0

OK キャンセル

左側の「表示項目」ボックスが、現在表示されている項目で、ボックス内の並び順が現在の表示順を示します。右側の「非表示項目」ボックスは、現在表示されていない項目です。

「<追加(A)」ボタンを押すと、「非表示項目」ボックス中の選択項目が「表示項目」ボックス中の選択位置に挿入される形で移動します。「>除外(D)」ボタンを押すと、「表示項目」ボックス中の選択項目が「非表示項目」ボックスに移動します。これによって表示する列とその順序を任意に指定できます。

「固定列数(F)」欄には、横スクロール時に固定する（スクロールアウトしない）先頭列数を指定します。

3.7.6 絞り込み

ツールバーの「絞り込み」ボタンを押すと、「表示形式：絞り込み」ダイアログボックスが現れます。

表示形式

表示項目 絞り込み 並べ替え

絞り込み条件

and/or	列タイトル	比較	値
and/or	品目	≥	パソコン
and	品目	≤	その他機器
and	状態	<	故障
▼			

絞り込み条件式

cat >= 1 and cat <= 7 and sta < 2

OK キャンセル

絞り込みの条件は、キーとなる項目、比較演算子、値の組み合わせで指定します。
9つまでの条件を「and」または「or」で結合して指定できます。

3.7.7 並べ替え

ツールバーの「並べ替え」ボタンを押すと、「表示形式：並べ替え」ダイアログボックスが現れます。

表示形式

表示項目

絞り込み

並べ替え

並べ替え条件

優先順位	列タイトル	並べ替え順序
1	事業所	昇順
2	品目	昇順
3	備品 I D	昇順
4		
5		
6		
7		
8		
9		
10		

並べ替え条件式
ofc, cat, qid

OK

キャンセル

並べ替え条件は、キーとなる項目と昇順／降順等の並べ替え順序の組み合わせで指定します。
並べ替えキー項目は10個まで選ぶことができ、上から順に優先して適用されます。

3.7.8 表示形式

[F5]キーまたはツールバーの「表示形式」ボタンを押すと、表示形式選択のダイアログボックスが現れます。

表示形式

表示形式

表示形式名	定義
未登録	ユーザー

登録(B)

名前変更(C)

削除(D)

OK

キャンセル

現在の表示形式（表示項目／絞り込み／並べ替えの指定の組み合わせ）に名前を付けて登録しておくと、次からは表示形式名を選択するだけでその表示形式を適用することができます。
登録した表示形式のセットは、表ごとに **表名.VIW** というファイル名でクライアントPCのユーザーフォルダに保存されます。最後に選択した表示形式は、ファイル中に記録され、次に同じ表を開いたときに自動的に適用されます。

3.7.9 全項目表示

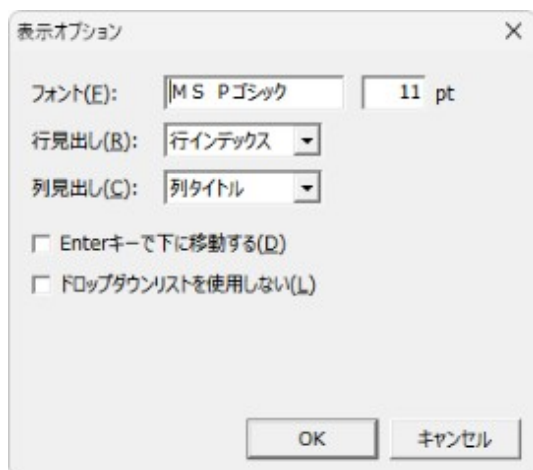
[F10]キーを押すと、現在の表示項目と絞り込みの指定が解除され、全ての列（計算項目を含む）と行が現在の並べ替え指定に基づいて表示されます。

3.7.10 表示形式解除

[Shift]+[F10]キーを押すと、表示形式の指定は全て解除され、全ての列（計算項目は含まない）と行が実表上での物理的な並び順のまま表示されます。

3.7.11 表示オプション

「表示(V)」 「表示オプション(0)」 コマンドを選ぶと、「表示オプション」ダイアログボックスが現れます。



表示オプション

フォント(E): MS Pゴシック 11 pt

行見出し(B): 行インデックス

列見出し(C): 列タイトル

☐ Enterキーで下に移動する(D)

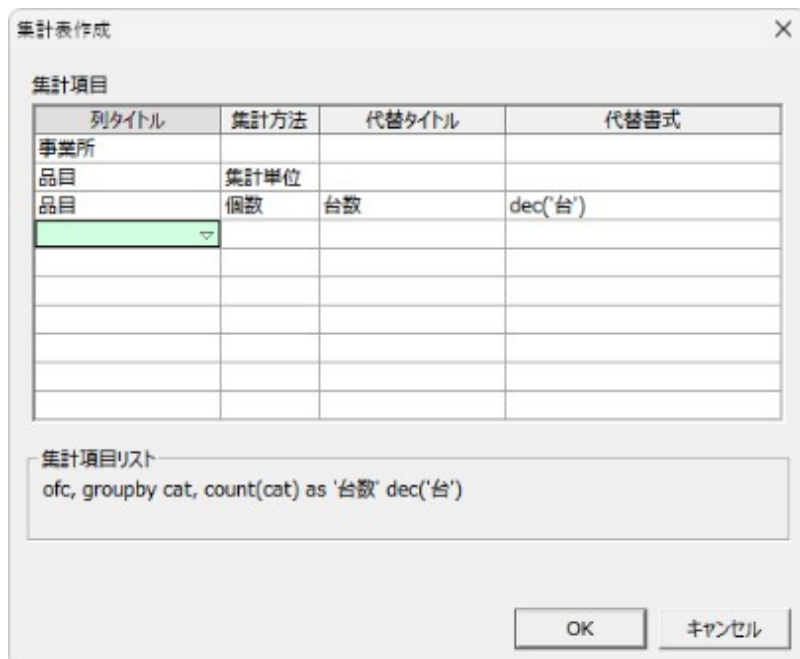
☐ ドロップダウンリストを使用しない(L)

OK キャンセル

ここでの設定は、現在アクティブな表ウィンドウに対してのみ適用されます。（表ウィンドウを閉じると設定はリセットされます。）全ての表ウィンドウに永続的な設定をしたいときは、「ツール(T)」 「オプション設定(0)」 コマンドを使用します。

3.7.12 集計表

ツールバーの「集計」ボタンを押すと、「集計表作成」ダイアログボックスが現れます。



集計表作成

集計項目

列タイトル	集計方法	代替タイトル	代替書式
事業所			
品目	集計単位		
品目	個数	台数	dec('台')

集計項目リスト

ofc, groupby cat, count(cat) as '台数' dec('台')

OK キャンセル

集計表は、表ウィンドウ上で、キーとなる項目の値が同じ行が連続している場合、この行のグループ（集計単位）ごとに、項目の個数、数値項目の合計値／平均値／最大値／最小値等を算出して表にしたものです。（集計表を作成するときは、最初に元となる表を集計したいグループごとに並べ替えておきます。）

集計単位のキーとする項目は、「集計方法」欄に「集計単位」を入れ、集計対象とする項目は、「集計方法」欄に、「合計」「平均」「最大値」「最小値」「個数」のいずれかを入れます。

集計対象項目には、代替タイトルと代替書式（10進書式に限る）を指定することができます。

代替書式は、次のような文字列で指定します。

```
dec   dec(w)   dec(w, s)   dec(u)   dec(w, u)   dec(w, s, u)
cdec  cdec(w)  cdec(w, s)  cdec(u)  cdec(w, u)  cdec(w, s, u)
```

dec は 3 桁コンマなし、cdec は 3 桁コンマあり、w は列幅、s は小数点以下桁数、u は単位を表す付加文字列を示します。

「集計方法」欄を空欄にすると、項目の値が（集計単位内で同一の場合に限り）そのまま表示されます。

集計表は、元の表とは別の読取専用の表ウィンドウに出力されます。

3.7.13 チェック

[Shift]+[F5] キーまたはツールバーの「チェック」ボタンを押すと、表データがチェックされ、メッセージボックスでチェック結果が表示されます。チェックされる事項は、以下の通りです。

- (1) 非ゼロ制約（非ゼロ制約が指定されている列がある場合）
- (2) 一意性制約（一意性制約が指定されている列がある場合）
- (3) 主キー制約
- (4) 外部キー制約
- (5) チェック制約（チェック制約が登録されている列がある場合）
- (6) 表トリガによるチェック（表トリガが登録されている表の場合）

3.7.14 ダイアログボックス入力

[F8] キーまたはツールバーの「行編集」ボタンを押すと、現在行の内容を編集する行編集ダイアログボックスが呼び出され、ダイアログボックス上で現在行のデータを編集できるようになります。

[F9] キーまたはツールバーの「行挿入」ボタンを押すと、現在位置に空行を挿入した上で行編集ダイアログボックスが呼び出されます。

[Shift]+[F7] キーまたはツールバーの「行追加」ボタンを押すと、表の末尾に空行を追加した上で行編集ダイアログボックスが呼び出されます。

3.7.15 物理的に並べ替え

「データ(D)」 「物理的に並べ替え(P)」 コマンドを実行すると、現在画面上に表示されている並び順で実表データの行が物理的に並べ替えられます。（絞り込み条件で除外された行は末尾に送られます。）

3.7.16 閉じる

表ウィンドウを閉じる際に、表データが変更されていれば、保存するかどうかの確認メッセージが現れます。「はい」を選ぶと表データがファイルに上書保存（次項参照）されます。

3.7.17 上書保存

[Ctrl]+[S] キーまたはツールバーの「上書保存」ボタンを押すと、表データがファイルに上書保存されます。

保存の前に、表データのチェック（3.7.13参照）が行われ、エラーがあると保存はキャンセルされます。

オプティミステック・ロック（「ファイルをロックする(L)」 オプションを選択しない）での運用の場合、ファイルを読込後、他のユーザーが同じファイルを開いて編集保存している可能性があります。このため保存時には、単にメモリ上の表データを書き出すのではなく、読込時の表データと現時点のディスク上の表データとの差分を、保存しようとしているメモリ上の表データと併合してから書き出します。このとき、もし編集内容が競合して（主キーで識別される同じ行の内容が異なると）いると、併合は失敗し、保存はキャンセルされます。

上書保存後の表データには、他のユーザーの編集内容が（もしあれば）反映されています。

3.7.18 テキスト取込

テキストファイルから表にデータを取り込むときは、「ファイル(F)」 「テキスト取込(I)」 コマンドを選択します。最初にファイルを選択する「開く」ダイアログボックスが現れ、ファイル名を入力すると、次にテキストファイルの形式を選択するダイアログボックスが現れます。

区切り文字は、タブ／コンマ／セミコロン／スペースのいずれか、引用符文字は、なし／シングルクォート／ダブルクォートのいずれかを指定できます。

タイトル行は、なし／列名／列タイトルのいずれかを選択します。タイトル行を列名にした場合のみ、表ウィンドウの表示項目とテキストファイル上の項目の順番が異なっても正しい位置に読み込むことができます。列名を使用しない場合は、両方の項目の順番が同じであると仮定して読み込みを行います。

3.7.19 テキスト書出

テキストファイルに表のデータを書き出すときは、「ファイル(F)」 「テキスト書出(E)」 コマンドを選択します。

最初にファイルを選択する「名前を付けて保存」ダイアログボックスが現れ、ファイル名を入力すると、次にテキストファイルの形式を選択するダイアログボックス（前節と同じ）が現れます。

3.7.20 印刷

表をプリンタに出力するときは、「ファイル(F)」 「印刷(P)」 コマンドを選択します。（[Ctrl]+[P]キーまたはツールバーの「印刷」ボタンも使えます。）

最初に次のような印刷用紙設定のダイアログボックスが現れます。

印刷用紙設定

用紙

サイズ(Z): A 4 (210 x 297 mm)

方 向: ☒ 縦(P) ☐ 横(A)

余白(mm)

左(L): 20 上(T): 25 右(R): 20

下(B): 25

拡大縮小

☐ 横幅に合わせる(J)

☒ 倍 率(S): 100 %

OK キャンセル

用紙設定をして「OK」ボタンを押すと、印刷プレビュー画面が現れます。

印刷プレビュー画面のツールバーには、以下のボタンが並んでいます。

「印刷」 プレビュー画面をプリンタに出力する「印刷」ダイアログボックスを呼び出します。

「前ページ」 前ページを表示します。

「次ページ」 次ページを表示します。

「用紙設定」 印刷用紙設定ダイアログボックスを呼び出します。

「拡大」「縮小」「等倍表示」「全体表示」 表示倍率を切り替えます。

以下のショートカットキーも使えます。

[Ctrl]+[Alt]+[T] 印刷

[Ctrl]+[Alt]+[P] 前ページ

[Ctrl]+[Alt]+[N] 次ページ

[Ctrl]+[Alt]+[Z] 等倍表示／全体表示の切り替え

[Ctrl]+[Alt]+[S] 用紙設定

3.7.21 単票印刷

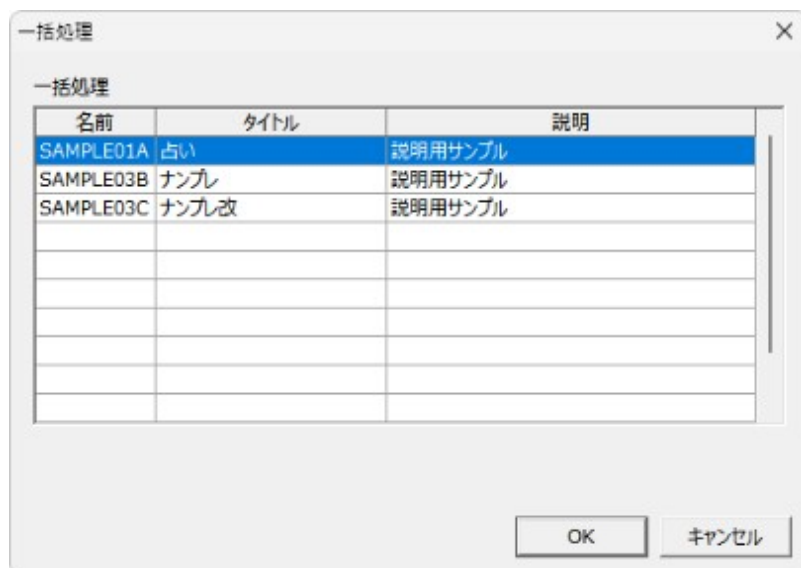
表の1行分のデータを1枚の帳票としてプリンタに出力するときは、「ファイル(F)」 「単票印刷(L)」 コマンドを選択します。

印刷用紙設定と印刷プレビュー画面の操作は前節と同じです。

3.8 処 理

3.8.1 一括処理

「処理(P)」 「一括処理(X)」 コマンドを選択すると、「一括処理」ダイアログボックスが現れます。（このコマンドは、表ウィンドウを開いた状態では実行できません。）



実行する処理を選択して「OK」を押すと、一括処理プロシージャが呼び出されます。

3.8.2 帳票作成

「処理(P)」 「帳票作成(R)」 コマンドを選択すると、「帳票作成」ダイアログボックスが現れます。（このコマンドは、表ウィンドウを開いた状態では実行できません。）



作成する帳票を選択して「OK」を押すと、帳票作成プロシージャが呼び出されます。

3.8.3 DB接続状況／ログイン状況

「処理(P)」 「DB接続状況(D)」 コマンドを選択すると、現在DBに接続中のユーザー名のリストが画面表示されます。

「処理(P)」 「ログイン状況(S)」 コマンドを選択すると、現在サイトにログイン中のユーザー名のリストが画面表示されます。

3.8.4 ロック解除

プログラムの異常終了等により、ファイルがロックされたままになる場合があります。そのときは、ファイルをロック中のユーザーを強制的にログアウトさせることでファイルのロックを解除します。このコマンドの実行には、DB管理者以上の権限が必要です。

3.8.5 ファイル転送

サイト内のデータベース構成ファイル、一括処理、帳票作成のバイトコードファイルを選択して指定のフォルダにコピーまたは移動します。このコマンドは、データのバックアップ、他サイトへのデータコピー、サイトからのデータ削除の目的で 사용할 ことができます。このコマンドの実行には、DB管理者以上の権限が必要です。

3.9 ツール

3.9.1 コンパイル

「ツール(T)」 「コンパイル(C)」 コマンドを選択すると、「コンパイル」ダイアログボックスが現れます。ソースコードファイル名を指定して「OK」ボタンを押すと、ファイルがコンパイルされ、コンパイルが成功すると、サイトにプロシージャが登録されます。このコマンドの実行には、DB管理者以上の権限が必要です。

3.9.2 テキスト編集

「ツール(T)」 「テキスト編集(T)」 コマンドを選択すると、ファイル名を入力する「開く」ダイアログボックスが現れます。テキストファイル（ソースコードファイル）を選ぶと、ファイルが呼び出されテキスト編集画面になります。このコマンドの実行には、DB管理者以上の権限が必要です。

テキスト編集画面のツールバーには、以下のボタンが並んでいます。

「開く」	新たなテキストファイルを開きます。
「上書保存」	テキストファイルを現在の名前で保存します。
「印刷」	テキストファイルを印刷します。
「名前を付けて保存」	テキストファイルに名前を付けて保存します。
「チェック」	テキストファイルをテストコンパイルします。
「コンパイル」	テキストファイルをコンパイルします。

テストコンパイルは、完成していないコード断片でも実行できます。

3.9.3 オプション設定

「ツール(T)」 「オプション設定(O)」 コマンドを選択すると、「オプション設定」ダイアログボックスが現れます。（このコマンドは、表ウィンドウを開いた状態では実行できません。）

The screenshot shows the 'オプション設定' (Option Settings) dialog box with the following settings:

- Font (F): MS Pゴシック, 11 pt
- Row display (R): 行インデックス, Cell pointer color (P): E0FFD0
- Column display (C): 列タイトル, Display highlight (H): D0D0D0
- ☐ Enter key moves down (D)
- ☐ Index starts from 0 (Z)
- ☐ Do not use dropdown list (L)
- ☐ Do not use dialog box (E)
- ☐ Auto login (A), User name: T.Yamada
- ☐ Use start menu (M)
- ☐ Auto connect to DB (N), DB name:
- ☐ Auto open table (O), Table name:
- ☐ Do not save error log (G)

Buttons: OK, キャンセル

ここでは、表ウィンドウの表示と操作に関する項目、プログラム起動時の動作等をカスタマイズできます。

「フォント(F)」は、表ウィンドウで使用するフォント名です。（印刷時にも使用されます。）

「セルポインタ色(P)」と「見出しハイライト(H)」は、16進表記（bbggrr）で色を指定します。

「自動的にDBに接続する(N)」と「自動的に表を開く(O)」オプションは、BOLYAI.INI ファイルを直接テキストエディタで編集し、StartDB= と OpenTbl= にそれぞれDB名と表名を与えた場合に限り選択可能になります。

3.9.4 計算日設定

計算日（計算上の現在日付）を変更するときは、「ツール(T)」 「計算日設定(D)」 コマンドを選択します。（このコマンドは、表ウィンドウを開いた状態では実行できません。）

4 Bolyai-Cの言語仕様

4.1 Bolyai-Cとは

Bolyai-Cは、Bolyaiに組み込まれて提供されるアプリケーション記述用のスクリプト言語です。

Bolyai-Cの文法は、C言語文法のサブセットです。式と文の構文は、ほぼC言語そのままであり、標準C言語の構文を逸脱するような要素はないため、C言語を使っているつもりでコーディングできます。（ただし一定のコーディングスタイルに従う必要があります。）

Bolyai-Cにおいて、標準C言語から省かれた主な要素は、以下の通りです。

- (1) プリプロセッサ
- (2) `union`（共用体）型
- (3) `signed` / `unsigned` / `long` 指定子
- (4) `register` 指定子、`volatile` 修飾子
- (5) 構造体タグ、列挙子タグ
- (6) ビットフィールド
- (7) 型宣言構文の簡略化
- (8) その他言語機能上の制限

省略要素のほとんどは、他に代替手段のあるもの、またはスクリプト言語には必要のないものです。これらの制限を回避して同等の機能を実現するためのコーディングスタイルについて、以下に説明します。

4.1.1 プリプロセッサ

Bolyai-Cにはプリプロセッサがないため、`#include` や `#ifdef` 等の `#` で始まる指令をプログラムに含めることができません。

例外的に `#define` 指令のみは、算術型の定数を宣言するために用いることができます。ただし任意の字句をマクロ置換できるわけではなく、宣言場所よりも以前のソースコード内で定義済みの定数リテラルからなる式の値に新たな定数名を与えることができるだけです。

定数宣言は、以下のよう使い分けます。

- (1) `#define` 指令

- ・4byte整数型、8byte整数型、8byte実数型の定数リテラルを定義します。値は以降の定数式中使用できます。アドレスを取ることはできません。

- (2) `enum` 宣言

- ・4byte整数型の値を表す定数リテラルを列挙して定義します。値は以降の定数式中使用できます。アドレスを取ることはできません。

- (3) `const` で始まる変数宣言

- ・任意の型の不変の変数を定義できます。定数式中で値を使用することはできません。アドレスを取ることができ、キャストすれば強制的に値を書き換えることもできます。

Bolyai-Cの `#define` 指令は、引数を取る関数型のマクロを定義できません。

なお、以下のビルトインマクロは、言語に組み込まれたものとして、定義なしで使用できます。

<code>NULL</code>	<code>((void *)0)</code> に同じ
<code>NELEM(a)</code>	<code>(sizeof(a) / sizeof(a[0]))</code> に同じ
<code>_countof(a)</code>	<code>NELEM(a)</code> に同じ
<code>LOU1B(w)</code>	<code>((U1B)((U4B)(w) & 0x00ff))</code> に同じ
<code>HIU1B(w)</code>	<code>((U1B)((U4B)(w) >> 8))</code> に同じ
<code>LOU2B(u)</code>	<code>((U2B)((U4B)(u) & 0xffff))</code> に同じ
<code>HIU2B(u)</code>	<code>((U2B)((U4B)(u) >> 16))</code> に同じ
<code>MKU2B(l, h)</code>	<code>((U2B)(((U4B)(l) & 0x00ff) ((U4B)(h) << 8)))</code> に同じ
<code>MKU4B(l, h)</code>	<code>((U4B)(((U4B)(l) & 0xffff) ((U4B)(h) << 16)))</code> に同じ
<code>LOBYTE(w)</code>	<code>LOU1B(w)</code> に同じ
<code>HIBYTE(w)</code>	<code>HIU1B(w)</code> に同じ
<code>LOWORD(u)</code>	<code>LOU2B(w)</code> に同じ
<code>HIWORD(u)</code>	<code>HIU2B(w)</code> に同じ
<code>MAKEWORD(l, h)</code>	<code>MKU2B(l, h)</code> に同じ
<code>MAKEUINT(l, h)</code>	<code>MKU4B(l, h)</code> に同じ
<code>STR(n)</code>	<code>MKU4B(TID_STR, n)</code> に同じ
<code>BIN(n)</code>	<code>MKU4B(TID_BIN, n)</code> に同じ
<code>ASC(n)</code>	<code>MKU4B(TID_ASC, n)</code> に同じ
<code>CAL(id)</code>	<code>((id) CAL_BIT)</code> に同じ
<code>ITP(id, bt)</code>	<code>MKU2B(id, bt)</code> に同じ
<code>ATR(wd, al)</code>	<code>MKU2B(wd, al)</code> に同じ

ALT(id, ic) MKU4B(id, ic) に同じ
RKY(id, ic) MKU4B(id, ic) に同じ

ビルトインマクロは、関数ではないので定数式中でも使用できます。

4.1.2 共用体型

共用体型は、キャストを使用すれば同等の機能を実現できます。

4.1.3 型指定子（基本型名）

C 言語では同じ基本型を示す複数の表記（`unsigned / unsigend int / int unsigned` 等）が許容されていますが、Bolyai-C では、修飾語を使用せずに全ての基本型を最初から 1 キーワードで表記します。

I1B 1byte 符号付整数（別名 `char, _i08`）
U1B 1byte 符号無整数（別名 `BYTE, _u08`）
I2B 2byte 符号付整数（別名 `short, _i16`）
U2B 2byte 符号無整数（別名 `WORD, USHORT, _u16`）
I4B 4byte 符号付整数（別名 `int, BOOL, LONG, _i32, iANY`）
U4B 4byte 符号無整数（別名 `UINT, DWORD, _u32, uANY, RGBC, BGRC`）
I8B 8byte 符号付整数（別名 `INT64, _i64`）
U8B 8byte 符号無整数（別名 `UINT64, _u64`）
F4B 4byte 単精度実数（別名 `float, _f32`）
F8B 8byte 倍精度実数（別名 `double, _f64, REAL`）

4.1.4 記憶クラス指定子、型修飾子

Bolyai-C の記憶クラス指定子は、`auto / static / extern` のみ、型修飾子は、`const` のみです。このうち自動変数の `auto` は常に省略可能で、外部変数の `static` と `extern` は何の効果も持ちません。関数内の `static` は局所変数が静的変数であることを示します。

記憶クラス指定子、型修飾子、型指定子は、この順番で記述しなければなりません。

```
const int i = 0;
```

は有効ですが、

```
int const i = 0;
```

はコンパイルエラーになります。

4.1.5 構造体タグ、列挙子タグ

構造体タグ、列挙子タグは使用できません。構造体タグが使用できないことで唯一困るのは、自己参照型の構造体を記述できないことですが、`PVOID` 型をそのつどキャストして使うことで同等の機能を実現することは可能です。

4.1.6 ビットフィールド

ビットフィールドは、ビット操作演算子を使用すれば同等の機能を実現できます。

4.1.7 型宣言構文の簡略化

配列型、構造体型、関数ポインタ型等の宣言構文が簡略化されており、これらの宣言は、一定の場所で一定の構文でしか宣言できなくなっています。ほとんどの制限は、複雑な型宣言構文を使わずに、常に `typedef` 名を使うことで回避できます。

具体的な制限事項には以下のようなものがあります。

(1) 構造体型と関数ポインタ型の導入には、`typedef` 名が必要です。

```
struct { int x; int y; } pt, *pp;
```

とは宣言できないので、

```
typedef struct { int x; int y; } POINT;  
POINT pt, *pp;
```

と宣言してください。

(2) ポインタの配列、配列へのポインタ、戻り値がポインタ型の関数の定義および宣言には `typedef` 名が必要です。

```
void *func(void *pv) { ... }
```

ではなく、

```
PVOID func(PVOID pv) { ... }
```

と書いてください。

(3) キャストの構文は、型指定子、または型指定子の後ろにポインタを置いたものを括弧で囲った形に限ります。

(`int`)、(`int *`)、(`const int *`)

は有効ですが、

(`int (*)[4]`)、(`int (*)(int, int)`)

などはコンパイルエラーになります。

(4) 変数については、宣言と定義の区別をしません。したがって定義の前に宣言を置くことはできません。

```
int a;
```

...

```
int a = 0;
```

と書くことで識別子の重複でコンパイルエラーになります。

(5) 関数の宣言では仮引数名を省略できません。また、宣言と定義の引数名は一致していなければなりません。

```
int foo(int, int);
```

や

```
int foo(int a, int b);
```

```
int foo(int x, int y) { ... }
```

とは書けません。

(6) 引数なしの関数は (void) で表現します。 () だとコンパイルエラーになります。

```
int main() { ... }
```

ではなく、

```
int main(void) { ... }
```

と書いてください。

(7) 配列の要素数を省略した不完全型の配列は宣言できません。

```
int a[];
```

は、

```
int *a;
```

と書いてください。

(8) 初期化式のリストがある場合でも配列の要素数は省略できません。

```
int x[] = { 1, 3, 5 };
```

は、

```
int x[3] = { 1, 3, 5 };
```

と書いてください。

4.1.8 その他言語機能上の制限

言語機能上以下の制限があります。

(1) 関数本体以外の任意の文ブロック先頭での変数宣言はできません。

```
void foo(void) { ... { int t; t = a; a = b; b = t; } ... }
```

とは書けず

```
void foo(void) { int t; ... { t = a; a = b; b = t; } ... }
```

と書くことになります。

(2) 構造体型や配列型の自動変数の宣言に初期化式のリストを与えることはできません。

```
void foo(void) { int a[3] = { 1, 3, 5 }; ... }
```

とは書けないので

```
void foo(void) { int a[3]; a[0] = 1; a[1] = 3; a[2] = 5; ... }
```

と書きます。

```
void foo(void) { static int a[3] = { 1, 3, 5 }; ... }
```

のような静的変数の場合は大丈夫です。

(3) 構造体の一括代入や値渡しはできません。 memcpy 等を使用してください。

```
POINT p1, p2; ... p1 = p2;
```

は、

```
POINT p1, p2; ... memcpy(&p1, &p2, sizeof(POINT));
```

のように書きます。

(4) 構造体のメンバ間に自動的にパディングが入ることはありません。

```
typedef struct { int a; double b; ... } BAR;
```

このようなメンバ配置の場合、

```
typedef struct { int a; UINT rsv; double b; ... } BAR;
```

このように明示的にパディング要素を記述してください。

(5) typedef 宣言と enum 宣言は、外部宣言のみ可能です。関数内で局所的に宣言することはできません。

(6) sizeof 演算子は、sizeof(型名) または sizeof(変数名) のみ使用可能です。sizeof 式 は使えません。

(7) 静的に確保できる配列の容量に制限があります。 calloc 等で動的に確保してください。

(8) 8byte整数型を、配列の要素数、列挙子の値、switch 文と case ラベルの式の値に使用することはできません。

4.2 文法の記述

4.2.1 字句規則の要約

字句規則は、プリプロセッサがないことを除きK&R第2版時代のC言語と同等です。C++形式のコメント//と、符号無整数型および8byte整数型を示すサフィックスUとLLは、サポートしています。文字定数リテラル、文字列リテラルのエンコード方式を示すプレフィックス等は、サポートしていません。文字コードは、Shift_JISです。

4.2.2 構文規則の要約

以下の拡張BNF記法を使用して文法を記述します。

: 定義
/ どれか一つの選択
[] 省略可能な要素

終端記号は、以下の通りです。

識別子 (*IDENTIFIER*)
整数定数、文字定数、浮動小数点定数 (*CONSTANT*)
文字列リテラル (*STRING*)
キーワード、演算子、区切り子 (直立のゴシック体で示す)

```
translation_unit  
: external_declaration  
/ translation_unit external_declaration
```

```
external_declaration  
: typedef_declaration  
/ define_directive  
/ enum_declaration  
/ variable_declaration  
/ function_declaration  
/ function_definition
```

```
typedef_declaration  
: typedef [ const ] struct_specifier IDENTIFIER ;  
/ typedef [ const ] type_specifier ( * IDENTIFIER ) ( parameter_type_list ) ;  
/ typedef [ const ] type_specifier declarator_list ;
```

```
struct_specifier  
: struct { struct_declaration_list }
```

```
struct_declaration_list  
: struct_declaration  
/ struct_declaration_list struct_declaration
```

```
struct_declaration  
: [ const ] type_specifier declarator_list ;
```

```
declarator_list  
: declarator  
/ declarator_list , declarator
```

```
declarator  
: direct_declarator  
/ pointer IDENTIFIER
```

```
direct_declarator  
: IDENTIFIER  
/ direct_declarator [ constant_expression ]
```

```
constant_expression  
: conditional_expression
```

type_specifier
: void / I1B / U1B / I2B / U2B / I4B / U4B / I8B / U8B / F4B / F8B
/ char / BYTE / short / WORD / int / UINT / INT64 / UINT64 / float / double
/ typedef_name

typedef_name
: IDENTIFIER

parameter_type_list
: parameter_list
/ parameter_list , ...

parameter_list
: parameter_declaration
/ parameter_list , parameter_declaration

parameter_declaration
: [const] type_specifier declarator

define_directive
: #define IDENTIFIER constant_expression

enum_declaration
: enum { [enumerator_list] } ;
/ enum { enumerator_list , } ;

enumerator_list
: enumerator
/ enumerator_list , enumerator

enumerator
: IDENTIFIER
/ IDENTIFIER = constant_expression

variable_declaration
: [storage_class_specifier] [const] type_specifier init_declarator_list ;

storage_class_specifier
: auto / static / extern

init_declarator_list
: init_declarator
/ init_declarator_list , init_declarator

init_declarator
: declarator
/ declarator = initializer

initializer
: assignment_expression
/ { initializer_list }
/ { initializer_list , }

initializer_list
: initializer
/ initializer_list , initializer

```

function_declaration
: [ const ] type_specifier IDENTIFIER ( parameter_type_list ) ;

function_definition
: [ const ] type_specifier IDENTIFIER ( parameter_type_list ) function_body

function_body
: { [ variable_declaration_list ] [ statement_list ] }

variable_declaration_list
: variable_declaration
/ variable_declaration_list variable_declaration

statement_list
: statement
/ statement_list statement

statement
: labeled_statement
/ expression_statement
/ compound_statement
/ selection_statement
/ iteration_statement
/ jump_statement

labeled_statement
: IDENTIFIER : statement
/ case constant_expression : statement
/ default : statement

expression_statement
: [ expression ] ;

compound_statement
: { [ statement_list ] }

selection_statement
: if ( expression ) statement
/ if ( expression ) statement else statement
/ switch ( expression ) statement

iteration_statement
: while ( expression ) statement
/ do statement while ( expression ) ;
/ for ( [ expression ] ; [ expression ] ; [ expression ] ) statement

jump_statement
: goto IDENTIFIER
/ continue ;
/ break ;
/ return [ expression ] ;

expression
: assignment_expression
/ expression , assignment_expression

```

```

assignment_expression
: conditional_expression
/ unary_expression assignment_operator assignment_expression

assignment_operator
: = / += / -= / *= / /= / %= / <<= / >>= / &= / |= / ^=

conditional_expression
: logical_or_expression
/ logical_or_expression ? conditional_expression : conditional_expression

logical_or_expression
: logical_and_expression
/ logical_or_expression || logical_and_expression

logical_and_expression
: inclusive_or_expression
/ logical_and_expression && inclusive_or_expression

inclusive_or_expression
: exclusive_or_expression
/ inclusive_or_expression | exclusive_or_expression

exclusive_or_expression
: and_expression
/ exclusive_or_expression ^ and_expression

and_expression
: equality_expression
/ and_expression & equality_expression

equality_expression
: relational_expression
/ equality_expression == relational_expression
/ equality_expression != relational_expression

relational_expression
: shift_expression
/ relational_expression < shift_expression
/ relational_expression > shift_expression
/ relational_expression <= shift_expression
/ relational_expression >= shift_expression

shift_expression
: additive_expression
/ shift_expression << additive_expression
/ shift_expression >> additive_expression

additive_expression
: multiplicative_expression
/ additive_expression + multiplicative_expression
/ additive_expression - multiplicative_expression

multiplicative_expression
: cast_expression
/ multiplicative_expression * cast_expression
/ multiplicative_expression / cast_expression
/ multiplicative_expression % cast_expression

```


cast_expression
: *unary_expression*
/ (*type_name*) *cast_expression*

type_name
: [**const**] *type_specifier* [*pointer*]

pointer
: * [**const**]
/ * [**const**] *pointer*

unary_expression
: *postfix_expression*
/ ++ *unary_expression*
/ -- *unary_expression*
/ *unary_operator* *cast_expression*
/ **sizeof** (*type_identifier*)

unary_operator
: & / * / + / - / ~ / !

type_identifier
: *type_name*
/ *postfix_name*

postfix_name
: *IDENTIFIER*
/ *postfix_name* [*CONSTANT*]
/ *postfix_name* . *IDENTIFIER*
/ *postfix_name* -> *IDENTIFIER*

postfix_expression
: *primary_expression*
/ *postfix_expression* [*expression*]
/ *postfix_expression* ([*argument_expression_list*])
/ *postfix_expression* . *IDENTIFIER*
/ *postfix_expression* -> *IDENTIFIER*
/ *postfix_expression* ++
/ *postfix_expression* --

primary_expression
: *IDENTIFIER*
/ *CONSTANT*
/ *STRING*
/ *builtin_macro_call*
/ (*expression*)

builtin_macro_call
: **NULL**
/ **NELEM** (*postfix_name*)
/ **_countof** (*postfix_name*)
/ **LOU1B** (*assignment_expression*)
/ **HIU1B** (*assignment_expression*)
/ **LOU2B** (*assignment_expression*)
/ **HIU2B** (*assignment_expression*)
/ **MKU2B** (*assignment_expression* , *assignment_expression*)
/ **MKU4B** (*assignment_expression* , *assignment_expression*)
/ **LOBYTE** (*assignment_expression*)
/ **HIBYTE** (*assignment_expression*)

```
/ LOWORD ( assignment_expression )  
/ HIWORD ( assignment_expression )  
/ MAKEWORD ( assignment_expression , assignment_expression )  
/ MAKEUINT ( assignment_expression , assignment_expression )  
/ STR ( assignment_expression )  
/ BIN ( assignment_expression )  
/ ASC ( assignment_expression )  
/ CAL ( assignment_expression )  
/ ITP ( assignment_expression , assignment_expression )  
/ ATR ( assignment_expression , assignment_expression )  
/ ALT ( assignment_expression , assignment_expression )  
/ RKY ( assignment_expression , assignment_expression )
```

argument_expression_list

: assignment_expression

/ argument_expression_list , assignment_expression

4.3 言語処理系

4.3.1 コンパイラとバイトコードインタプリタ

Bolyai-Cのソースコードファイルは、Bolyaiのコンパイラによってバイトコードファイルにコンパイルされ、サイトに格納されます。これが、Bolyaiのストアードプロシージャです。バイトコードファイルは、必要に応じてクライアントの主記憶内にロードされ、Bolyaiのバイトコードインタプリタによって実行されます。

ソースコードファイルは、プレーンなテキストファイルとしてWindowsのメモ帳などのテキストエディタで作成します。コンパイラが対応しているのは、ANSI (Shift_JIS) 形式のテキストファイルのみです。

※注意※ Windows 10以降機のメモ帳は、デフォルトのエンコードがUTF-8なので、ANSIに切り替えて保存する必要があります。

コンパイルは、ソースコードファイルが文法的に正しければ成功しますが、作成されたプログラムが正しく動作するかどうかは当然ながら別の問題です。C言語の特性上、配列のオーバーランや不正な値のポインタによるアクセスはコンパイル時にはチェックできないので、プログラムのミスで不正なメモリ参照が発生すると、Bolyai本体が異常終了します。

4.3.2 ビルトイン関数

ストアードプロシージャ用として提供されているBolyaiのAPI関数には、次のようなものがあります。

- (1) C標準ライブラリ関数の一部
- (2) ユーティリティ関数
- (3) データベース操作関数
- (4) 帳票作成関数等

これらの関数は、言語に組み込まれたものとして、宣言なしに使用できます。宣言内容は、**BOLYAI_C.H**というテキストファイルに記述されています。**BOLYAI_C.H**はコンパイルのつど自動的に読み込まれます。**BOLYAI_C.H**内の関数の宣言順は、API関数呼出時の序数と対応しているため、並べ替え等の編集はしないでください。

4.3.3 既定の型名

BolyaiのAPI関数で共通的に使用される型名の定義を以下に示します。

```
typedef void *_p32, *P4B, *pANY, *PVOID;  
typedef I1B _i08, *pI1B, *pSTR, *PSTR;  
typedef U1B _u08, *pU1B, *pBIN, *PBYTE;  
typedef I2B _i16, *pI2B;  
typedef U2B _u16, *pU2B, *PWORD;  
typedef I4B _i32, *pI4B, iANY, *PINT;  
typedef U4B _u32, *pU4B, uANY, *PUINT;  
typedef I8B _i64, *pI8B;  
typedef U8B _u64, *pU8B;  
typedef F4B _f32, *pF4B;  
typedef F8B _f64, *pF8B, REAL;  
typedef const void *rANY, *PCVOID, *HFIL, *HANDLE;  
typedef const I1B *rI1B, *rSTR, *PCSTR;  
typedef const U1B *rU1B, *rBIN, *PCBYTE;  
typedef const I2B *rI2B;  
typedef const U2B *rU2B, *PCWORD;  
typedef const I4B *rI4B, *PCINT;  
typedef const U4B *rU4B, *PCUINT;  
typedef const I8B *rI8B;  
typedef const U8B *rU8B;  
typedef const F4B *rF4B;  
typedef const F8B *rF8B;  
typedef struct { I4B unused;} HWND_;  
typedef HWND_ *HWND;  
typedef U4B RGBC;  
typedef RGBC *pRGBC;  
typedef const RGBC *rRGBC;  
typedef U4B BGRC, COLORREF;
```

Bolyai本体のソースコードと第6章の関数リファレンスにおいては、型名は以下の標準名で統一しています。従来からの別名もそのまま使えるため、本マニュアル内の説明文およびサンプルソースコードでは別名を用いている場合があります。

標準名 別名 (標準名の表す意味)

I1B	char に同じ
U1B	BYTE に同じ
I2B	short に同じ
U2B	WORD, USHORT に同じ
I4B	int, LONG に同じ
U4B	DWORD, UINT に同じ
F4B	float に同じ
F8B	double, REAL に同じ
BOOL	int, LONG に同じだが、非 0 / 0 で、真 / 偽を表す
pANY	PVOID に同じ
rANY	PCVOID に同じ
pSTR	PSTR に同じ
rSTR	PCSTR に同じ
pBIN	PBYTE に同じ
rBIN	PCBYTE に同じ
HFIL	HANDLE に同じ
BGRC	DWORD, UINT に同じだが、16進表記: 00bbggrr メモリ内: rrgbbb00 の色を表す、COLORREF 相当
RGBC	DWORD, UINT に同じだが、16進表記: 00rrggbb メモリ内: bbggrr00 の色を表す、RGBQUAD 相当
iANY	int, LONG に同じだが、ポインタとして使われることを示す、LONG_PTR, LRESULT, LPARAM 相当
uANY	DWORD, UINT に同じだが、ポインタとして使われることを示す、UINT_PTR, WPARAM 相当

4.3.4 既定の定数名

BolyaiのAPI関数で共通的に使用される定数名の定義を以下に示します。

```
#define FALSE 0
#define TRUE 1
#define RAND_MAX 32767
```

4.3.5 構造体型名の命名規則

ソースコードを簡潔にするため、API関数で用いられる構造体型名は、全て4文字程度の短縮名としています。また、構造体型名と同時に、構造体の配列を記述する構造体型名、およびそれぞれの構造体へのポインタ型（可変参照と不変参照）も一定の命名規則に基づいて宣言しています。

```
typedef struct {
    ...
} XYZI;
typedef XYZI *pXYZI, *pXYZI;
typedef const XYZI *rXYZI, *PCXYZI;

typedef struct {
    pXYZI pv;
    I4B ni;
} XYZS;
typedef XYZS *pXYZS, *pXYZS;
typedef const XYZS *rXYZS, *PCXYZS;
```

5 プログラミングガイド

5.1 ストアドプロシージャの概要

Bolyai-Cでプログラミングできるストアドプロシージャには、以下の3つの種類があります。

- (1) DB定義
- (2) 一括処理
- (3) 帳票作成

5.1.1 DB定義

DB定義プロシージャ（バイトコードのファイル名 **名前.BCF**）は、データベースの構造を定義するものです。このプロシージャは、ユーザーが「ファイル(F)」 「開く(O)」 コマンドでDB名を選択した際、および他のプロシージャ内で **ConnectDB** 関数が呼ばれた際にクライアントの主記憶にロードされ実行されます。

DB定義プロシージャは、データベースの構造を記述した定義データを与えて **ConnectDBX** 関数を呼ぶことでデータベースに接続します。（Bolyaiの流儀では、DB定義プロシージャのコンパイルが、サイトへのデータベースの新規登録を意味し、DB定義プロシージャの実行が、データベースへの接続を意味します。）

ソースコードには、プログラムの実行開始位置となるエン트리ポイント（以下「メイン関数」と呼びます。）として、次のような関数の定義を含めなければなりません。

```
BOOL ConnectMain(void) { ... }
```

DB定義プロシージャのプログラム例は、5.2節に示します。

DB定義プロシージャには、表操作に対応するデータベーストリガ（以下「表トリガ」と呼びます。）の定義を含めることができます。表トリガは、表に対して変更、挿入、削除等の操作が行われたときに自動的に実行されます。

表トリガプログラミングに関しては、5.5節で解説します。

5.1.2 一括処理

一括処理プロシージャ（バイトコードのファイル名 **名前.BCO**）は、データベースに対する一連の処理手順を登録して必要なときに呼び出せるようにしたものです。このプロシージャは、ユーザーが「処理(P)」 「一括処理(X)」 コマンドで処理名を選択した際にクライアントの主記憶にロードされ実行されます。（一括処理プロシージャを他のプロシージャから呼び出して実行することはできません。）

一括処理プロシージャとして記述できる処理内容には特に制限はありませんが、入出力の手段は以下の範囲に限られます。

- (1) 行編集用のダイアログボックスと各種メッセージボックス
- (2) サイト内のデータベースへのアクセス
- (3) 汎用のファイル入出力

画面上に表示できるのは、決められたメッセージボックスとダイアログボックスだけです。表ウィンドウを画面上に表示して操作することはできません。一括処理コマンドは、表ウィンドウを1つも開いていない状態（データベースに未接続の状態）でのみ実行できます。

ソースコードには、メイン関数として、次のような関数の定義を含めなければなりません。

```
void CommandMain(HWND hwnd) { ... }
```

一括処理プロシージャのプログラム例は、5.3節に示します。

5.1.3 帳票作成

帳票作成プロシージャ（バイトコードのファイル名 **名前.BCR**）は、自由な形式／内容で定義された帳票を、印刷プレビュー画面上で確認してプリンタに出力できるようにしたものです。このプロシージャは、ユーザーが「処理(P)」 「帳票作成(R)」 コマンドで帳票名を選択した際にクライアントの主記憶にロードされ実行されます。（帳票作成プロシージャを他のプロシージャから呼び出して実行することはできません。）

帳票作成プロシージャ内では、文字や図形を帳票のページ上に描画する関数を使用して手続き的に帳票内容を定義します。

ページ描画以外の入出力手段は一括処理と同じですが、帳票作成コマンドは閲覧ユーザーも実行できるため、データベースの内容を変更するような機能は使用しないようにします。

帳票作成コマンドは、表ウィンドウを1つも開いていない状態（データベースに未接続の状態）でのみ実行できます。ソースコードには、メイン関数として、次のような関数の定義を含めなければなりません。

```
BOOL ReportMain(HREP rp, I4B msg, I4B pg, uANY pa) { ... }
```

帳票作成プロシージャのプログラム例は、5.4節に示します。

5.2 DB定義プロシージャ

5.2.1 DB定義のプログラム例その1

最初に、一番簡単なプログラムの例を示します。

```
COLI ciTMP[1] = {
    { STR(64), 0, "cnam", NULL, NULL, CF_KEY, },
};
COLS csTMP = { ciTMP, NELEM(ciTMP) };

TBLI tiTMP[1] = {
    { &csTMP, NULL, "TNAM", },
};
TBLS tsTMP = { tiTMP, NELEM(tiTMP) };

DBBI dbTMP = { &tsTMP, "SAMPLE", };

BOOL ConnectMain(void)
{ // サンプル
    return ConnectDBX(&dbTMP);
}
```

このプログラムは、**cnam** という名前の64byte文字列項目 1 つだけからなる、**TNAM** という名前の表を 1 つだけ持つ、**SAMPLE** という名前のデータベースを定義しています。

COLI 構造体は、表の列項目の仕様を定義しています。(6.3.1参照)

COLS 構造体は、**COLI** 構造体の配列を記述しています。

TBLI 構造体は、表の仕様を定義しています。表の仕様は、列項目の仕様の配列 (**COLS** 構造体への参照) とその他の付加情報で記述されます。(6.3.2参照)

TBLS 構造体は、**TBLI** 構造体の配列を記述しています。

DBBI 構造体は、データベースの仕様を定義しています。データベースの仕様は、表の仕様の配列 (**TBLS** 構造体への参照) とその他の付加情報で記述されます。(6.3.6参照)

DB定義プロシージャは、**ConnectMain** というメイン関数の呼び出しから開始されます。DB定義プロシージャの役割は、**ConnectDBX** 関数にDB定義情報を渡してデータベースに接続することです。

5.2.2 DB定義のプログラム例その2

次に示すのは、複数の列項目からなる表を複数もつデータベースの定義例です。

```
BOOL fchkNUM(PCVOID pb, PCCOLI co)
{ // チェック制約
    int va = *(int *)VPtr(pb, co); return va >= 0 && va <= 9;
}

COLI ciNUM[9] = {
    { TID_I4B, ATR(2, ALN_CEN), "c1", "1", NULL, CF_KEY|CF_HZR, NULL, fchkNUM, },
    { TID_I4B, ATR(2, ALN_CEN), "c2", "2", NULL, CF_KEY|CF_HZR, NULL, fchkNUM, },
    { TID_I4B, ATR(2, ALN_CEN), "c3", "3", NULL, CF_KEY|CF_HZR, NULL, fchkNUM, },
    { TID_I4B, ATR(2, ALN_CEN), "c4", "4", NULL, CF_KEY|CF_HZR, NULL, fchkNUM, },
    { TID_I4B, ATR(2, ALN_CEN), "c5", "5", NULL, CF_KEY|CF_HZR, NULL, fchkNUM, },
    { TID_I4B, ATR(2, ALN_CEN), "c6", "6", NULL, CF_KEY|CF_HZR, NULL, fchkNUM, },
    { TID_I4B, ATR(2, ALN_CEN), "c7", "7", NULL, CF_KEY|CF_HZR, NULL, fchkNUM, },
    { TID_I4B, ATR(2, ALN_CEN), "c8", "8", NULL, CF_KEY|CF_HZR, NULL, fchkNUM, },
    { TID_I4B, ATR(2, ALN_CEN), "c9", "9", NULL, CF_KEY|CF_HZR, NULL, fchkNUM, },
};
COLS csNUM = { ciNUM, NELEM(ciNUM) };

TBLI tiNUM[2] = {
    { &csNUM, NULL, "MON", "問題表", },
    { &csNUM, NULL, "KAI", "解答表", },
};
```

```

TBLS tsNUM = { tiNUM, NELEM(tiNUM) };

BDBI dbNUM = { &tsNUM, "SAMPLE03A", "ナンプレ", "説明用サンプル" };

BOOL ConnectMain(void)
{ // ナンプレ問題／解答
  int re;

  re = ConnectDBX(&dbNUM);
  if (re) NewDBFiles();
  return re;
}

```

このプログラムは、「ナンバープレース」と呼ばれる数字パズルを解くプログラムで使用するための表を定義しています。ナンプレの問題（と解答）は、ちょうど9×9の行列の形をしているので、整数値の項目が9列ある表で表現することが可能です。問題用の表と解答用の表は仕様が同じなので、列項目情報を共有しています。

実際に問題を解くプログラムは、一括処理プロシージャとして別途作成します。（5.3.3参照）

ATR(2, ALN_CEN) は、列幅と文字揃えコード（2桁中央揃え）を指定しています。

fchkNUM は、入力できる値を制限するためのチェック制約を与えるコールバック関数です。

VPtr 関数は、コールバック関数内で項目のアドレスを取得するものです。

NewDBFiles 関数は、データベースを構成する表ファイルがまだ存在していない場合、空の表ファイルを自動的に作成するものです。（データベースを定義しただけではサイト内に表ファイルは作成されません。）

※注意※ 実運用段階のプログラムに NewDBFiles 関数を残しておくのは安全ではありません。事故で表ファイルが失われた際にエラーが出ないため発見が遅れる可能性があるためです。

5.2.3 DB定義のプログラム例その3

ある程度実用的なデータベースの定義例を次に示します。

```

// リース等契約区分
PCSTR SLTP[3] = { "リース", "レンタル", "買取" };
ENMS esLTP = { SLTP, NELEM(SLTP) };

// 事業所
CODI diOFC[6] = {
  { 19, "本社" }, { 20, "仙台" }, { 23, "名古屋" }, { 24, "大阪" },
  { 26, "広島" }, { 27, "九州" }
};
CODS dsOFC = { diOFC, NELEM(diOFC) };

// リース等リスト
COLI ciLEA[14] = {
  { ASC(10), 9, "lid", "リース等 I D", NULL, CF_KEY, },
  { TID_E1B, 6, "ltp", "区分", "契約区分", CF_DDL, NULL, NULL, (UINT)&esLTP, },
  { TID_C1B, 0, "ofc", "事業所", NULL, CF_DDL, NULL, NULL, (UINT)&dsOFC, },
  { STR(40), 0, "dsc", "物件名", },
  { TID_I4B, 4, "ter", "期間", "期間(月数)", },
  { TID_YMD, 0, "bdt", "開始日", },
  { TID_YMD, 0, "edt", "終了日", },
  { TID_M4B, 10, "mnp", "月払額", },
  { TID_M4B, 10, "amp", "物件価格", },
  { STR(24), 0, "ven", "販売会社名", },
  { STR(24), 0, "len", "リース等会社名", },
  { STR(40), 20, "lcd", "契約コード", },
  { STR(40), 0, "rem", "備考", },
  { STR(12), 8, "fil", "ファイル", },
};
COLS csLEA = { ciLEA, NELEM(ciLEA) };

```

```

// 保守契約区分
PCSTR SMTP[3] = { "月払", "年払", "その他" };
ENMS esMTP = { SMTP, NELEM(SMTP) };

// 保守契約リスト
COLI ciMNT[12] = {
    { ASC(10), 9, "mid", "保守 I D", NULL, CF_KEY, },
    { TID_E1B, 6, "mtp", "区分", "契約区分", CF_DDL, NULL, NULL, (UINT)&esMTP, },
    { TID_C1B, 0, "ofc", "事業所", NULL, CF_DDL, NULL, NULL, (UINT)&dsOFC, },
    { STR(40), 0, "dsc", "物件名", },
    { TID_I4B, 4, "ter", "期間", "期間(月数)", },
    { TID_YMD, 0, "bdt", "開始日", },
    { TID_YMD, 0, "edt", "終了日", },
    { TID_M4B, 10, "mnp", "月払額", },
    { STR(24), 0, "ven", "保守会社名", },
    { STR(40), 20, "mcd", "保守コード", },
    { STR(40), 0, "rem", "備考", },
    { STR(12), 8, "fil", "ファイル", },
};
COLS csMNT = { ciMNT, NELEM(ciMNT) };

// 品目
PCSTR SCAT[10] = {
    "", "パソコン", "ディスプレイ", "大判プリンタ", "コピー機", "プリンタ",
    "サーバ等", "その他機器", "ソフト等", "I P 割当"
};
ENMS esCAT = { SCAT, NELEM(SCAT) };

// 状態
PCSTR SSTA[5] = { "現用", "予備", "故障", "撤去済", "未配備" };
ENMS esSTA = { SSTA, NELEM(SSTA) };

// 備品リスト
COLI ciEQU[15] = {
    { ASC(10), 9, "qid", "備品 I D", NULL, CF_KEY, },
    { TID_E1B, 10, "cat", "品目", "品 目", CF_DDL, NULL, NULL, (UINT)&esCAT, },
    { TID_C1B, 0, "ofc", "事業所", NULL, CF_DDL, NULL, NULL, (UINT)&dsOFC, },
    { TID_E1B, 0, "sta", "状態", "状 態", CF_DDL, NULL, NULL, (UINT)&esSTA, },
    { STR(23), 12, "mak", "メーカー名", },
    { STR(24), 16, "mdl", "製品名／型番", },
    { STR(24), 16, "snm", "製造番号等", },
    { STR(24), 16, "spc", "スペック等", },
    { STR(40), 0, "use", "主要用途等", },
    { STR(10), 9, "lid", "リース等 I D", },
    { STR(10), 9, "mid", "保守 I D", },
    { STR(40), 16, "qcd", "個体保守コード", },
    { STR(16), 0, "ipa", "I P アドレス", },
    { TID_YMD, 0, "ddt", "個体撤去日", },
    { STR(44), 0, "rem", "備考", },
};
COLS csEQU = { ciEQU, NELEM(ciEQU) };

TBLI tiBIH[3] = {
    { &csLEA, NULL, "LEADAT", "リース等リスト", NULL, },
    { &csMNT, NULL, "MNTDAT", "保守契約リスト", NULL, },
    { &csEQU, NULL, "EQU DAT", "備品リスト", NULL, },
};
TBLS tsBIH = { tiBIH, NELEM(tiBIH) };

```



```
BDBI dbBIH = { &tsBIH, "SAMPLE04A", "備品管理", "説明用サンプル" };
```

```
BOOL ConnectMain(void)
{ // サンプル
  return ConnectDBX(&dbBIH);
}
```

このプログラムは、パソコン等のOA機器類を対象とした備品管理データベースを定義しています。備品管理データベースは、以下の前提条件があるものとして設計されています。

- ・複数の事業所（東京、仙台、名古屋、大阪、広島、九州）の備品をまとめて管理したい。
- ・備品は、リースで導入することが多いが、買取やレンタルの場合もある。
- ・備品には、保守契約を付けることがある。
- ・備品の配備状況だけでなく、月間や年間のコスト（リース料、保守料、買取金額）も概算できるようにしたい。

データベースは、3つの表で構成され、それぞれ以下のように使用します。

(1) リース等リスト

- ・リース契約等（レンタル、買取を含む）1件ごとにIDコードを付けて管理する。
- ・買取の場合、買取日を「開始日」にし、「期間」「月払額」は使用しないものとする。
- ・リース／レンタルの場合、「期間」は当初予定の期間とし、「終了日」は実際の返却日とする。
- ・途中解約や再リース延長の場合、終了日のみ変更し、期間と月払額は変更しないものとする。

(2) 保守契約リスト

- ・保守契約1件ごとにIDコードを付けて管理する。
- ・「期間」は当初予定の期間とし、「終了日」は実際の解約日とする。
- ・途中解約や契約延長の場合、終了日のみ変更し、期間は変更しないものとする。

(3) 備品リスト

- ・備品機器1台ごとにIDコードを付けて管理する。
- ・「個体保守コード」は、保守契約の形態によっては製造番号とは別に1台ごとに保守サービス用のコードを割り当てられることがあるため、その場合に使用する。
- ・「IPアドレス」は、ネットワーク接続機器の管理のために使用する。
- ・「個体撤去日」は、リース契約等の一部のみを途中解約する場合等の記録のために使用する。

5.2.4 DB定義のプログラム例その4

前節に示したコード（SAMPLE04A.C）は、最小限の機能のみを実装した簡易版です。実用的なものとするには、さらに以下のような機能を加える必要があります。

- (1) 必須入力項目等のチェック事項を追加する
- (2) 計算項目を追加して、備品リストに、リース等リスト、保守契約リストの情報を結合して表示できるようにする
- (3) データの更新履歴（誰が、いつ何を編集したかの記録）を残せるようにする
- (4) 各事業所の担当者が入力する際に、他の事業所のデータを編集できないようにする

これらの機能を実装した完全版は、SAMPLE フォルダ内の SAMPLE04B.C で見ることができます。

(3)(4)の機能の実装には、表トリガを利用しています。プログラム例は、5.5.1節と5.5.2節でそれぞれ扱います。

SAMPLE04B.C のDB定義は、SAMPLE04A.C のDB定義とデータの物理的レイアウトが等しいため、SAMPLE04A.C で作成した表データファイルをそのまま引き継いで使用することができます。このようにデータの物理的レイアウトを変えない範囲のDB定義の変更は、いつでも柔軟に行うことができます。一方、データの物理的レイアウトが変更されるときは、DB定義の変更に伴い表データファイルの再編成が必要になります。その場合は、再編成用の一括処理プロシージャを作成して実行するか、テキストファイル経由でデータを移行します。テキストファイル経由でのデータ移行手順は次のようなものです。

- (1) 最初に元の表から「テキスト書出(E)」でテキストファイルに全実表データを書き出しておく
- (2) 表の全行を削除して保存する（表データファイルを空にする）
- (3) DB定義プロシージャを新版に差し替える
- (4) 空の表に「テキスト取込(I)」でテキストファイルからデータを取り込む

5.3 一括処理プロセス

5.3.1 一括処理のプログラム例その1

最初に、一番簡単なプログラムの例を示します。

```
void CommandMain(HWND hwnd)
{ // サンプル
  OkBox(hwnd, "hello, world", NULL);
}
```

このプログラムは、画面に「hello, world」というメッセージボックスを表示するものです。一括処理プロセスは、**CommandMain** というメイン関数の呼び出しから開始されます。引数の **hwnd** は、フレームウィンドウのハンドルです。**hwnd** はメッセージボックスやダイアログボックスを表示する際に親ウィンドウのハンドルとして関数に渡します。**OkBox** 関数は「OK」ボタンを1つだけ持ったメッセージボックスを表示します。

5.3.2 一括処理のプログラム例その2

次に、ユーザーからの入力を受け取って結果を返すプログラムの例を示します。

```
void CommandMain(HWND hwnd)
{ // 占い
  int sr, re;
  HBUF rb;
  char msg[64];
  static char un[6][6] = { "大吉", "中吉", "小吉", "吉", "末吉", "凶" };
  static COLI ci[1] = { { TID_YMD, 0, "sr", "生年月日", } };
  static COLS cs = { ci, NELEM(ci) };
  static TBLI ti = { &cs, };

  sr = today();
  rb = CreateRowBufX(&sr, &ti);
  re = EditRowDlgBox(hwnd, rb, "占い");
  DeleteRowBuf(rb);
  if (!re) return;
  sprintf(msg, "あなたの本日の運勢は、「%s」です。", un[(sr + today()) % 6]);
  OkBox(hwnd, msg, NULL);
}
```

このプログラムは、ユーザーに生年月日を聞いて、その日の運勢（「大吉」～「凶」）を占うものです。ユーザーからの入力を受け取るのは、**EditRowDlgBox** 関数（ダイアログボックス上で表の1行分のデータを編集する関数）です。この関数は、データの受け渡しに1行分の行バッファを使用します。行バッファオブジェクトは、**CreateRowBufX** 関数で作成し、使用後に **DeleteRowBuf** 関数で削除します。

5.3.3 一括処理のプログラム例その3

それでは、実際にデータベースにアクセスするプログラムの例を見てみましょう。以下に示すのは、5.2.2節で定義した SAMPLE03A データベースにアクセスして「ナンバープレース」の問題を解くプログラムです。

```
BOOL load_p(PINT p, PCSTR nam)
{ // 問題読込
    int i, j;
    HOPN op; HTBL tb;

    op = OpenTable(nam, OM_RDO); if (!op) return FALSE;
    tb = OHTBL(op);
    for (i = 0; i < 9; i++) {
        for (j = 0; j < 9; j++) {
            GetTableCellValue(&p[9*i + j], tb, i, j);
        }
    }
    return CloseTable(op, FALSE);
}

BOOL save_p(PINT p, PCSTR nam)
{ // 解答書出
    int i, j;
    HOPN op; HTBL tb;

    op = OpenTable(nam, OM_RDW); if (!op) return FALSE;
    tb = OHTBL(op);
    DeleteTableRows(tb, 0, NTableRows(tb));
    InsertTableRows(tb, 0, 9);
    for (i = 0; i < 9; i++) {
        for (j = 0; j < 9; j++) {
            SetTableCellValue(tb, i, j, &p[9*i + j]);
        }
    }
    return CloseTable(op, TRUE);
}

void CommandMain(HWND hwnd)
{ // ナンプレソルバー
    int re, p[9*9];

    if (!ConnectDB("SAMPLE03A")) {
        MessageBox(hwnd, "データベースに接続できません。", NULL); return;
    }
    if (!load_p(p, "MON")) {
        MessageBox(hwnd, "問題の読込に失敗しました。", NULL); DisconnectDB(); return;
    }
    BusyMsg(hwnd, "計算中です。");
    re = solve(p);
    EndBusy(hwnd);
    if (!re) {
        OkBox(hwnd, "解けません。", NULL);
    } else if (save_p(p, "KAI")) {
        OkBox(hwnd, "解けました。", NULL);
    } else {
        MessageBox(hwnd, "解答の書出に失敗しました。", NULL);
    }
    DisconnectDB();
}
```

問題（と解答）データは、`p[9*9]` という `int` の配列に格納されます。

実際に問題を解いているのは、`solve` という関数です。`solve` の関数定義を含めた完全なソースコードは、ここでは省略していますが、配布メディアの `SAMPLE` フォルダ内の `SAMPLE03B.C` で見ることができます。

プロシージャ内でデータベースにアクセスする際の処理手順は、

- (1) `ConnectDB` でデータベースに接続
 - (2) `OpenTable` で表ファイルを開く
 - (3) `GetTableCellValue` / `SetTableCellValue` で表からデータを読み出す／表にデータを書き込む
 - (4) `CloseTable` で表ファイルを閉じる（変更を保存する）
 - (5) `DisconnectDB` でデータベースへの接続を解除
- という流れになります。

データベースに接続した時点で、クライアントの主記憶内には、そのデータベースを構成する全ての表に対する表オブジェクトが作成され、それぞれの表オブジェクトを参照する表ハンドルが有効になります。ただし、この時点ではデータはまだロードされていないので、各表オブジェクトは空のままです。表オブジェクトにファイルからデータを読み込むのが `OpenTable` 関数、表オブジェクトからファイルにデータを書き出すのが `CloseTable` 関数です。

`OpenTable` 関数は、成功すると表オープン情報のハンドル（`HOPN` 型）を返します。

`OHTBL` 関数は、表オープン情報ハンドルから表ハンドル（`HTBL` 型）を取得します。

`DeleteTableRows(tb, 0, NTableRows(tb))` は、表を空にする（全行削除する）ための定型句です。

`InsertTableRows(tb, 0, 9)` は、空行を 9 行挿入します。

`GetTableCellValue` / `SetTableCellValue` 関数は、表から項目の値を取得／表の項目に値を設定します。

時間のかかりそうな処理は、`BusyMsg` 関数と `EndBusy` 関数で囲んでおくと、その間ステータスバーにメッセージが表示され、マウ斯卡ーソルが待機状態になります。

5.3.4 一括処理のプログラム例その4

Bolyaiの表データファイルは、固定長のレコードを順編成しているだけなので、データベースへのアクセス関数を使用しなくても、汎用のファイル入出力関数のみで表ファイルからの読込／表ファイルへの書出ができます。前節のプログラムを汎用のファイル入出力関数で書き換えた版を以下に示します。

```
BOOL load_p(PINT p, PCSTR nam)
{ // 問題読込
    int re;
    HANDLE hf;

    if (! (hf = open_file(nam, OP_RDO))) return FALSE;
    re = read_file(hf, p, 9 * 9 * sizeof(int));
    close_file(hf);
    return re;
}

BOOL save_p(PINT p, PCSTR nam)
{ // 解答書出
    int re;
    HANDLE hf;

    if (! (hf = open_file(nam, OP_WRI))) return FALSE;
    re = write_file(hf, p, 9 * 9 * sizeof(int));
    close_file(hf);
    return re;
}

void CommandMain(HWND hwnd)
{ // ナンプレソルバー
    int re, p[9*9];

    if (!load_p(p, "MON.BTB")) {
        ErrorBox(hwnd, "問題の読込に失敗しました。", NULL); return;
    }
    BusyMsg(hwnd, "計算中です。");
    re = solve(p);
    EndBusy(hwnd);
    if (!re) {
        OkBox(hwnd, "解けません。", NULL);
    } else if (save_p(p, "KAI.BTB")) {
        OkBox(hwnd, "解けました。", NULL);
    } else {
        ErrorBox(hwnd, "解答の書出に失敗しました。", NULL);
    }
}
```

open_file／close_file／read_file／write_file は、それぞれWindowsのAPI関数 CreateFile／CloseHandle／ReadFile／WriteFile のラッパー関数です。

5.4 帳票作成プロシージャ

5.4.1 帳票作成のプログラム例その1

最初に、一番簡単なプログラムの例を示します。

```
BOOL ReportMain(HREP rp, I4B msg, I4B pg, uANY pa)
{ // サンプル
  static CELL cel = { 20.0, 80.0, 190.0, 100.0 };

  SetTextFont(rp, NULL, 15.0, 0, 0);
  CellText(rp, &cel, "hello, world", DT_CEN);
  return TRUE;
}
```

このプログラムは、A4用紙の上部中央に大きく「hello, world」と印字するものです。

帳票作成プロシージャは、**ReportMain** というメイン関数の呼び出しから開始されます。引数の **rp** は、レポート情報（帳票作成の環境を管理するオブジェクト）のハンドルです。描画関数を始めほとんどの帳票作成関数は、現在のコンテキストを示す **rp** を引数に与えて呼び出します。

CELL 構造体は、用紙上の矩形領域の座標（左上隅 **x**、**y** 座標、右下隅 **x**、**y** 座標）を保持します。座標系は、用紙左上隅が原点で、単位はmm、座標値は、実数（**REAL** 型）で表現します。

SetTextFont 関数は、文字のフォントを設定します。

CellText 関数は、矩形領域に文字列を描画します。

上の例では使用されていませんが、引数の **msg** は **ReportMain** 関数の呼び出しフェーズを、**pg** はページ番号を、**pa** は任意の付加情報を表します。

ReportMain 関数は、以下のフェーズごとに呼び出されます。

- (1) **REPM_CREATE** レポート情報作成の直後
- (2) **REPM_SETUP** ユーザー入力のタイミング
- (3) **REPM_PAGESETUP** 印刷用紙設定ダイアログボックスで「OK」が押された直後
- (4) **REPM_STARTPRINT** 印刷の開始
- (5) **REPM_DRAWPAGE** ページの描画
- (6) **REPM_SKIPPAGE** ページのスキップ
- (7) **REPM_ENDPRINT** 印刷の終了
- (8) **REPM_DESTROY** レポート情報削除の直前

このため、**ReportMain** 関数内では、**msg** の値によって処理を分岐する必要があります。上の例では **msg** による分岐をしていませんが、**CellText** 関数等の描画関数は、**DRAWPAGE** フェーズで呼ばれない限り何もしない仕様になっているので、この場合は問題ありません。

ReportMain 関数の戻り値が意味を持つのは、**CREATE** / **SETUP** / **PAGESETUP** フェーズのみで、**FALSE** を返すと印刷の取りやめを意味します。それ以外の場合は常に **TRUE** を返すようにします。

複数のページからなる帳票の場合、印刷（プレビュー画面上での描画を含む）のつどページごとに **DRAWPAGE** または **SKIPPAGE** フェーズで **ReportMain** 関数が呼び出されます。ページ数は、**PAGESETUP** フェーズまでに **SetNPages** 関数で与えます。（ページ数の設定を省略すると単一ページの帳票になります。）

CREATE フェーズで印刷を取りやめた場合、**DESTROY** フェーズの呼び出しはありませんが、**SETUP** / **PAGESETUP** フェーズで印刷を取りやめた場合、後始末のため **DESTROY** フェーズの呼び出しがあります。

5.4.2 帳票作成のプログラム例その2

次に示すのは、カレンダー作成プログラムです。

```
BOOL ReportMain(HREP rp, I4B msg, I4B pg, uANY pa)
{ // カレンダー
  int i, j, sr, yo, dm, dy, tco, re;
  CELL cel;
  GRID grd;
  HBUF rb;
  static int sr0;
  static COLI ciCAL = { TID_YMN, 0, "sr0", NULL, "開始年月(&M)", };
  static COLS csCAL = { &ciCAL, 1 };
  static TBLI tiCAL = { &csCAL, };
  static BGRC mc[12] = {
    0x004488, 0x006666, 0x008844, 0x00cc00, 0x448800, 0x666600,
```

```

    0x884400, 0xcc0000, 0x880044, 0x660066, 0x440088, 0x0000cc,
};
static char wk[7][4] = { "SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT" };

if (msg == REPM_SETUP) {

    sr0 = date_serial(year(today()), month(today()), 1);
    rb = CreateRowBufX(&sr0, &tiCAL);
    re = EditRowDlgBox(hWnd(rp), rb, "カレンダー");
    DeleteRowBuf(rb);
    if (!re) return FALSE;

} else if (msg == REPM_PAGESETUP) {

    SetNPages(rp, 12);

} else if (msg == REPM_DRAWPAGE) {

    sr = date_serial(year(sr0), month(sr0) + pg - 1, 1);
    SetCell(&cel, 80.0, 40.0, 130.0, 90.0);
    FillCell(rp, &cel, RST_NUL, mc[month(sr) - 1]);
    tco = 0xffffffff;
    SetCell(&cel, 80.0, 45.0, 130.0, 55.0);
    SetTextFont(rp, NULL, 9.0, 0, tco);
    CellText(rp, &cel, format("%d", year(sr)), DT_CEN);
    SetCell(&cel, 80.0, 55.0, 130.0, 85.0);
    SetTextFont(rp, NULL, 30.0, 0, tco);
    CellText(rp, &cel, format("%d", month(sr)), DT_CEN);
    SetGrid0(&grd, 21.0, 100.0, 7, 6, 24.0, 20.0);
    yo = youbi(sr);
    dm = days_of_month(year(sr), month(sr));
    dy = 0;
    for (i = 0; i < grd.nr; i++) {
        for (j = 0; j < grd.nc; j++) {
            tco = (j == 0)? 0x0000ff: (j == 6)? 0xff0000: 0;
            if (i == 0) {
                SetTextFont(rp, NULL, 8.0, 0, tco);
                GridCellText(rp, &grd, i, j, RST_R_M, wk[j], DT_CEN);
            } else {
                SetTextFont(rp, NULL, 12.0, 0, tco);
                if (i == 1 && j == yo) dy = 1;
                if (dy > 0 && dy <= dm) {
                    if (i == grd.nr - 1 && dy + 7 <= dm) {
                        SetTextFont(rp, NULL, 10.0, 0, tco);
                        GridCellText(rp, &grd, i, j, RST_R_M, "／", DT_CEN);
                        MoveCell(GridCell(&cel, &grd, i, j), 1.0, -5.0);
                        CellText(rp, &cel, format("%d", dy), DT_LFT);
                        MoveCell(GridCell(&cel, &grd, i, j), -1.0, 5.0);
                        CellText(rp, &cel, format("%d", dy + 7), DT_RGT);
                    } else {
                        GridCellText(rp, &grd, i, j, RST_R_M, format("%d", dy), DT_CEN);
                    }
                    dy++;
                } else {
                    GridCellText(rp, &grd, i, j, RST_R_M, "*", DT_CEN);
                }
            }
        }
    }
}
}

```

```

    }
    return TRUE;
}

```

このプログラムは、ユーザーの指定する開始年月から始まる12ヵ月分のカレンダーを12ページ分の帳票として印刷するものです。

GRID 構造体は、用紙上にグリッドを設定します。グリッドには、縦横とも等間隔のグリッド、縦は等間隔で横は任意間隔のグリッド、縦は任意間隔で横は等間隔のグリッド、縦横とも任意間隔のグリッドの4つの種類があり、それぞれ次の関数で設定します。

```

void SetGrid0(PGRID gr, REAL x0, REAL y0, int nc, int nr, REAL dcw, REAL drh);
void SetGrid1(PGRID gr, REAL x0, REAL y0, int nc, int nr, REAL *cw, REAL drh);
void SetGrid2(PGRID gr, REAL x0, REAL y0, int nc, int nr, REAL dcw, REAL *rh);
void SetGrid3(PGRID gr, REAL x0, REAL y0, int nc, int nr, REAL *cw, REAL *rh);

```

ここでは縦横とも等間隔のグリッドを使用しています。

hWND 関数は、レポート情報ハンドルから親ウィンドウのハンドルを取得します。

FillCell 関数は、矩形領域を塗りつぶします。

SetTextFont 関数の第5引数は、文字の色を指定します。

GridCellText 関数は、グリッド内のセル矩形に罫線と文字列を描画します。

GridCell 関数は、グリッドからセル矩形を取り出します。

MoveCell 関数は、セル矩形を移動します。

このカレンダープログラムには、祝日の表示機能がありません。祝日を表示するためには、祝日のリストをそのソースコード内に埋め込むか、祝日のリストを表として持つデータベースを作成し、そのデータベースにアクセスする機能を実装する必要があります。

5.4.3 帳票作成のプログラム例その3

実際にデータベースにアクセスするプログラムの例は、配布メディアのSAMPLE フォルダ内のSAMPLE04C.Cで見ることができます。このプログラムは、5.2.3節で作成した備品管理データベースを用いて、年度ごとのコスト推移をレポートするものです。

以下に抜粋するのは、そのコードの一部ですが、メモリ内の表データを構造体の配列とみなしてアクセスしています。

```

// リース等データ
typedef struct {
    char lid[10];
    BYTE ltp, ofc;
    char dsc[40];
    int ter;
    int bdt, edt;
    int mnp;
    int amp;
    char ven[24];
    char len[24];
    char lcd[40];
    char rem[40];
    char fil[12];
} LEADAT;

```

```

// 保守契約データ
typedef struct {
    char mid[10];
    BYTE mtp, ofc;
    char dsc[40];
    int ter;
    int bdt, edt;
    int mnp;
    char ven[24];
    char mcd[40];
    char rem[40];
    char fil[12];
} MNTDAT;

```



```

void MonthlyCost(REAL *mc, int ym, int ofc)
{ // 月間コスト
  HTBL tb;
  LEADAT *p0, *p0e;
  MNTDAT *p1, *p1e;

  mc[0] = mc[1] = mc[2] = 0.0;
  tb = DBHTBL("LEADAT");
  for (p0 = (LEADAT *)TablePtr(tb), p0e = p0 + NTableRows(tb); p0 < p0e; p0++) {
    if (ofc == 0 || p0->ofc == ofc) {
      switch (p0->ltp) {
        case 0:
          if (ym >= p0->bdt) {
            if (ym <= date_serial(year(p0->bdt), month(p0->bdt) + p0->ter, 0)) {
              mc[0] += p0->mnp;
            } else if (ym <= p0->edt) {
              mc[0] += iround(p0->mnp / 10.0);
            }
          }
          break;
        case 1:
          if (ym >= p0->bdt && ym <= p0->edt) {
            mc[0] += p0->mnp;
          }
          break;
        case 2:
          if (ym == date_serial(year(p0->bdt), month(p0->bdt), 1)) {
            mc[2] += p0->amp;
          }
          break;
      }
    }
  }
  tb = DBHTBL("MNTDAT");
  for (p1 = (MNTDAT *)TablePtr(tb), p1e = p1 + NTableRows(tb); p1 < p1e; p1++) {
    if (ofc == 0 || p1->ofc == ofc) {
      if (ym >= p1->bdt && ym <= p1->edt) {
        mc[1] += p1->mnp;
      }
    }
  }
}

```

DBHTBL 関数は、表名から表ハンドルを取得する関数です。

TablePtr 関数は、表データ（構造体の配列）の先頭アドレスを返します。

date_serial, year, month 関数は、日付データを操作します。（6.2.4参照）

5.5 表トリガ

表トリガは、表に対して変更、挿入、削除等の操作が行われた際に、自動的に呼び出されるストアドプロシージャです。表トリガを使用すると、表データに対する操作を一定の条件を満たすように限定したり、操作の履歴を残したりすることができます。

表に対して表トリガを登録しておく、その表に対する行単位（または項目単位）の操作イベントごとに表トリガが呼び出されます。

表トリガは、DB定義プロシージャ内で、次の宣言で表される型のコールバック関数として記述し、そのアドレスを TBLI 構造体の `ftrg` メンバに与えることで登録します。

```
typedef BOOL (*FTRG) (HTBL tb, I4B msg, I4B ir, pANY pb);
```

引数の `tb` は表ハンドルで、`msg` は表トリガの呼び出しフェーズを、`ir` は行インデックスを、`pb` は行の先頭アドレスを表します。`pb` は、一部例外（下記(7)と(9)）を除き `TableRow(tb, ir)` と一致します。

表トリガの呼び出しフェーズは、以下の通りです。

- | | |
|-----------------------------------|-------------------------|
| (1) TRGM_BEFOREUPDATEOF | 行内の項目の変更前 |
| (2) TRGM_AFTERUPDATEOF | 行内の項目の変更後 |
| (3) TRGM_BEFOREUPDATE/TRGM_CHECK | 行全体の変更前／行のチェック |
| (4) TRGM_AFTERUPDATE | 行全体の変更後 |
| (5) TRGM_DEFAULT/TRGM_AFTERINSERT | 行の初期化／行の挿入後 |
| (6) TRGM_BEFOREDELETE | 行の削除前 |
| (7) TRGM_UPDATERECORD | 表の保存前（読込時以降に変更された行について） |
| (8) TRGM_INSERTRECORD | 表の保存前（読込時以降に挿入された行について） |
| (9) TRGM_DELETERECORD | 表の保存前（読込時以降に削除された行について） |

変更は、項目単位の場合と行単位の場合があり、それぞれについて変更前と変更後に、挿入に対しては、挿入後に、削除に対しては、削除前に、表トリガが呼び出されます。また、これらとは別に、行のチェックまたは行の初期化の際にも表トリガが呼ばれますが、行全体の変更前にはその行のチェックが必要であり、行の挿入後にはその行の初期化が必要なため、(3)と(5)にそれぞれ2つある名前は同一の機能を表すシノニムになります。

(1)～(6)は、全てクライアントのメモリ内での表データの操作イベントです。したがってこれらのイベントを監視しても、実際にデータベースに変更が生じたかどうかわかりません。（保存がキャンセルされる、または保存に失敗する可能性があります。）このため、実際にデータを保存する直前に変更行、挿入行、削除行それぞれについてもう一度表トリガ(7)～(9)を呼び出すようになっています。この段階では操作の取り消しはできませんが、操作の履歴を残す場合は、この呼び出しフェーズを利用します。

表トリガの戻り値が意味を持つのは、`BEFORUPDATEOF` / `BEFOREUPDATE` / `BEFOREDELETE` フェーズの場合で、変更または削除の可否を `TRUE` / `FALSE` で示します。`BEFOREUPDATEOF` / `BEFOREUPDATE` フェーズの `pb` は、変更後の行内容をいわば仮置きした状態で示しています。一方、`UPDATERECORD` / `DELETERECORD` フェーズの `pb` は、保存の実行で失われる変更または削除前の行内容を示しています。

`BEFOREUPDATEOF` / `AFTERUPDATEOF` フェーズの `msg` は、上位ワードが列インデックスとなります。すなわち、

```
MAKEUINT(TRGM_BEFOREUPDATEOF, ic)
```

```
MAKEUINT(TRGM_AFTERUPDATEOF, ic)
```

で項目に対する変更フェーズを表します。

それぞれの表トリガを呼び出すデータベース操作関数は、次表の通りです。

(1) TRGM_BEFOREUPDATEOF (2) TRGM_AFTERUPDATEOF	SetTableCellValue, ScanTableCellValue, SetViewCellValue, ScanViewCellValue, ScanViewCellValueF
(3) TRGM_BEFOREUPDATE/TRGM_CHECK (4) TRGM_AFTERUPDATE	ClearTableRow, ClearTableRows, ClearViewRow, ClearViewRows, SetTableRow, SetNewTableRow, CheckTable, ReadCSV
(5) TRGM_DEFAULT/TRGM_AFTERINSERT	ClearTableRow, ClearTableRows, InsertTableRow, InsertTableRows, AddTableRow, NewTableRow, ClearViewRow, ClearViewRows, InsertViewRow, InsertViewRows, AddViewRow, NewViewRow, etc.
(6) TRGM_BEFOREDELETE	DeleteTableRow, DeleteTableRows, DeleteViewRow, DeleteViewRows
(7) TRGM_UPDATERECORD (8) TRGM_INSERTRECORD (9) TRGM_DELETERECORD	SaveTable, SaveTables, CloseTable, CloseTables

5.5.1 表トリガのプログラム例その1

それでは、操作履歴を残す表トリガの例を見てみましょう。次の表トリガプログラムは、5.2.3節で作成した備品管理データベース上で動作します。

```
BOOL ftrgBIH(HTBL tb, int msg, int ir, PVOID pb)
{ // 備品管理操作履歴
  HANDLE hf;
  char bf[256];
  static char opr[3][6] = { "変更 ", "挿入 ", "削除 " };

  switch (msg) {
  case TRGM_UPDATERECORD:
  case TRGM_INSERTRECORD:
  case TRGM_DELETERECORD:
    strcpy(bf, dtformat("yyyy/mm/dd hh:mm:ss ", now()));
    strcat(bf, opr[msg - TRGM_UPDATERECORD]);
    strcat(bf, format("%s ", (PCSTR)pb));
    strcat(bf, CurUser());
    hf = open_file(strconc(TableName(tb), ".LOG"), OP_WRI);
    if (!hf) break;
    set_file_ptr(hf, 0, FILE_END);
    write_line(hf, bf);
    close_file(hf);
    break;
  }
  return DefaultTableTrigger(tb, msg, ir, pb);
}
```

このプログラムは、サイトのフォルダ内の **表名.LOG** という名前のテキストファイルに操作履歴を書き出します。書き出されるのは、操作日時、操作内容（変更／挿入／削除）、操作された行の主キーの値、操作したユーザー名です。

2021/10/01 14:58:54 変更 PS0019 T.Yamada

DefaultTableTrigger 関数は、表トリガが登録されていない表に対するデフォルトの処理関数です。非ゼロ制約／一意性制約／主キー制約／外部キー制約／チェック制約の検査と行の初期化は、この関数が行っています。表トリガ内で独自の処理をする場合を除き、受け取った引数は、全てこの関数に渡さなければなりません。

5.5.2 表トリガのプログラム例その2

次に示すのは、表トリガを利用してユーザーからのアクセスをコントロールするプログラムです。5.2.3節で作成した備品管理データベースで、各事業所の担当者が他の事業所のデータを編集できないようにするにはどうすればよいと考えます。この機能を実現するためには、ログイン中のユーザーの所属事業所がわかるようになっていなければなりません。ユーザー名は CurUser 関数で取得できるので、ユーザー名からその所属事業所コードを得る uso という関数を用意します。

```
BYTE uso(PCSTR usr)
{ // ユーザーの所属事業所
  BYTE ofc;

  if (strcmp(usr, "T.Yamada") == 0) ofc = 19;
  else if (strcmp(usr, "A.Hirose") == 0) ofc = 20;
  else if (strcmp(usr, "S.Wakamiya") == 0) ofc = 23;
  else if (strcmp(usr, "K.Nanba") == 0) ofc = 24;
  else if (strcmp(usr, "K.Kamiya") == 0) ofc = 26;
  else if (strcmp(usr, "F.Nakagawa") == 0) ofc = 27;
  else ofc = 0;
  return ofc;
}
```

表トリガ関数内では、行の初期化時に事業所欄に自動的にユーザーの所属事業所コードが入力されるようにした上で、

D B 管理者権限のないユーザー（権限レベルは **CurAuth** 関数で取得します。）が、事業所欄を変更できないようにし、かつ事業所欄の値が所属事業所と違う行を変更または削除できないようにすれば、所望の機能が実現します。

```
BOOL ftrgBIH(HTBL tb, int msg, int ir, PVOID pb)
{ // 備品管理アクセス制御
  BYTE ofc;

  switch (msg) {
  case TRGM_DEFAULT:
    DefaultTableTrigger(tb, msg, ir, pb);
    ofc = uso(CurUser());
    SetRowBufCellValue(TableRowBuf(tb, ir), 2, &ofc);
    return TRUE;
  }
  if (CurAuth() < 3) {
    switch (LOWORD(msg)) {
    case TRGM_BEFOREUPDATEOF:
      if (HIWORD(msg) == 2) return FALSE;
      // fall through
    case TRGM_BEFOREUPDATE:
    case TRGM_BEFOREDELETE:
      GetTableCellValue(&ofc, tb, ir, 2);
      if (uso(CurUser()) != ofc) return FALSE;
      break;
    }
  }
  return DefaultTableTrigger(tb, msg, ir, pb);
}
```

SetRowBufCellValue 関数は、行バッファ中の項目に値を設定します。 **TableRowBuf** 関数は、表中の指定行を一時的に
行バッファとして扱います。したがって、

```
SetRowBufCellValue(TableRowBuf(tb, ir), 2, &ofc);
```

は、

```
SetTableCellValue(tb, ir, 2, &ofc);
```

と同じことになります。 **SetTableCellValue** 関数を使用しないのは、表トリガ関数内で再び表トリガを呼び出す関数
を使用すると予測のつかない動作となる可能性があるためです。

6 関数リファレンス

ここでは、ストアドプロシージャ中で使用できるBolyai-Cのビルトイン関数を機能別に分類して示します。使用法が自明な関数や、ソースコードから仕様を容易に推測できる単純な関数等は、関数宣言のみ示し詳細な説明は省略することとします。

6.1 C標準ライブラリ関数

C標準ライブラリ関数のうち、以下の関数は、Bolyai-Cでもそのまま使えます。（一部の文字列処理関数は、内部的によりセキュアな関数で再実装されています。）

```
F8B  sqrt(F8B x);
F8B  sin(F8B x);
F8B  cos(F8B x);
F8B  tan(F8B x);
F8B  asin(F8B x);
F8B  acos(F8B x);
F8B  atan(F8B x);
F8B  atan2(F8B y, F8B x);
F8B  exp(F8B x);
F8B  pow(F8B x, F8B y);
F8B  log(F8B x);
F8B  log10(F8B x);
F8B  ceil(F8B x);
F8B  floor(F8B x);
F8B  fabs(F8B x);
I4B  abs(I4B n);
I4B  atoi(rSTR st); // strtol で再実装
F8B  atof(rSTR st); // strtod で再実装
I4B  rand(void);
void srand(U4B sd);
pANY calloc(U4B n, U4B sz);
pANY malloc(U4B sz);
pANY realloc(pANY pv, U4B sz);
void free(pANY pv);
pANY memchr(rANY pv, I4B c, I4B n);
I4B memcmp(rANY p1, rANY p2, I4B n);
pANY memcpy(pANY p1, rANY p2, I4B n);
pANY memmove(pANY p1, rANY p2, I4B n);
pANY memset(pANY pv, I4B c, I4B n);
pSTR strchr(rSTR st, I4B c);
I4B strcmp(rSTR s1, rSTR s2);
I4B stricmp(rSTR s1, rSTR s2); // _stricmp で再実装
I4B strncmp(rSTR s1, rSTR s2, U4B n);
I4B strnicmp(rSTR s1, rSTR s2, U4B n); // _strnicmp で再実装
pSTR strcpy(pSTR ds, rSTR ss); // strcpys で再実装
pSTR strcat(pSTR ds, rSTR ss); // strcats で再実装
U4B strlen(rSTR st);
U4B strspn(rSTR s1, rSTR s2);
I4B strtol(rSTR st, pSTR *ep, I4B ra);
U4B strtoul(rSTR st, pSTR *ep, I4B ra);
F8B strtod(rSTR st, pSTR *ep);
I4B sprintf(pSTR st, rSTR fmt, ...); // vsprintf_s で再実装
I8B _atoi64(rSTR st);
I8B _strtoi64(rSTR st, pSTR *ep, I4B ra);
U8B _strtoui64(rSTR st, pSTR *ep, I4B ra);
```

※注意※ 再実装版の strcpy, strcat 関数および sprintf 関数は、512byteを超える文字列を扱えません。

6.2 ユーティリティ関数

6.2.1 汎用関数

ユーティリティ関数のうち、他に分類されない関数をここにまとめます。

```
I4B sign(F8B d); // 符号(-1, 0, 1)
```

```
I4B iround(F8B d); // 四捨五入整数化
```

```
U4B hash(rSTR st, U4B hs); // 文字列のハッシュ値
```

文字列 `st` のハッシュ値 (0 ~ `hs-1`) を求めます。ハッシュ関数の特性上、ハッシュ表のサイズ `hs` は、8 の倍数より 1 だけ小さい数としてください。

```
U4B nbound(U4B i, U4B n); // n単位に切り上げ
```

整数値 `i` を `n` 単位に切り上げた値を返します。

以下の関数は、メモリ領域の割り当てを行います。

```
pANY halloc(U4B sz, BOOL clr); // ヒープメモリの割り当て
```

```
pANY hrealloc(pANY pv, U4B sz); // ヒープメモリの再割り当て
```

```
BOOL hfree(pANY pv); // ヒープメモリの解放
```

ヒープメモリの割り当てと開放を行います。これらの関数は、C 標準ライブラリの `malloc`、`calloc`、`realloc` 関数を使用して実装されていますが、`hfree` 関数に `halloc`、`hrealloc` 関数で割り当てた現在有効なヒープメモリ領域へのポインタ以外のものを渡しても何も起こらないことが保証されているため、より安全に使用できます。

```
pANY talloc(U4B sz); // 一時的なメモリの割り当て
```

```
pANY ualloc(U4B sz); // 一時的なメモリの割り当て
```

```
pANY valloc(U4B sz); // ゼロクリアされた一時的なメモリの割り当て
```

サイズ `sz` バイトの一時的なメモリ領域を内部の循環バッファ中に割り当てます。このメモリ領域は明示的に解放する必要がありません。割り当てたメモリ領域は、やがて他の呼び出しによって上書きされ不定となるため、その場ですぐに使わなければなりません。循環バッファのサイズは、`talloc` 用が 8KB、`ualloc`、`valloc` 共通用が 256KB で、割り当てサイズの合計がこれを超えると古いデータから順に上書きされて行きます。`talloc` 関数は、512byte以内の小サイズのメモリ割り当て専用で、これを超えるメモリを要求するとエラーになります。`ualloc`、`valloc` 関数も、16KBを超えるメモリを要求するとエラーになります。エラーの場合でも、これらの関数の戻り値が `NULL` となることはなく、循環バッファ内の有効なメモリ領域を指すポインタ値が返されます。

以下の関数は、与えられた整数値と整数値の対応リストを走査するためのものです。

```
I4B idxval(rIDXs ls, I4B id); // インデックス idに対応する整数値
```

```
I4B srchidx(rIDXs ls, I4B va); // 整数値 vaに対応するインデックス
```

整数値は、配列のインデックスと対応します。`ls` は、対応を与える `IDXS` 構造体への参照です。

走査対象が見つからないとき、`idxval` は既定値 0 を、`srchidx` は -1 を返します。

`IDXS` 構造体は、以下のように宣言されています。

```
typedef struct {  
    pI4B pi; // 4byte整数の配列の先頭アドレス  
    I4B ni; // 配列の要素数  
} IDXs;  
typedef IDXs *pIDXs, *PIDXS;  
typedef const IDXs *rIDXs, *PCIDXs;
```

```
I4B mapval(rMAPS ls, I4B ky); // コード番号 kyに対応する整数値
```

```
I4B srchmap(rMAPS ls, I4B va); // 整数値 vaに対応するコード番号
```

整数値は、任意のコード番号と対応します。`ls` は、対応を与える `MAPS` 構造体への参照です。

走査対象が見つからないとき、`mapval` は既定値 0 を、`srchmap` は -1 を返します。

`MAPS` 構造体は、以下のように宣言されています。

```
typedef struct {  
    I4B ky; // コード番号  
    I4B va; // 4byte整数値  
} MAPi;  
typedef MAPi *pMAPi, *PMAPi;
```

```

typedef const MAPI *rMAPI, *PCMAPI;
typedef struct {
    pMAPI pv;    // MAPI 構造体の配列の先頭アドレス
    I4B   ni;    // 配列の要素数
} MAPS;
typedef MAPS *pMAPS, *PMAPS;
typedef const MAPS *rMAPS, *PCMAPS;

```

以下の関数は、与えられた整数値と文字列の対応リストを走査するためのものです。

```

rSTR enmstr(rENMS ls, I4B va);    // インデックス va に対応する文字列
I4B srchenm0(rENMS ls, rSTR st); // 文字列 st に対応するインデックス
I4B srchenm(rENMS ls, rSTR st);  // 文字列 st に対応するインデックス

```

文字列は、配列のインデックスと対応します。ls は、対応を与える ENMS 構造体への参照です。
走査対象が見つからないとき、enmstr は空文字列を、srchenm0 は -1 を、srchenm は既定値 0 を返します。
ENMS 構造体は、以下のように宣言されています。

```

typedef struct {
    rSTR *ps; // 文字列ポインタの配列の先頭アドレス
    I4B   ni; // 配列の要素数
} ENMS;
typedef ENMS *pENMS, *PENMS;
typedef const ENMS *rENMS, *PCENMS;

```

```

rSTR codstr(rCODS ls, I4B va);    // コード番号 va に対応する文字列
I4B srchcod0(rCODS ls, rSTR st); // 文字列 st に対応するコード番号
I4B srchcod(rCODS ls, rSTR st);  // 文字列 st に対応するコード番号

```

文字列は、任意のコード番号と対応します。ls は、対応を与える CODS 構造体への参照です。
走査対象が見つからないとき、codstr は空文字列を、srchcod0 は -1 を、srchcod は既定値 0 を返します。
CODS 構造体は、以下のように宣言されています。

```

typedef struct {
    I4B   va; // コード番号
    rSTR  st; // 文字列ポインタ
} CODI;
typedef CODI *pCODI, *PCODI;
typedef const CODI *rCODI, *PCCODI;
typedef struct {
    pCODI pv; // CODI 構造体の配列の先頭アドレス
    I4B   ni; // 配列の要素数
} CODS;
typedef CODS *pCODS, *PCODS;
typedef const CODS *rCODS, *PCCODS;

```

以下の関数は、Windows の A P I 関数を単にラッピングしたものです。

```

void get_system_time(pSTMI lp); // システム日時の取得
void get_local_time(pSTMI lp);  // 現在日時の取得

```

STMI 構造体は、Windows A P I の SYSTEMTIME 構造体に対応しています。

```

typedef struct {
    U2B   yr, mo, dw, dy; // 年、月、曜日、日
    U2B   hr, mn, sc, ms; // 時、分、秒、ミリ秒
} STMI;
typedef STMI *pSTMI, *PSTMI;
typedef const STMI *rSTMI, *PCSTMI;

```

```

U4B get_tick_count(void); // 経過時間の取得
void wait_millsec(U4B ms); // ms ミリ秒待つ

```

```

I4B compare_file_time(rFTMI ft1, rFTMI ft2); // ファイル時刻の比較
BOOL file_time_to_system_time(pSTMI lp, rFTMI ft); // ファイル時刻をシステム日時に変換

```

```

BOOL system_time_to_file_time(pFTMI ft, rSTRMI lp); // システム日時をファイル時刻に変換
FTMI 構造体は、Windows API の FILETIME 構造体に対応しています。
typedef struct {
    U4B ldt: // ファイル時刻下位 4byte
    U4B hdt: // ファイル時刻上位 4byte
} FTMI;
typedef FTMI *pFTMI, *PFTMI;
typedef const FTMI *rFTMI, *PCFTMI;

```

```

void get_user_name(pSTR nam, I4B sz); // OS のログインユーザー名の取得
void get_computer_name(pSTR nam, I4B sz); // クライアント PC のコンピュータ名の取得

```

6.2.2 文字種の判別関数

以下の関数は、文字種を判別します。文字コードは、Shift_JISを前提しています。

```

BOOL isspc(rSTR st); // 空白文字 0x20, 0x09~0x0d
BOOL isctl(rSTR st); // 制御文字 0x09~0x0d
BOOL issym(rSTR st); // 特殊記号 ! # $ % & ( ) * + , - . / : ; < = > ? @ [ ] ^ ` { | } ~
BOOL isdig(rSTR st); // 10進数字 0~9
BOOL isodig(rSTR st); // 8進数字 0~7
BOOL isxdig(rSTR st); // 16進数字 0~9, A~F, a~f
BOOL isupr(rSTR st); // 英大文字 A~Z
BOOL islwr(rSTR st); // 英小文字 a~z
BOOL isalp(rSTR st); // 英字 A~Z, a~z
BOOL isalpx(rSTR st); // 英字、アンダーバー
BOOL isaln(rSTR st); // 英字、数字
BOOL isalnx(rSTR st); // 英字、数字、アンダーバー
BOOL isasc(rSTR st); // 表示可能なASCII文字 0x20~0x7e
BOOL ishan(rSTR st); // 半角カナ文字 0xa1~0xdf
BOOL isprn(rSTR st); // 表示可能文字 0x20~0x7e, 0xa1~0xdf
BOOL iszen1(rSTR st); // 全角文字の第1バイト目 0x81~0x9f, 0xe0~0xfc
BOOL iszen2(rSTR st); // 全角文字の第2バイト目 0x40~0x7e, 0x80~0xfc
BOOL iszen(rSTR st); // 全角文字
BOOL isret(rSTR st); // 改行 CRLF
BOOL iszspc(rSTR st); // 全角空白

```

6.2.3 文字列関数

以下の関数は、文字列をコピーします。

```

pSTR spncpy(pSTR sd, rSTR p1, rSTR p2); // 部分文字列のコピー
文字列中のポインタ p1 から p2 の直前までの部分を切り出して格納先 sd にコピーし sd を返します。格納先 sd には、
p2-p1+1 byte の領域が必要です。

```

```

pSTR strcpyn(pSTR sd, rSTR ss, I4B n); // 指定長さまで文字列をコピー
pSTR strcpyx(pSTR sd, rSTR ss, I4B n); // 指定長さまで文字列をコピー(残りはヌル文字で埋める)
文字列 ss を格納先 sd にコピーし sd を返します。格納先 sd には、n+1 byte の領域が必要です。文字列 ss の長さが
n を超える場合、領域に収まるように切り詰めてコピーします。切り詰めた場合でも sd は必ずヌル文字で終了し、
全角文字が途中で切れることはありません。strcpy と strcpyx の違いは、ss が n より短い場合、残りのバイトを
全てヌル文字で埋めるか埋めないかの違いです。

```

```

pSTR strcpsys(pSTR sd, rSTR ss); // strcpy のセキュア版
pSTR strcats(pSTR sd, rSTR ss); // strcat のセキュア版
文字列の長さが 511 を超える場合、512byte の領域に収まるように切り詰めてコピーします。切り詰めた場合でも sd は
必ずヌル文字で終了し、全角文字が途中で切れることはありません。

```

以下の関数は、文字列を調べて結果を返します。

```

BOOL nthzen1(rSTR st, I4B n); // 文字列中の指定バイトが全角文字第1バイト目かどうかの判定
BOOL nthyen(rSTR st, I4B n); // 文字列中の指定バイトが「¥」かどうかの判定

```



```
pSTR mbschr(rSTR st, I4B c); // strchr の全角文字対応版
pSTR mbsrchr(rSTR st, I4B c); // strrchr の全角文字対応版
U4B mbslen(rSTR st); // 全角文字を1文字と数えた時の文字列の長さ
```

```
I4B strmatchx(rSTR s1, rSTR s2, I4B flg); // 文字列の照合
```

文字列 **s1** と **s2** を先頭から1文字ずつ比較して一致する長さ (**s1** 側のバイト数) を返します。比較に際しては、大文字と小文字の違い、全角と半角の違い、空白の有無を無視するかどうかの条件を指定できます。比較条件 **flg** は、以下のフラグ値の組み合わせで与えます。

- STX_UPR 大文字と小文字を区別しない
- STX_ZEN 全角と半角を区別しない
- STX_DSP 空白を無視する

```
BOOL strmatch(rSTR s1, rSTR s2); // 文字列のパターン照合
```

文字列 **s1** がパターン文字列 **s2** に合致するかどうか判定します。パターン文字列内では、以下に示す記号が文字種を表します。

- 「#」 半角10進数字 0～9
- 「\$」 半角英字 A～Z, a～z
- 「%」 半角英字と数字 0～9, A～Z, a～z
- 「*」 表示可能なASCII文字 0x20～0x7e
- 「~」 任意の1バイト

```
pSTR strfindx(rSTR s1, rSTR s2, I4B flg); // 部分文字列の検索(汎用)
```

文字列 **s1** 中に文字列 **s2** が現れるかどうか検索し、見つければその部分文字列へのポインタ値を、見つからなければ **NULL** を返します。部分文字列の比較に際しては、大文字と小文字の違い、全角と半角の違い、空白の有無を無視するかどうかの比較条件 **flg** (**strmatchx** 関数と同じ) を指定できます。

```
pSTR strfind(rSTR s1, rSTR s2); // 部分文字列の検索
BOOL strlfind(rSTR s1, rSTR s2); // 文字列の前方一致検索
BOOL strrfind(rSTR s1, rSTR s2); // 文字列の後方一致検索
pSTR strifind(rSTR s1, rSTR s2); // 部分文字列の検索(大文字と小文字を区別しない)
BOOL strlfind(rSTR s1, rSTR s2); // 文字列の前方一致検索(大文字と小文字を区別しない)
BOOL strirfind(rSTR s1, rSTR s2); // 文字列の後方一致検索(大文字と小文字を区別しない)
BOOL strlike(rSTR s1, rSTR s2); // 文字列の曖昧検索
BOOL strmlike(rSTR s1, rSTR s2); // 部分文字列の曖昧検索
BOOL strllike(rSTR s1, rSTR s2); // 文字列の前方一致曖昧検索
BOOL strrlike(rSTR s1, rSTR s2); // 文字列の後方一致曖昧検索
I4B strcount(rSTR s1, rSTR s2); // 部分文字列のカウント
```

以下の関数は文字列を変換します。

```
pSTR tozenb(pSTR bf, rSTR st); // 文字列中の半角文字を対応する全角文字に変換
pSTR tohanb(pSTR bf, rSTR st); // 文字列中の(半角化可能な)全角文字を対応する半角文字に変換
pSTR touprb(pSTR bf, rSTR st); // 文字列中の英小文字を英大文字に変換
pSTR tolwrb(pSTR bf, rSTR st); // 文字列中の英大文字を英小文字に変換
pSTR tohirab(pSTR bf, rSTR st); // 文字列中の全角カタカナを全角ひらがなに変換
pSTR tokatab(pSTR bf, rSTR st); // 文字列中の全角ひらがなを全角カタカナに変換
pSTR ltrimb(pSTR bf, rSTR st); // 文字列左側の空白削除
pSTR rtrimb(pSTR bf, rSTR st); // 文字列右側の空白削除
pSTR btrimb(pSTR bf, rSTR st); // 文字列両側の空白削除
pSTR delspcb(pSTR bf, rSTR st); // 文字列中の空白削除
pSTR delctlb(pSTR bf, rSTR st); // 文字列中の制御文字を削除
pSTR ctlspcb(pSTR bf, rSTR st); // 文字列中の制御文字を空白に変換
pSTR cipherb(pSTR bf, rSTR st); // 暗号化
pSTR inscma(pSTR bf, rSTR st); // 数値に3桁コンマを付加
pSTR delcma(pSTR bf, rSTR st); // 数値の3桁コンマを削除
pSTR extascb(pSTR bf, rSTR st); // ASCII文字のみ取り出し
pSTR midstrb(pSTR bf, rSTR st, I4B n1, I4B n2); // n1文字目からn2文字分の文字列を取り出し
pSTR streplb(pSTR bf, rSTR st, rSTR s1, rSTR s2); // 部分文字列の置換(s1→s2)
pSTR strconcb(pSTR bf, rSTR s1, rSTR s2); // 文字列の連結
```

```

pSTR tozen(rSTR st); // 文字列中の半角文字を対応する全角文字に変換
pSTR tohan(rSTR st); // 文字列中の(半角化可能な)全角文字を対応する半角文字に変換
pSTR toupr(rSTR st); // 文字列中の英小文字を英大文字に変換
pSTR tolwr(rSTR st); // 文字列中の英大文字を英小文字に変換
pSTR tohira(rSTR st); // 文字列中の全角カタカナを全角ひらがなに変換
pSTR tokata(rSTR st); // 文字列中の全角ひらがなを全角カタカナに変換
pSTR ltrim(rSTR st); // 文字列左側の空白削除
pSTR rtrim(rSTR st); // 文字列右側の空白削除
pSTR btrim(rSTR st); // 文字列両側の空白削除
pSTR delspc(rSTR st); // 文字列中の空白削除
pSTR delctl(rSTR st); // 文字列中の制御文字を削除
pSTR ctlspsc(rSTR st); // 文字列中の制御文字を空白に変換
pSTR cipher(rSTR st); // 暗号化
pSTR inscma(rSTR st); // 数値に3桁コンマを付加
pSTR delcma(rSTR st); // 数値の3桁コンマを削除
pSTR extasc(rSTR st); // ASCII文字のみ取り出し
pSTR midstr(rSTR st, I4B n1, I4B n2); // n1文字目からn2文字分の文字列を取り出し
pSTR strrepl(rSTR st, rSTR s1, rSTR s2); // 部分文字列の置換(s1→s2)
pSTR strconc(rSTR s1, rSTR s2); // 文字列の連結

```

文字列の変換関数には、第1引数bfに変換結果を格納する第1グループと、`malloc`で確保した一時的なメモリ領域中に変換結果を格納する第2グループがあります。

以下の関数は、値を書式化した文字列を返します。

```

I4B sprintfs(pSTR st, rSTR fmt, ...); // sprintfのセキュア版
I4B csprintf(pSTR st, rSTR fmt, ...); // sprintfの3桁コンマ付加版

```

```

rSTR format(rSTR fmt, ...); // 書式化文字列
rSTR cformat(rSTR fmt, ...); // 書式化文字列(数値に3桁コンマを付加)

```

これらの関数は、`sprintfs`、`csprintf`で書式化された文字列を`malloc`で確保した一時的なメモリ領域中に格納してそのポインタ値を返します。

※注意※ 第2グループの文字列変換関数および書式化文字列関数の使い方には注意が必要です。これらの関数の戻り値のポインタ先の内容は、その場ですぐに他の領域にコピーするなどして使わなければ、やがて他の呼び出しによって上書きされ不定となります。またこれらの関数は512byteを超える文字列を扱えません。

以下の関数は、文字列を数値に変換します。

```

I4B ztoi(rSTR st); // 文字列をintに変換(atoiの全角文字対応版)
U4B ztou(rSTR st); // 文字列をUINTに変換
I8B ztoi64(rSTR st); // 文字列をINT64に変換
U8B ztou64(rSTR st); // 文字列をUINT64に変換
F8B ztof(rSTR st); // 文字列をdoubleに変換(atofの全角文字対応版)
I4B cztoi(rSTR st); // 3桁コンマを削除して文字列をintに変換
U4B cztou(rSTR st); // 3桁コンマを削除して文字列をUINTに変換
I8B cztoi64(rSTR st); // 3桁コンマを削除して文字列をINT64に変換
U8B cztou64(rSTR st); // 3桁コンマを削除して文字列をUINT64に変換
F8B cztof(rSTR st); // 3桁コンマを削除して文字列をdoubleに変換

```

6.2.4 日付／時刻関数

以下の関数は、日付データを操作します。

```
B00L is_leap_year(I4B yr); // yr 年がうるう年かどうかの判定
I4B days_of_month(I4B yr, I4B mo); // yr 年 mn 月の月の日数
I4B day_of_week(I4B yr, I4B mo, I4B dy); // yr 年 mn 月 dy 日の曜日 (0:日曜~6:土曜)
I4B date_serial(I4B yr, I4B mo, I4B dy); // yr 年 mn 月 dy 日の日付シリアル値
I4B year(I4B sr); // 日付シリアル値の年
I4B month(I4B sr); // 日付シリアル値の月
I4B day(I4B sr); // 日付シリアル値の日
I4B youbi(I4B sr); // 日付シリアル値の曜日 (0:日曜~6:土曜)
I4B era_year(I4B sr); // 日付シリアル値の年号の年
I4B today(void); // ローカル時間での今日の日付の日付シリアル値
I4B date_diff(I4B s1, I4B s2, I4B sw); // 日付s1, s2の差 (sw = 1:年数、2:月数)
I4B ztod(rSTR st); // 年月日を表す文字列を日付シリアル値に変換
I4B ztom(rSTR st); // 年月を表す文字列を日付シリアル値に変換
I4B dprintf(pSTR st, rSTR fmt, I4B sr); // 日付シリアル値を書式化して文字列に変換
rSTR dformat(rSTR fmt, I4B sr); // 日付シリアル値の書式化文字列
```

日付シリアル値とは、1900年1月1日を1日目と数えるときの累計日数です。

書式指定文字列 **fmt** 中で使用できる変換指定文字は、以下の通りです。

```
yyyy 西暦の年 4 桁
yy   西暦の年 2 桁
y    yyyyに同じ
mm   月 2 桁
m    月
dd   日 2 桁
d    日
ggg  年号 (明治、大正、昭和、平成、令和)
gg   年号 (明、大、昭、平、令)
g    年号 (M、T、S、H、R)
ee   年号の年 2 桁
e    年号の年
```

以下の関数は、時刻データを操作します。

```
I4B time_serial(I4B hr, I4B mn, I4B sc); // hr 時 mn 分 sc 秒の時刻シリアル値
I4B hour(I4B tm); // 時刻シリアル値の時
I4B minute(I4B tm); // 時刻シリアル値の分
I4B second(I4B tm); // 時刻シリアル値の秒
I4B ztot(rSTR st); // 時分秒を表す文字列を時刻シリアル値に変換
I4B tprintf(pSTR st, rSTR fmt, I4B tm); // 時刻シリアル値を書式化して文字列に変換
rSTR tformat(rSTR fmt, I4B tm); // 時刻シリアル値の書式化文字列
```

時刻シリアル値とは、累計秒数 ($hr * 3600 + mn * 60 + sc$) です。

書式指定文字列 **fmt** 中で使用できる変換指定文字は、以下の通りです。

```
hh   時 2 桁
h    時
mm   分 2 桁
m    分
ss   秒 2 桁
s    秒
```

以下の関数は、日付時刻データを操作します。

```
F8B now(void); // 現在時刻を表す実数値 (整数部が日付シリアル値、小数部が時刻シリアル値／86400)
F8B ztodt(rSTR st); // 現在時刻を表す文字列を実数値に変換
I4B dtprintf(pSTR st, rSTR fmt, F8B dt); // 現在時刻を表す実数値を書式化して文字列に変換
rSTR dtformat(rSTR fmt, F8B dt); // 現在時刻を表す実数値の書式化文字列
```

書式指定文字列 **fmt** は、日付の書式指定の後ろに空白を挟んで時刻の書式指定を続けたものです。

6.2.5 ファイル入出力関数

以下の関数は、ファイル名（ディレクトリ名）を操作します。

I4B check_path(rSTR nam); // ファイル名要素の検査

ファイル名 **nam** が含む要素を次のフラグ値の組み合わせで返します。

CHP_DIR ディレクトリ名（ドライブ名を含む）

CHP_NAM ファイル名

CHP_EXT 拡張子

pSTR change_path(pSTR nam, rSTR src, I4B flg); // ファイル名要素の変更

ファイル名 **nam** の要素のうち、フラグ値の組み合わせ **flg** が示す要素を **src** の要素で置換します。**src** に **NULL** を与えると、該当要素の削除になります。

以下の関数は、汎用のファイル入出力およびファイル（ディレクトリ）操作をサポートします。ほとんどの関数は、WindowsのAPI関数を単にラッピングしたものです。

HFIL open_file(rSTR nam, U4B flg); // ファイルを開く

flg は、ファイルアクセスのモードを示します。

OP_RDO 既存読込

OP_WRI 新規書出

OP_APD 追加書出

OP_RDW 既存読込／書出

BOOL close_file(HFIL hf); // ファイルを閉じる

U4B set_file_ptr(HFIL hf, I4B ds, U4B md); // ファイルポインタを移動する

BOOL set_end_of_file(HFIL hf); // 現在のファイルポインタ位置をファイルの終わりにする

U4B get_file_size(HFIL hf); // ファイルサイズを取得する

BOOL read_file(HFIL hf, pANY bf, U4B sz); // ファイル読込

BOOL write_file(HFIL hf, rANY bf, U4B sz); // ファイル書出

BOOL read_line(HFIL hf, pSTR bf, I4B sz); // テキスト行の読込

BOOL write_line(HFIL hf, rSTR bf); // テキスト行の書出

HFIL find_first_file(rSTR nam, pFFDI lp); // ファイル名の検索(初回)

BOOL find_next_file(HFIL hf, pFFDI lp); // ファイル名の検索(継続)

BOOL find_close(HFIL hf); // ファイル名の検索ハンドルを閉じる

FFDI 構造体は、Windows API の WIN32_FIND_DATA 構造体に対応しています。

typedef struct {

U4B atr; // ファイル属性

FTMI cre; // ファイル作成時間

FTMI acc; // ファイル最終アクセス時間

FTMI wri; // ファイル最終書込時間

U4B szh; // ファイルサイズ上位 4byte

U4B szl; // ファイルサイズ下位 4byte

U4B rs0;

U4B rs1;

I1B nam[260]; // ファイル名

I1B alt[14]; // 短いファイル名

U2B rs2;

} FFDI;

typedef FFDI *pFFDI, *PFFDI;

BOOL path_exists(rSTR dir); // ディレクトリが存在する

BOOL file_exists(rSTR nam); // ファイルが存在する

```

BOOL get_file_info(rSTR nam, pFADI lp); // ファイル情報の取得
FADI 構造体は、Windows A P I の WIN32_FILE_ATTRIBUTE_DATA 構造体に対応しています。
typedef struct {
    U4B  atr; // ファイル属性
    FTMI cre; // ファイル作成時間
    FTMI acc; // ファイル最終アクセス時間
    FTMI wri; // ファイル最終書込時間
    U4B  szh; // ファイルサイズ上位 4byte
    U4B  szl; // ファイルサイズ下位 4byte
} FADI;
typedef FADI *pFADI, *PFADI;

U4B  file_size(rSTR nam); // ファイルサイズの取得

BOOL new_file(rSTR nam); // 空のファイルを作成(ファイルを空にする)
BOOL new_file_if_not_exists(rSTR nam); // ファイルが存在しないとき空のファイルを作成
BOOL move_file(rSTR nam, rSTR src); // ファイル名の付け替え
BOOL force_move_file(rSTR nam, rSTR src); // ファイルの移動
BOOL copy_file(rSTR nam, rSTR src); // ファイルのコピー
BOOL delete_file(rSTR nam); // ファイルの削除

BOOL new_path(rSTR dir); // ディレクトリの作成
BOOL new_path_if_not_exists(rSTR dir); // ディレクトリが存在しないとき作成
BOOL move_file_to(rSTR dir, rSTR src); // ファイルを指定ディレクトリに移動
BOOL copy_file_to(rSTR dir, rSTR src); // ファイルを指定ディレクトリにコピー
void get_working_path(pSTR dir, I4B sz); // カレントディレクトリの取得
BOOL set_working_path(rSTR dir); // カレントディレクトリの設定

```

6.2.6 エラー処理関数

以下の関数は、エラー処理に関する情報を設定／取得します。

BOOL register_error_descriptions(rCODS ed); // エラー定義
エラーコードとエラー記述文字列の対応情報を登録します。

rSTR error_description(U4B ec); // エラー記述
エラーコード **ec** のエラー記述文字列を返します。

I4B set_error_message_index_base(I4B i); // インデックスベース (0／1) の設定
I4B error_message_index_base(void); // インデックスベース (0／1) の取得

エラーメッセージの行／列番号を 0 から数えるか 1 から数えるかの設定です。

プログラム内部では、行／列インデックスは、常に 0 から数えますが、画面表示上は 1 から数えている（デフォルト設定時）ため、その違いを調整するためのものです。

void set_error_log_file(rSTR nam); // エラーログファイル名の設定
rSTR error_log_file(void); // エラーログファイル名の取得
エラーログファイル名のデフォルト値は、**サイト名.LOG** です。

I4B first_error(void); // 最初のエラー
I4B last_error(void); // 最後のエラー
最初／最後に検出されたエラーのエラーコードを返します。

void reset_error(void); // エラー情報のクリア
エラー情報をクリアします。

BOOL enable_error_reporting(void); // エラー表示許可
BOOL disable_error_reporting(void); // エラー表示抑止
エラーメッセージの表示を許可／抑止します。

戻り値は、関数を呼び出す前に表示許可状態であったか、表示抑止状態であったかを **TRUE** / **FALSE** で示します。
プログラム起動時のデフォルト設定は、表示許可です。エラーメッセージにユーザーが「キャンセル」で答えると、表示抑止状態になります。**reset_error** 関数でエラー情報をクリアすると、再び表示許可状態になります。

BOOL enable_error_logging(void); // エラーログを残す設定
BOOL disable_error_logging(void); // エラーログを残さない設定
エラーログをファイルに書き出す／書き出さない設定をします。
戻り値は、関数を呼び出す前に書き出す設定であったか、書き出さない設定であったかを **TRUE** / **FALSE** で示します。
プログラム起動時のデフォルトは残さない設定ですが、ユーザーがオプション設定で変更できます。

pANY ierr(I4B ec); // エラー処理
pANY ferr(I4B ec, rSTR nam); // エラー処理
pANY serr(I4B ec, rSTR nam, I4B ir, I4B ic); // エラー処理
エラーコード **ec** のエラーメッセージボックスを表示します。**nam** はエラーが発生したファイル名または表名を示します。**ir**, **ic** は行／列を示します。
これらの関数は、常に **NULL** を返します。

6.2.7 値をポインタにする関数

以下の関数は、値を受け取ってその値を指すポインタ値を返します。値は **uallloc** の循環バッファに格納されます。

rANY I4BBox(I4B va); // int 値を int へのポインタに変換
rANY I8BBox(I8B va); // INT64 値を INT64 へのポインタに変換
rANY F4BBox(F4B va); // float 値を float へのポインタに変換
rANY F8BBox(F8B va); // double 値を double へのポインタに変換

6.3 データベース操作関数

6.3.1 列情報操作関数

以下の関数は、列情報を操作します。列情報は、表データの列項目の仕様を定義します。

BOOL RegisterColumnType(rTYPI pe); // データ型の登録

ユーザー定義のデータ型を登録します。pe は、データ型を定義する TYPI 構造体への参照です。

TYPI 構造体は、以下のように宣言されています。

```
typedef struct {
    I4B    itp; // 登録型コード／基本型コード
    I4B    atr; // 既定の列幅／既定の文字揃えコード
    FSCN    fscn; // 文字列から値を読み込む関数
    FPRN    fprn; // 値を文字列に書き出す関数
    FCMP    fcmp; // 値同士を比較するときの大小関係を定義する関数
    FLST    flst; // 入力候補リストを生成する関数
    uANY    pa0; // 付加情報の既定値(書式指定文字列、入力候補リスト情報等)
    uANY    pa1; // 付加情報の既定値
    uANY    pa2; // 付加情報の既定値
    I4B    flg; // 内部使用のため予約
} TYPI;
```

typedef TYPI *pTYPI, *PTYPI;

typedef const TYPI *rTYPI, *PCTYPI;

itp は、登録型コード id と基本型コード bt を次のように組み合わせて示します。

MKU2B(id, bt) または、ITP(id, bt)

既定の基本型コードは、次の12種類で、全てのデータ型は、どれかの基本型の派生型となります。

TID_I1B char 型
TID_U1B BYTE 型
TID_I2B short 型
TID_U2B WORD 型
TID_I4B int 型
TID_U4B UINT 型
TID_I8B INT64 型
TID_U8B UINT64 型
TID_F4B float 型
TID_F8B double 型
TID_STR char[n] 型
TID_BIN BYTE[n] 型

文字列型とバイト列型は、列情報として実際に使用する際に具体的なサイズを指定します。

文字列は、ヌル文字で終了させる必要があるため、格納できる文字列長は n-1 文字（全角なら (n-1)/2 文字まで）となります。既定のバイト列型には、有効な fscn, fprn がいないため、通常はスペースを予約するためにのみ使用します。

既定の基本型以外に、以下の型コードが既定のデータ型としてあらかじめ登録されています。

TID_HEX TID_U4B の派生型、16進表記
TID_M4B TID_I4B の派生型、3 桁コンマ付表記
TID_M8B TID_F8B の派生型、3 桁コンマ付表記
TID_YMD TID_U4B の派生型、日付シリアル値、年月日表記
TID_YMN TID_U4B の派生型、日付シリアル値、年月表記
TID_TIM TID_U4B の派生型、時刻シリアル値、時分秒表記
TID_DTM TID_F8B の派生型、整数部が日付シリアル値、小数部が時刻シリアル値／86400、年月日時分秒表記
TID_E1B TID_U1B の派生型、枚举型（ENMS 構造体でリスト定義）
TID_E2B TID_U2B の派生型、枚举型（ENMS 構造体でリスト定義）
TID_E4B TID_U4B の派生型、枚举型（ENMS 構造体でリスト定義）
TID_C1B TID_U1B の派生型、コード型（CODS 構造体でリスト定義）
TID_C2B TID_U2B の派生型、コード型（CODS 構造体でリスト定義）
TID_C4B TID_U4B の派生型、コード型（CODS 構造体でリスト定義）
TID_S1B TID_U1B の派生型、集合型（ENMS 構造体でリスト定義）
TID_S2B TID_U2B の派生型、集合型（ENMS 構造体でリスト定義）
TID_S4B TID_U4B の派生型、集合型（ENMS 構造体でリスト定義）
TID_A1B TID_U1B の派生型、代替キー入力型

TID_A2B TID_U2B の派生型、代替キー入力型
TID_A4B TID_U4B の派生型、代替キー入力型
TID_ASC TID_STR の派生型、ASCII文字列型

新たなデータ型を登録する際には、登録型コード **id** を TID_USR 以上の未使用コード (64~127) とし、基本型コード **bt** を登録済みの型コードから選択します。他のメンバ項目は、基本型の定義を上書きしたい項目のみ値を設定し、それ以外の項目は 0 または **NULL** にしておくことで基本型の定義を継承します。

atr は、既定の列幅 **wd**、既定の文字揃えコード **al** を次のように組み合わせて示します。

MKU2B(wd, al) または、**ATR(wd, al)**

文字揃えコードは、0 が左寄せ、1 が中央揃え、2 が右寄せです。

既定の列幅と文字揃え指定は、数値項目の列情報に列幅と文字揃え指定がないときに適用されます。

fscn は、次の宣言で表される型のコールバック関数のアドレスです。

```
typedef pANY (*FSCN) (pANY pv, rSTR st, rCOLI co);
```

この関数は、文字列 **st** から値を読み込んでアドレス **pv** に格納します。**co** は列情報への参照、戻り値は **pv** です。

fprn は、次の宣言で表される型のコールバック関数のアドレスです。

```
typedef pSTR (*FPRN) (pSTR st, rANY pv, rCOLI co);
```

この関数は、アドレス **pv** にある値を文字列として **st** に書き出します。**co** は列情報への参照、戻り値は **st** です。

fcmp は、次の宣言で表される型のコールバック関数のアドレスです。

```
typedef I4B (*FCMP) (rANY p1, rANY p2, rCOLI co);
```

この関数は、アドレス **p1**、**p2** にある値同士を比較してその大小関係を返します。**co** は列情報への参照、戻り値は、値が等しいとき 0、「>」のとき正、「<」のとき負です。

flst は、次の宣言で表される型のコールバック関数のアドレスです。

```
typedef uANY (*FLST) (I4B im, rCOLI co);
```

この関数は、項目のインデックス **im** に対応する文字列を指すポインタ値を返します。項目数は **im** を -1 にすると得られます。**co** は列情報への参照です。

pa0、**pa1**、**pa2** は、データ型によって異なる使われ方をする付加情報の既定値を格納します。既定値は、項目の列情報に付加情報の指定がないときに適用されます。既定のデータ型では、数値項目の書式指定文字列の既定値が **pa0** に格納されています。数値項目の書式指定文字列の既定値は、以下の通りです。

```
TID_I1B "%d"
TID_U1B "%d"
TID_I2B "%d"
TID_U2B "%d"
TID_I4B "%d"
TID_U4B "%u"
TID_I8B "%lld"
TID_U8B "%llu"
TID_F4B "%. 6G"
TID_F8B "%. 15G"
TID_HEX "%X"
TID_M4B "%d"
TID_M8B "%. 0f"
TID_YMD "yyyy/mm/dd"
TID_YMN "yyyy/mm"
TID_TIM "hh:mm:ss"
TID_DTM "yyyy/mm/dd hh:mm:ss"
```

例えば、TID_I1B の **fprn** は、

```
sprintfs(st, (rSTR)((COLI *)co)->pa0, (I4B)*(rI1B)pv); return st;
```

TID_M4B の **fprn** は、

```
csprintf(st, (rSTR)((COLI *)co)->pa0, *(rI4B)pv); return st;
```

TID_YMD の **fprn** は、

```
dprintf(st, (rSTR)((COLI *)co)->pa0, *(rU4B)pv); return st;
```

のように実装されています。

列挙型、コード型、集合型の入力候補リスト情報 (ENMS または CODS 構造体への参照) も **pa0** を使用しますが、既定値はないため項目の列情報での指定が必須となります。

TID_E1B の **fprn** は、

```
return strcpys(st, enmstr((rENMS)((COLI *)co)->pa0, *(rU1B)pv));
```

TID_C4B の **fprn** は、

```
return strcpys(st, codstr((rCODS)((COLI *)co)->pa0, *(rU4B)pv));
```

のように実装されています。

項目の列情報を記述するのは、COLI 構造体です。
COLI 構造体は、以下のように宣言されています。

```
typedef struct {  
    I4B    itp; // 型コード／サイズ  
    I4B    atr; // 列幅／文字揃えコード  
    rSTR   nam; // 列名  
    rSTR   tit; // 列タイトル  
    rSTR   dsc; // 列説明  
    I4B    flg; // 列属性  
    FCAL   fcal; // 計算関数／初期値設定関数  
    FCHK   fchk; // チェック関数  
    uANY   pa0; // 付加情報  
    uANY   pa1; // 付加情報  
    uANY   pa2; // 付加情報  
    uANY   pa3; // 内部使用のため予約  
} COLI;
```

itp は、型コード id とサイズ n を次のように組み合わせて示します。

MKU4B(id, n)

文字列型とバイト列型以外はサイズ指定を省略できるので、型コードだけでよく、サイズ指定の必要な文字列型とバイト列型は、以下のマクロを使用して型を示します。

STR(n) char[n] 型
BIN(n) BYTE[n] 型
ASC(n) char[n] 型、ASCII 文字列型

また、型コードには、それが計算項目であることを示すビットフラグ (0x80) を付加することができます。計算項目フラグを付加するマクロは、CAL(id) です。計算項目用としてあらかじめ以下の既定の定数が用意されています。

```
enum {  
    CAL_I1B = CAL(TID_I1B), CAL_U1B = CAL(TID_U1B),  
    CAL_I2B = CAL(TID_I2B), CAL_U2B = CAL(TID_U2B),  
    CAL_I4B = CAL(TID_I4B), CAL_U4B = CAL(TID_U4B),  
    CAL_I8B = CAL(TID_I8B), CAL_U8B = CAL(TID_U8B),  
    CAL_F4B = CAL(TID_F4B), CAL_F8B = CAL(TID_F8B),  
    CAL_M4B = CAL(TID_M4B), CAL_M8B = CAL(TID_M8B),  
    CAL_YMD = CAL(TID_YMD), CAL_YMN = CAL(TID_YMN),  
    CAL_E1B = CAL(TID_E1B), CAL_E2B = CAL(TID_E2B),  
    CAL_E4B = CAL(TID_E4B), CAL_C1B = CAL(TID_C1B),  
    CAL_C2B = CAL(TID_C2B), CAL_C4B = CAL(TID_C4B),  
    CAL_S1B = CAL(TID_S1B), CAL_S2B = CAL(TID_S2B),  
    CAL_S4B = CAL(TID_S4B), CAL_A1B = CAL(TID_A1B),  
    CAL_A2B = CAL(TID_A2B), CAL_A4B = CAL(TID_A4B),  
};
```

atr は、列幅 wd、文字揃えコード al を次のように組み合わせて示します。

MKU2B(wd, al) または、ATR(wd, al)

列幅も文字揃え指定も省略時にはデータ型に応じて適切に自動設定されます。列幅のみ変えたいときは atr に列幅を指定します。文字揃えコードは、0 が既定値、1 が中央揃え、2 が右寄せ、-1 が左寄せです。

nam は、列名にする16byte以内の英字と数字の (C 言語の識別子形式の) 文字列です。

tit は、列タイトルにする32byte以内の任意文字列です。省略すると、nam が tit として使われます。列タイトルは、既定の列見出しとして使われるため、列タイトルの長さは項目の列幅に影響します。

dsc は、列説明にする64byte以内の任意文字列です。省略すると、tit が dsc として使われます。列説明は、列タイトルが短くて説明を補う必要がある場合に用います。

flg は、以下のフラグ値を組み合わせて項目の属性を示します。

CF_ACT 制御文字の入力を許す (画面表示やテキスト書出に不具合が発生する場合があります。)
CF_DCT 制御文字を削除する (デフォルトでは制御文字は空白文字に変換されます。)
CF_LTR 文字列先頭の空白を削除する
CF_RTR 文字列末尾の空白を削除する
CF_DSP 文字列中の空白を削除する
CF_HAN 文字列を半角に変換する
CF_ZEN 文字列を全角に変換する
CF_HCH 文字列を暗号化する (編集できなくなります。)
CF_LNK ハイパーリンク項目

以上のフラグは、文字列型の項目に対してのみ有効です。

CF_HZR 0を表示しない
CF_RDO 編集できない項目
CF_DDL ドロップダウンリスト入力
CF_DDN コンボボックス入力
CF_DED 内部使用のため予約
CF_GR1 ダイアログボックス上での項目配置指定（前項目と同じ行の12文字目に配置）
CF_GR2 ダイアログボックス上での項目配置指定（前項目と同じ行の18文字目に配置）
CF_GR3 ダイアログボックス上での項目配置指定（前項目と同じ行の24文字目に配置）
CF_PAC ダイアログボックス上での項目配置指定（前項目と同じ行のすぐ後ろに配置）
CF_KEY 主キー制約項目
CF_REF 外部キー制約項目／外部参照項目
CF_NZR 非ゼロ制約項目
CF_UNI 一意性制約項目
CF_OCL 内部使用のため予約
CF_OCK 内部使用のため予約
CF_OSC 内部使用のため予約
CF_OPR 内部使用のため予約
CF_OCM 内部使用のため予約
CF_OLS 内部使用のため予約

fcal は、次の宣言で表される型のコールバック関数のアドレスです。

```
typedef pANY (*FCAL) (pANY pv, rANY pb, rCOLI co);
```

この関数は、行のアドレス pb と列情報 co を参照して得た計算結果の値をアドレス pv に格納します。
計算項目の計算内容は、この関数で定義します。外部参照項目以外の計算項目には、fcal の指定が必須です。
実表項目に fcal を指定すると、項目への初期値設定関数となります。無指定時のデフォルト動作はゼロクリアです。
fchk は、次の宣言で表される型のコールバック関数のアドレスです。

```
typedef BOOL (*FCHK) (rANY pb, rCOLI co);
```

この関数は、行のアドレス pb と列情報 co を参照して得たチェック結果の可否を戻り値として返します。
実表項目に fchk を指定すると、その項目にチェック制約を設定することになります。計算項目への指定は無効です。
pa0, pa1, pa2 は、データ型によって異なる使われ方をする付加情報を示します。
数値項目の場合、pa0 には書式指定文字列を与えることができます。

```
{ CAL_F8B, 7, "rat", "料 率", NULL, CF_HZR, fcalLRA, NULL, (uANY) "%.4f", },
```

列挙型、コード型、集合型の場合、pa0 には ENMS または CODS 構造体への参照を与える必要があります。

```
{ TID_E1B, 6, "ltp", "区分", "契約区分", CF_DDL, NULL, NULL, (uANY) &esLTP, },
```

```
{ TID_C1B, 0, "ofc", "事業所", NULL, CF_DDL, NULL, NULL, (uANY) &dsOFC, },
```

外部キー制約項目の場合、pa1 に参照先の表とその主キー列のインデックスを与えます。

```
{ ASC(10), 9, "lid", "リース等 I D", NULL, CF_REF, NULL, NULL, 0, RKY(0, 0), },
```

外部キー制約項目があると、計算項目に CF_REF フラグを指定して pa1 に参照先の表とその任意の列のインデックスを与えることで外部参照項目を表に結合できます。

```
{ CAL_YMD, 0, "lbd", "リース開始", NULL, CF_REF, NULL, NULL, 0, RKY(0, 5), },
```

整数型の外部キー制約項目に対しては、参照先の表の主キーとは別の識別項目（代替キー）による入力の指定ができます。例えば、「社員番号」の代わりに「氏名」で、「科目コード」の代わりに「科目名」で入力するといったことです。代替キー入力型の場合、pa0 に代替キー列、pa1 に主キー列のインデックスを与えます。

```
{ TID_A4B, 0, "nam", "氏名", NULL, CF_REF, NULL, NULL, ALT(0, 2), RKY(0, 0), },
```

```
{ TID_A2B, 0, "acc", "科目", NULL, CF_REF|CF_DDL, NULL, NULL, ALT(0, 1), RKY(0, 0), },
```

参照先の表とその列のインデックスは、表インデックス id と列インデックス ic を次のように組み合わせて示します。

```
ALT(id, ic) または、RKY(id, ic)
```

pa2 は、既定のデータ型では使用していません。

既定のデータ型における付加情報の使い方を表にして示すと、以下のようになります。

コード	型	説明	pa0	pa1	pa2
TID_I1B	char	基本型	書式文字列（省略可）	—	—
TID_U1B	BYTE	基本型	書式文字列（省略可）	—	—
TID_I2B	short	基本型	書式文字列（省略可）	—	—
TID_U2B	WORD	基本型	書式文字列（省略可）	—	—
TID_I4B	int	基本型	書式文字列（省略可）	—	—
TID_U4B	UINT	基本型	書式文字列（省略可）	—	—
TID_I8B	INT64	基本型	書式文字列（省略可）	—	—
TID_U8B	UINT64	基本型	書式文字列（省略可）	—	—
TID_F4B	float	基本型	書式文字列（省略可）	—	—
TID_F8B	double	基本型	書式文字列（省略可）	—	—
STR(n)	char[n]	文字列	—	—	—
BIN(n)	BYTE[n]	バイト列	—	—	—
TID_HEX	UINT	16進数	書式文字列（省略可）	—	—
TID_M4B	int	金額等	書式文字列（省略可）	—	—
TID_M8B	double	金額等	書式文字列（省略可）	—	—
TID_YMD	UINT	年月日	書式文字列（省略可）	—	—
TID_YMN	UINT	年月	書式文字列（省略可）	—	—
TID_TIM	UINT	時刻	書式文字列（省略可）	—	—
TID_DTM	double	日付時刻	書式文字列（省略可）	—	—
TID_E1B	BYTE	列挙型	ENMS 構造体への参照	—	—
TID_E2B	WORD	列挙型	ENMS 構造体への参照	—	—
TID_E4B	UINT	列挙型	ENMS 構造体への参照	—	—
TID_C1B	BYTE	コード型	CODS 構造体への参照	—	—
TID_C2B	WORD	コード型	CODS 構造体への参照	—	—
TID_C4B	UINT	コード型	CODS 構造体への参照	—	—
TID_S1B	BYTE	集合型	ENMS 構造体への参照	—	—
TID_S2B	WORD	集合型	ENMS 構造体への参照	—	—
TID_S4B	UINT	集合型	ENMS 構造体への参照	—	—
TID_A1B	BYTE	代替キー	代替キー列指定	主キー列指定	—
TID_A2B	WORD	代替キー	代替キー列指定	主キー列指定	—
TID_A4B	UINT	代替キー	代替キー列指定	主キー列指定	—
ASC(n)	char[n]	文字列	—	—	—

pANY ColumnScan(pANY pv, rSTR st, rCOLI co); // 文字列から値を読み

pv が指すメモリ領域を、列情報 co によって定義された表の列項目であるとみなして、文字列からの読みを行います。

pSTR ColumnPrint(pSTR st, rANY pv, rCOLI co); // 値を文字列に書出

pv が指すメモリ領域を、列情報 co によって定義された表の列項目であるとみなして、文字列への書出を行います。

uANY ColumnList(I4B im, rCOLI co); // リスト項目情報

列挙型、コード型等の入力候補リストをもつ項目の列情報 co からリスト情報を取得します。

im が -1 のときリストの項目数、im が 0 ～ 項目数-1 のとき項目文字列へのポインタ値が返されます。

rANY VPtr(rANY pb, rCOLI co); // 項目のアドレス

行のアドレス pb と列情報 co から項目のアドレスを取得します。引数として co が与えられるコールバック関数内でのみ有効です。

HTBL CHTBL(rCOLI co); // 表ハンドル

列情報 co から表ハンドルを取得します。引数として co が与えられるコールバック関数内でのみ有効です。

6.3.2 表操作関数

以下の関数は、表オブジェクトを操作します。表オブジェクトは、表の定義情報と表内に保持されているデータを一括して管理するオブジェクトで、表ハンドル（HTBL 型）によって参照されます。

HTBL CreateTableX(pANY pv, I4B len, I4B cap, rTBLI ti); // 表の作成

定義情報を与えて表オブジェクトを作成します。

pv は、表データ領域のアドレスです。通常の使用法では、pv は NULL にしておきます。既存のメモリ領域をそのまま表データ領域にして静的な（容量可変でない）表を作成する場合のみ、pv を指定します。

len は、行数の初期値です。通常は 0 にします。

cap は、容量（最大行数）の初期値です。0 にしておくとも適当な値が設定されます。pv にアドレスを指定して作成した静的な表以外は容量可変です。現在の容量を超えて行を挿入しようとする、自動的にデータ領域の再割り当てが行われ、適当な刻み幅で容量が増加します。

ti は、表の定義情報を記述する TBLI 構造体への参照です。

TBLI 構造体は、以下のように宣言されています。

```
typedef struct {
    pCOLS cs; // COLS 構造体への参照
    pVIWS vs; // VIWS 構造体への参照
    rSTR nam; // 表名
    rSTR tit; // 表タイトル
    rSTR dsc; // 表説明
    I4B flg; // 表属性
    FTRG ftrg; // 表トリガ関数
    rSTR imp; // 表データのファイル名
    uANY ext; // 付加情報
    U4B rsz; // 行のサイズ
} TBLI;
```

```
typedef TBLI *pTBLI, *PTBLI;
```

```
typedef const TBLI *rTBLI, *PCTBLI;
```

cs は、表の列構成を定義する COLS 構造体への参照です。

COLS 構造体は、COLI 構造体の配列を記述します。

```
typedef struct {
    COLI* co; // COLI 構造体の配列の先頭アドレス
    I4B nc; // 配列の要素数
} COLS;
```

```
typedef COLS *pCOLS, *PCOLS;
```

```
typedef const COLS *rCOLS, *PCCOLS;
```

列のインデックスと実表項目の物理的レイアウトは、COLI 構造体の配列順によって決定されます。

vs は、システム定義のビューセット（表示形式の候補リスト）がある場合に使用する VIWS 構造体への参照です。

VIWS 構造体は、VIWI 構造体の配列を記述します。

```
typedef struct {
    rSTR nam; // 表示形式名
    I4B fc; // 固定列数
    rSTR arr; // 列名リスト
    rSTR cnd; // 絞り込み条件式
    rSTR srt; // 並べ替え条件式
} VIWI;
```

```
typedef VIWI *pVIWI, *PVIWI;
```

```
typedef const VIWI *rVIWI, *PCVIWI;
```

```
typedef struct {
    pVIWI vi; // VIWI 構造体の配列の先頭アドレス
    I4B ni; // 配列の要素数
} VIWS;
```

```
typedef VIWS *pVIWS, *PVIWS;
```

```
typedef const VIWS *rVIWS, *PCVIWS;
```

ビューについては、6.3.4 節で扱います。

nam は、表名にする 16byte 以内の英字と数字の（C 言語の識別子形式の）文字列です。

tit は、表タイトルにする 32byte 以内の任意文字列です。省略すると、nam が tit として使われます。

dsc は、表説明にする 64byte 以内の任意文字列です。省略すると、tit が dsc として使われます。

`flg` は、以下のフラグ値を組み合わせて表の属性を示します。

- `TF_RDO` 読取専用
- `TF_FIX` 行編集（行の追加／削除）を不可にする
- `TF_IMP` 他のデータベースの表（読取専用になる）
- `TF_CYP` 暗号化（ファイル書出時に乱数でマスクする、読込時には自動的に復号化する）
- `TF_DTR` 表トリガ関数を無効にする
- `TF_NOA` アライメント制約のチェックをしない
- `TF_STA` 静的な表（データ領域の再割り当てを行わない）
- `TF_STB` 内部使用のため予約
- `TF_VIW` 内部使用のため予約
- `TF_TMP` 内部使用のため予約
- `TF_OTR` 内部使用のため予約

`ftrg` は、次の宣言で表される型のコールバック関数（表トリガ関数）のアドレスです。（5.5節参照）

```
typedef BOOL (*FTRG) (HTBL tb, I4B msg, I4B ir, pANY pb);
```

`imp` は、表データのファイル名とする文字列です。省略すると、既定のファイル名（**表名.BTB** または **表名.BTC**）が用いられます。フルパス名を与えることで他サイトの表を参照することもできます。

`ext` は、任意の付加情報を表します。

`rsz` は、1行分の表データと等価であるべき構造体のサイズを与えて一致をチェックするときに用います。

戻り値は、成功時は有効な表ハンドル、失敗時は `NULL` です。

`HTBL CreateTable(I4B cap, rTBLI ti); // 表の作成`

定義情報を与えて表オブジェクトを作成します。`CreateTableX(NULL, 0, cap, ti)` と同じです。

`void DeleteTable(HTBL tb); // 表の削除`

表オブジェクトを削除します。

`I4B NTableKeys(HTBL tb); // 主キーの数`

`CF_KEY` 属性のついた列の数を返します。

`I4B NTableRefs(HTBL tb, pI4B pi); // 外部キー制約項目の数`

実表項目でかつ `CF_REF` 属性のついた列の数を返します。配列 `pi` には、参照先の表のインデックスが格納されます。

`HTBL CreateCompatibleTable(HTBL tb, I4B cap); // 同型表の作成`

表 `tb` と同型の（定義情報が等しい）空の表を作成します。作成された表は、元の表と定義情報を共有しているため、元の表を削除すると使用できなくなります。

`BOOL IsCompatible(HTBL tb, HTBL tx); // 表が同型`

2つの表の定義情報を比較して表が同型であるかどうか判定します。表が同型であるとは、実表項目の数が等しく、それぞれの実表項目のオフセット／データ型／列名が一致することをいいます。

`HTBL DuplicateTable(HTBL tb); // 表の複製`

表 `tb` を複製（同型表を作成し、データ内容をコピー）します。複製された表は、元の表と定義情報を共有しているため、元の表を削除すると使用できなくなります。

`I4B TableRowSize(HTBL tb); // 行のサイズ`

表 `tb` の行のサイズを返します。

`I4B NTableRows(HTBL tb); // 行数`

表 `tb` の行数を返します。

`pANY TablePtr(HTBL tb); // 表データの先頭アドレス`

表データ領域の先頭アドレスを返します。`TableRow(tb, 0)` と同じです。

`pANY TableRow(HTBL tb, I4B ir); // 行のアドレス`

表 `tb` の `ir` 行のアドレスを返します。行インデックス `ir` は、0～行数-1の範囲で与えます。

`I4B TableRowIndex(HTBL tb, rANY pb); // 行のアドレスから行インデックスを求める`

行のアドレス `pb` から行インデックスを求めます。

I4B TableCapacity(HTBL tb); // 最大行数
表の容量（最大行数）を返します。

BOOL ChangeTableCapacity(HTBL tb, I4B cap); // 最大行数の変更
表の容量（最大行数）を変更します。

rSTR TableName(HTBL tb); // 表名
表名文字列を返します。

void SetTableTitle(HTBL tb, rSTR tit); // 表タイトルの設定
表タイトル文字列を設定します。

rSTR TableTitle(HTBL tb); // 表タイトル
表タイトル文字列を返します。

void SetTableDescription(HTBL tb, rSTR dsc); // 表説明の設定
表説明文字列を設定します。

rSTR TableDescription(HTBL tb); // 表説明
表説明文字列を返します。

void SetTableFlags(HTBL tb, I4B flg); // 表属性の設定
表の属性（フラグ値の組み合わせ）を設定します。

I4B TableFlags(HTBL tb); // 表属性
表の属性（フラグ値の組み合わせ）を返します。

void SetTableFileName(HTBL tb, rSTR nam); // 表ファイル名の設定
表データのファイル名を設定します。

rSTR TableFileName(HTBL tb); // 表ファイル名
表データのファイル名を返します。

void SetTableExtra(HTBL tb, uANY pa); // 表の付加情報の設定
表の付加情報を設定します。

uANY TableExtra(HTBL tb); // 表の付加情報
表の付加情報を返します。

BOOL DefaultTableTrigger(HTBL tb, I4B msg, I4B ir, pANY pb); // 既定のトリガ関数
既定の表トリガ関数を呼び出します。

BOOL TableTrigger(HTBL tb, I4B msg, I4B ir, pANY pb); // トリガ関数呼び出し
表トリガ関数を呼び出します。

BOOL EnableTableTrigger(HTBL tb); // トリガ有効化
BOOL DisableTableTrigger(HTBL tb); // トリガ無効化
表トリガを有効化／無効化します。
戻り値は、関数を呼び出す前に有効であったか無効であったかを TRUE / FALSE で示します。

I4B NTableColumns(HTBL tb); // 列数
表 tb の列数を返します。

rCOLI TableColumnInfo(HTBL tb, I4B ic); // 列情報
表 tb の ic 列の列情報（COLI 構造体への参照）を返します。

void SetTableColumnWidth(HTBL tb, I4B ic, I4B wd); // 列幅の設定
I4B TableColumnWidth(HTBL tb, I4B ic); // 列幅
列の表示幅は、0 ～ 254 の範囲で指定します。

void SetTableColumnTextAlign(HTBL tb, I4B ic, I4B al); // 列の文字揃えコードの設定
I4B TableColumnTextAlign(HTBL tb, I4B ic); // 列の文字揃えコード
文字揃えコードは、0 が左寄せ、1 が中央揃え、2 が右寄せです。

I4B TableColumnOffset(HTBL tb, I4B ic); // 列のオフセット
表 tb の ic 列の項目の行頭からのオフセット値を返します。計算項目のオフセット値は 0 です。

rSTR TableColumnName(HTBL tb, I4B ic); // 列名
列名文字列を返します。

I4B TableColumnIndex(HTBL tb, rSTR nam); // 列名から列インデックスを求める
列名 nam から列インデックスを求めます。

void SetTableColumnTitle(HTBL tb, I4B ic, rSTR tit); // 列タイトルの設定
列タイトル文字列を設定します。

rSTR TableColumnTitle(HTBL tb, I4B ic); // 列タイトル
列タイトル文字列を返します。

void SetTableColumnDescription(HTBL tb, I4B ic, rSTR dsc); // 列説明の設定
列説明文字列を設定します。

rSTR TableColumnDescription(HTBL tb, I4B ic); // 列説明
列説明文字列を返します。

void SetTableColumnFlags(HTBL tb, I4B ic, I4B flg); // 列属性の設定
列の属性（フラグ値の組み合わせ）を設定します。

I4B TableColumnFlags(HTBL tb, I4B ic); // 列属性
列の属性（フラグ値の組み合わせ）を返します。

void SetTableColumnExtra(HTBL tb, I4B ic, uANY pa); // 列の付加情報の設定
列の付加情報を設定します。

uANY TableColumnExtra(HTBL tb, I4B ic); // 列の付加情報
列の付加情報を返します。

BOOL SetTableCellValue(HTBL tb, I4B ir, I4B ic, rANY pv); // 値の設定
表 tb の ir 行 ic 列の項目に pv のポイント先の値を格納します。pv は項目のデータ型に応じたポインタ型でなければなりません。

pANY GetTableCellValue(pANY pv, HTBL tb, I4B ir, I4B ic); // 値の取得
表 tb の ir 行 ic 列の項目から pv のポイント先に値を取得します。pv は項目のデータ型に応じたポインタ型でなければなりません。

BOOL ScanTableCellValue(HTBL tb, I4B ir, I4B ic, rSTR st); // 文字列から読込
表 tb の ir 行 ic 列の項目に文字列 st から値を読み込んで格納します。

pSTR PrintTableCellValue(pSTR st, HTBL tb, I4B ir, I4B ic); // 文字列に書出
表 tb の ir 行 ic 列の項目の値を文字列 st に書き出します。

BOOL ClearTableRow(HTBL tb, I4B ir); // 行のクリア
表 tb の ir 行を初期化して空行にします。ClearTableRows(tb, ir, 1) と同じです。

BOOL ClearTableRows(HTBL tb, I4B ir, I4B nr); // 範囲のクリア
表 tb の ir 行から nr 行分を初期化して空行にします。

BOOL InsertTableRow(HTBL tb, I4B ir); // 行の挿入
表 tb の ir 行の前に初期化された空行を 1 行挿入します。InsertTableRows(tb, ir, 1) と同じです。

BOOL InsertTableRows(HTBL tb, I4B ir, I4B nr); // 範囲の挿入
表 tb の ir 行の前に初期化された空行を nr 行挿入します。

BOOL DeleteTableRow(HTBL tb, I4B ir); // 行の削除
表 tb の ir 行を削除します。DeleteTableRows(tb, ir, 1) と同じです。

BOOL DeleteTableRows(HTBL tb, I4B ir, I4B nr); // 範囲の削除
表 tb の ir 行から nr 行分を削除します。

BOOL AddTableRow(HTBL tb); // 行の追加
表 tb の末尾に初期化された空行を 1 行追加します。

pANY NewTableRow(HTBL tb); // 追加行
表 tb の末尾に初期化された空行を 1 行追加し、その行のアドレスを返します。

HTBL CopyTable(HTBL tb, HTBL tx); // 表のコピー
表 tx のデータ内容を表 tb にコピーします。tb と tx は同型の表でなければなりません。戻り値は tb です。

HTBL MergeTableX(HTBL tb, HTBL tx); // 表のマージ
表 tx のデータ内容を表 tb に併合します。tb と tx は同型の表でなければなりません。
この関数は行の重複 (tb と tx に同じ主キー値を持った行があること) をチェックしません。行の重複を考慮して表をマージするときは、MergeTable 関数を使用します。

I4B CompareTable(HTBL tb, HTBL tx); // 表の比較
2 つの同型の表 tb と tx のデータ内容が (行の物理的な並び順も含め完全に) 等しいかどうか判定します。
戻り値は、データ内容が等しいとき 0、等しくないとき 0 以外の値となります。

I4B SearchTableRowX(HTBL tb, I4B i0, FSEL fsel, uANY pa); // サーチ
表 tb 内で、i0 行目から末尾に向かって、選択関数 fsel で指定される絞り込み条件に合う行を検索します。戻り値は、条件に合う行が見つければその行のインデックス、見つからなければ -1 です。
fsel は、次の宣言で表される型のコールバック関数のアドレスです。
typedef BOOL (*FSEL)(rANY pb, uANY pa);
pb は行のアドレス、pa は任意の付加情報、戻り値は選択する／しないを TRUE / FALSE で示します。

I4B CountTableRowsX(HTBL tb, FSEL fsel, uANY pa); // カウント
表 tb 内で、選択関数 fsel で指定される絞り込み条件に合う行をカウントします。戻り値は、条件に合ってカウントされた行数です。

I4B SelectTableRowsX(HTBL tb, FSEL fsel, uANY pa); // 絞り込み
表 tb に対し、選択関数 fsel で指定される絞り込み条件に合う行のみ残り、合わない行を削除する選択操作を行います。戻り値は、条件に合って残った行数です。

HTBL SelectCopyTableX(HTBL tb, HTBL tx, FSEL fsel, uANY pa); // 絞り込みコピー
表 tx から、選択関数 fsel で指定される絞り込み条件に合う行のみを選択して表 tb にコピーします。tb と tx は同型の表でなければなりません。

HTBL SelectMergeTableX(HTBL tb, HTBL tx, FSEL fsel, uANY pa); // 絞り込みマージ
表 tx から、選択関数 fsel で指定される絞り込み条件に合う行のみを選択し、表 tb に対し同絞り込み条件に合う行を削除する選択操作を行った結果と併合します。tb と tx は同型の表でなければなりません。

I4B AggregateTableX(HTBL tb, FSEL fsel, uANY ps, FAGG fagg, uANY pa); // 集計
表 tb 内で、選択関数 fsel で指定される絞り込み条件に合う行に対し、集計関数 fagg を適用することで集計操作を行います。ps は fsel に与えられる付加情報、pa は fagg に与えられる付加情報です。
fagg は、次の宣言で表される型のコールバック関数のアドレスです。
typedef BOOL (*FAGG)(rANY pb, uANY pa);
pb は行のアドレス、pa は任意の付加情報、戻り値はカウントする／しないを TRUE / FALSE で示します。

void SortTableRowsX(HTBL tb, FSRT fcmp, uANY pa); // 並べ替え

比較関数 **fcmp** で指定される並べ替え条件によって、表 **tb** 内の行を物理的に並べ替えます。

fcmp は、次の宣言で表される型のコールバック関数のアドレスです。

typedef I4B (*FSRT)(rANY p1, rANY p2, uANY pa);

p1, p2 は行のアドレス、**pa** は任意の付加情報、戻り値は、比較結果が等しいとき 0、「>」のとき正、「<」のとき負です。

I4B SearchTableCellValue(HTBL tb, I4B i0, I4B ic, rANY pv); // 値のサーチ

表 **tb** の **ic** 列を **i0** 行目から末尾に向かって走査し、**pv** のポイント先の値を検索します。**pv** は項目のデータ型に応じたポインタ型でなければなりません。戻り値は、値が見つければその行のインデックス、見つからなければ -1 です。

I4B SearchCellText(HTBL tb, I4B i0, I4B ic, rSTR st, I4B flg); // 文字列のサーチ

表 **tb** の **ic** 列を **i0** 行目から末尾に向かって走査し、文字列 **st** を検索します。**flg** は、以下のフラグ値の組み合わせで検索モードを指定します。

STX_UPR 大文字と小文字を区別しない

STX_ZEN 全角と半角を区別しない

STX_DSP 空白を無視する

0 部分一致 (省略時既定値)

STX_ALL 全体一致

STX_LFT 前方一致

STX_RGT 後方一致

戻り値は、文字列が見つければその行のインデックス、見つからなければ -1 です。

I4B SearchAlt(HTBL tb, I4B ic, rSTR st); // 代理キーによるサーチ

表 **tb** の **ic** 列を走査し、曖昧検索モードで (大文字と小文字の違い、全角と半角の違い、空白の有無を無視して) 文字列 **st** を検索し、結果が一意 (合致する行が 1 行だけ) のとき、その行インデックスを返します。合致する行がないか、結果が一意でなければ、-1 を返します。

BOOL CheckTable(HTBL tb); // 表データのチェック

表 **tb** の各行に対し、表トリガを **BEFOREUPDATE** フェーズで呼び出します。

戻り値は、全ての行でトリガ関数が **TRUE** を返したとき **TRUE**、1 つでも **FALSE** を返す行があれば **FALSE** となります。

BOOL LoadBTB(HTBL tb); // 読込

表データをファイルから読み込みます。

BOOL SaveBTB(HTBL tb); // 保存

表データをファイルへ書き出します。

6.3.3 行バッファ操作関数

以下の関数は、行バッファオブジェクトを操作 (または行バッファオブジェクトを使って表オブジェクトを操作) します。行バッファオブジェクトは、1 行分のデータを保持するオブジェクトで、行バッファハンドル (**HBUF** 型) によって参照されます。

HBUF CreateRowBufX(pANY pv, rTBLI ti); // 行バッファの作成

行バッファ領域のアドレス **pv** と表定義情報 **ti** を与えて行バッファオブジェクトを作成します。

行バッファ領域は、表の 1 行分のデータを格納できるサイズでなければなりません。

戻り値は、成功時は有効な行バッファハンドル、失敗時は **NULL** です。

HBUF CreateRowBuf(HTBL tb); // 行バッファの作成

表ハンドル **tb** を与えてその表に対する行バッファオブジェクトを作成します。作成された行バッファは、元の表と定義情報を共有しているため、元の表を削除すると使用できなくなります。

void DeleteRowBuf(HBUF rb); // 行バッファの削除

行バッファオブジェクトを削除します。

HBUF DuplicateRowBuf(HBUF rb); // 行バッファの複製

行バッファ **rb** を複製 (定義情報を共有し、データ内容をコピー) します。

HTBL RowBufHTBL (HBUF rb); // 元の表

行バッファ **rb** の元の表の表ハンドルを返します。 **CreateRowBufX** 関数で作成した行バッファの場合、 (**HTBL**) **rb** が返されます。

pANY RowBufPtr (HBUF rb); // バッファのアドレス

行バッファ領域のアドレスを返します。

HBUF MakeRowBuf (HTBL tb, pANY pb); // 行アドレスを一時的に行バッファとして扱う

表の 1 行分のメモリ領域を指すポインタ値を一時的に行バッファとして扱います。戻り値が参照するオブジェクトは、 **ualloc** の循環バッファ中に割り当てられます。この関数の戻り値はその場で使い捨てにしなければなりません。

HBUF TableRowBuf (HTBL tb, I4B ir); // 表中の行を一時的に行バッファとして扱う

表 **tb** の **ir** 行を一時的に行バッファとして扱います。 **MakeRowBuf (tb, TableRow (tb, ir))** と同じです。

HBUF NewTableRowBuf (HTBL tb); // 追加行を一時的に行バッファとして扱う

表 **tb** の末尾に初期化された空行を 1 行追加し、一時的に行バッファとして扱います。

MakeRowBuf (tb, NewTableRow (tb)) と同じです。

BOOL SetRowBufCellValue (HBUF rb, I4B ic, rANY pv); // 値の設定

行バッファ **rb** の **ic** 列の項目に **pv** のポイント先の値を格納します。 **pv** は項目のデータ型に応じたポインタ型でなければなりません。

pANY GetRowBufCellValue (pANY pv, HBUF rb, I4B ic); // 値の取得

行バッファ **rb** の **ic** 列の項目から **pv** のポイント先に値を取得します。 **pv** は項目のデータ型に応じたポインタ型でなければなりません。

BOOL ScanRowBufCellValue (HBUF rb, I4B ic, rSTR st); // 文字列から読込

行バッファ **rb** の **ic** 列の項目に文字列 **st** から値を読み込んで格納します。

pSTR PrintRowBufCellValue (pSTR st, HBUF rb, I4B ic); // 文字列に書出

行バッファ **rb** の **ic** 列の項目の値を文字列 **st** に書き出します。

BOOL ClearRowBuf (HBUF rb); // 行バッファのクリア

行バッファ領域を初期化して空行にします。

HBUF CopyRowBuf (HBUF rb, HBUF rx); // 行バッファから行バッファへのコピー

行バッファ **rx** から行バッファ **rb** にデータ内容をコピーします。 **rb** と **rx** は同型でなければなりません。

戻り値は **rb** です。

BOOL SetTableRow (HTBL tb, I4B ir, HBUF rb); // 行の設定

表 **tb** の **ir** 行に行バッファ **rb** のデータ内容をコピーします。 **tb** と **rb** は同型でなければなりません。

HBUF GetTableRow (HBUF rb, HTBL tb, I4B ir); // 行の取得

表 **tb** の **ir** 行のデータ内容を行バッファ **rb** にコピーします。 **tb** と **rb** は同型でなければなりません。

BOOL SetNewTableRow (HTBL tb, HBUF rb); // 行の追加

表 **tb** の末尾に行を 1 行追加し、そこへ行バッファ **rb** のデータ内容をコピーします。 **tb** と **rb** は同型でなければなりません。

HBUF CopyValues (HBUF rb, HBUF rx); // 値のコピー

行バッファ **rx** から行バッファ **rb** に項目単位でデータ内容をコピーします。 **rb** と **rx** は同型でなくても構いません。

rb と **rx** に同じ列名の項目があれば、その項目間（**rb** 側は実表項目に限る）でデータがコピーされます。

項目のデータ型が異なっても、可能な限り値を保つようにデータのコピーが行われます。

HBUF CopyKey (HBUF rb, HBUF rx); // 外部キーのコピー

行バッファ **rx** の外部キー項目を行バッファ **rb** の主キー項目にコピーします。この関数は、**rx** の元の表が **rb** の元の表を外部キー参照している場合にのみ使用できます。

I4B CompareRowBuf(HBUF rb, HBUF rx); // 行バッファの比較

2つの同型の行バッファ rb と rx のデータ内容が等しいかどうか判定します。
戻り値は、データ内容が等しいとき 0、等しくないとき 0 以外の値となります。

I4B SearchKey(HTBL tb, HBUF rb); // 主キーによるサーチ

表 tb 内で、行バッファ rb の主キー項目と一致する行を検索します。tb と rb は同型でなければなりません。
戻り値は、一致する行が見つければその行のインデックス、見つからなければ -1 です。

HTBL MergeTable(HTBL tb, HTBL tx); // 表のマージ

表 tx のデータ内容を表 tb に併合します。tb と tx は同型の表でなければなりません。
tb と tx に重複（同じ主キー値を持った行）があると、tb の行が tx の行で上書きされます。

I4B UniqueKey(HTBL tb, I4B ic, U4B h0, U4B hs); // キー値の自動採番

表 tb の ic 列（整数型の項目に限る）を走査して、表内で一意なキー値（h0 以上 hs 未満でまだ使われていない値）を返します。ハッシュ表のサイズ hs - h0 は、8 の倍数より 1 だけ小さい数としてください。

HTBL CreateSummaryV(HTBL tb, rSTR grp, rSTR cnd, rSTR srt); // 集計表の作成

表 tb に対し、集計項目リスト grp、絞り込み条件式 cnd、並べ替え条件式 srt を与えて集計表を作成します。
集計項目リストとは、以下に示す集計項目をコンマで区切って並べた文字列です。

列名 そのまま表示する項目（代替書式指定不可）
groupby 列名 集計単位とする項目（代替書式指定不可）
sum(列名) 項目の合計値を求める
avg(列名) 項目の平均値を求める
max(列名) 項目の最大値を求める
min(列名) 項目の最小値を求める
count(列名) 項目の個数を求める

上記指定の後には、代替タイトルの指定
as 'タイトル'

および、代替書式の指定

dec dec(w) dec(w, s) dec(u) dec(w, u) dec(w, s, u)
cdec cdec(w) cdec(w, s) cdec(u) cdec(w, u) cdec(w, s, u)

をオプションとして加えることができます。

絞り込み条件式とは、列名、比較演算子、値をこの順に並べた式を and と or で結合した文字列です。

比較演算子は、以下のどれかとします。

= 等しい
<> 等しくない
< より小
<= 以下
> より大
>= 以上
mlike 含む
notmlike 含まない

and は or よりも結合順位が高いものとし、結合順位を変えるには括弧を用います。

絞り込み条件式を空文字列にすると、全行が選択対象となります。

並べ替え条件式とは、列名をコンマで区切って並べた文字列です。

列名の後には、以下のような並べ替え順序を指定するオプションが付けられます。

asc 昇順（無指定時の既定順序）
desc 降順
zerolast 昇順、ただし空欄は最後
desc zerofirst 降順、ただし空欄は最初
parenlast 昇順、ただし括弧で始まる文字列は最後
desc parenfirst 降順、ただし括弧で始まる文字列は最初
zerolast parenlast 昇順、ただし空欄と括弧で始まる文字列は最後
desc zerofirst parenfirst 降順、ただし空欄と括弧で始まる文字列は最初

並べ替え条件式を空文字列にすると、行の並び順はメモリ内での物理的な並び順となります。

BOOL IsConflict(HTBL tb, HTBL tx); // 競合のチェック

2つの同型の表 tb と tx のデータ内容が競合して（主キーで識別される同じ行の内容が異なって）いるかどうか判定します。戻り値は、競合している／いないを TRUE / FALSE で示します。

BOOL IsDependent(HTBL tb, I4B i1, I4B i2, rANY p1, rANY p2); // 従属関係のチェック

表 tb の列 i1, i2 が同じデータ型の項目（実表項目に限る）で何らかの従属関係（例えば、「子」と「親」）を表している場合、p1 の指す値が p2 の指す値に従属しているかどうか判定します。

戻り値は、従属している／いないを TRUE / FALSE で示します。

6.3.4 ビュー操作関数

以下の関数は、ビューオブジェクトを操作（またはビューオブジェクトを使って表オブジェクトを操作）します。

ビューオブジェクトは、自由な配列順で行と列にアクセスできるように表オブジェクトをラッピングしたオブジェクトで、ビューハンドル（HVIW 型）によって参照されます。

HVIW CreateViewX(HTBL tb, rIDX pa, FSEL fsel, uANY ps, FSRT fcmp, uANY pc); // ビューの作成

表 tb に対し、列インデックスのリスト pa で列の表示順を、選択関数 fsel で絞り込み条件を、比較関数 fcmp で並べ替え条件を指定してビューオブジェクトを作成します。ps は fsel に与えられる付加情報、pc は fcmp に与えられる付加情報です。

列インデックスのリストは、IDX 構造体への参照で指定します。戻り値は、成功時は有効なビューハンドル、失敗時は NULL です。

作成されたビューオブジェクトは、元の表と定義情報およびデータ内容を共有しているため、元の表が削除されると使用できなくなります。一旦ビューオブジェクトを作成したら、表に対する行の挿入や削除等の操作は、必ずビューオブジェクトを通して行わなければなりません。ビューを持つ表に対し、ビューを通さずに行の挿入や削除を行うと、ビューオブジェクト内の行インデックス情報が破壊されます。

ビューオブジェクトは、元の表の属性（読取専用／行編集不可等）を引き継ぎます。

HVIW CreateView(HTBL tb, rSTR arr, rSTR cnd, rSTR srt); // ビューの作成

表 tb に対し、列名リスト arr、絞り込み条件式 cnd、並べ替え条件式 srt を与えてビューオブジェクトを作成します。

列名リストとは、列名をコンマで区切って並べた文字列です。

列名リストを空文字列にすると、全ての実表項目を列インデックス順に指定したのと同じになります。また、列名リストを半角アスタリスク 1 文字「*」で与えると、計算項目を含む全ての項目を列インデックス順に指定したのと同じになります。

void DeleteView(HVIW vw); // ビューの削除

ビューオブジェクトを削除します。

HTBL ViewHTBL(HVIW vw); // 元の表

ビュー vw の元の表の表ハンドルを返します。

I4B NViewRows(HVIW vw); // 行数

ビュー上の行数を返します。

pANY ViewRow(HVIW vw, I4B ir); // 行のアドレス

ビュー vw の ir 行のアドレスを返します。

I4B ViewRowIndex(HVIW vw, I4B ir); // ビュー上の行インデックスから実表上の行インデックスを求める

ビュー上の行インデックスから実表上の行インデックスを求めます。

I4B NViewColumns(HVIW vw); // 列数

ビュー上の列数を返します。

rCOLI ViewColumnInfo(HVIW vw, I4B ic); // 列情報

ビュー vw の ic 列の列情報（COLI 構造体への参照）を返します。

void SetViewColumnWidth(HVIW vw, I4B ic, I4B wd); // 列幅の設定

I4B ViewColumnWidth(HVIW vw, I4B ic); // 列幅

I4B ViewColumnWidthF(HVIW vw, I4B ic); // 列幅

ビュー上の列インデックスを指定して列の表示幅を設定／取得します。

ViewColumnWidthF 関数は、ic が範囲外でもエラーにせず、列幅の既定値を返します。

void SetViewColumnTextAlign(HVIW vw, I4B ic, I4B al); // 列の文字揃えコードの設定
I4B ViewColumnTextAlign(HVIW vw, I4B ic); // 列の文字揃えコード
I4B ViewColumnTextAlignF(HVIW vw, I4B ic); // 列の文字揃えコード
ビュー上の列インデックスを指定して列の文字揃えコードを設定／取得します。
ViewColumnTextAlignF 関数は、ic が範囲外でもエラーにせず、文字揃えコードの既定値を返します。

rSTR ViewColumnName(HVIW vw, I4B ic); // 列名
ビュー上の列インデックスに対応する列名文字列を返します。

I4B ViewColumnIndex(HVIW vw, I4B ic); // ビュー上の列インデックスから実表上の列インデックスを求める
ビュー上の列インデックスから実表上の列インデックスを求めます。

void SetViewColumnTitle(HVIW vw, I4B ic, rSTR tit); // 列タイトルの設定
rSTR ViewColumnTitle(HVIW vw, I4B ic); // 列タイトル
void SetViewColumnDescription(HVIW vw, I4B ic, rSTR dsc); // 列説明の設定
rSTR ViewColumnDescription(HVIW vw, I4B ic); // 列説明
rSTR ViewColumnDescriptionR(HVIW vw, I4B ic); // 列説明
ビュー上の列インデックスを指定して列タイトル／列説明を設定／取得します。
ViewColumnDescriptionR 関数は、列説明文字列末尾の「(&M)」等を削除して返します。

void SetViewColumnFlags(HVIW vw, I4B ic, I4B flg); // 列属性の設定
I4B ViewColumnFlags(HVIW vw, I4B ic); // 列属性
I4B ViewColumnFlagsF(HVIW vw, I4B ic); // 列属性
ビュー上の列インデックスを指定して列属性を設定／取得します。
ViewColumnFlagsF 関数は、ic が範囲外でもエラーにせず、既定値 0 を返します。

void SetViewColumnExtra(HVIW vw, I4B ic, uANY pa); // 列の付加情報の設定
uANY ViewColumnExtra(HVIW vw, I4B ic); // 列の付加情報
ビュー上の列インデックスを指定して列の付加情報を設定／取得します。

BOOL SetViewCellValue(HVIW vw, I4B ir, I4B ic, rANY pv); // 値の設定
ビュー vw の ir 行 ic 列の項目に pv のポイント先の値を格納します。pv は項目のデータ型に応じたポインタ型でなければなりません。

pANY GetViewCellValue(pANY pv, HVIW vw, I4B ir, I4B ic); // 値の取得
ビュー vw の ir 行 ic 列の項目から pv のポイント先に値を取得します。pv は項目のデータ型に応じたポインタ型でなければなりません。

BOOL ScanViewCellValue(HVIW vw, I4B ir, I4B ic, rSTR st); // 文字列から読込
BOOL ScanViewCellValueF(HVIW vw, I4B ir, I4B ic, rSTR st); // 文字列から読込
ビュー vw の ir 行 ic 列の項目に文字列 st から値を読み込んで格納します。
ScanViewCellValueF 関数は、ir, ic が範囲外でもエラーにしません。

pSTR PrintViewCellValue(pSTR st, HVIW vw, I4B ir, I4B ic); // 文字列に書出
pSTR PrintViewCellValueF(pSTR st, HVIW vw, I4B ir, I4B ic, I4B flg); // 文字列に書出
ビュー vw の ir 行 ic 列の項目の値を文字列 st に書き出します。
PrintViewCellValueF 関数は、ir, ic が範囲外でもエラーにせず、見出し文字列を返します。
flg は、以下のフラグ値を組み合わせて見出し文字列の内容を指定します。

- PVF_ZBI インデックスを 0 から始める (省略時既定値)
- PVF_OBI インデックスを 1 から始める
- PVF_NAM 列見出しを列名にする (省略時既定値)
- PVF_TIT 列見出しを列タイトルにする
- PVF_IDX 列見出しを列インデックスにする

BOOL ClearViewRow(HVIW vw, I4B ir); // 行のクリア
ビュー vw の ir 行を初期化して空行にします。ClearViewRows(vw, ir, 1) と同じです。

BOOL ClearViewRows(HVIW vw, I4B ir, I4B nr); // 範囲のクリア
ビュー vw の ir 行から nr 行分を初期化して空行にします。

BOOL InsertViewRow(HVIW vw, I4B ir); // 行の挿入

ビュー **vw** の **ir** 行の前に初期化された空行を 1 行挿入します。InsertViewRows(vw, ir, 1) と同じです。

BOOL InsertViewRows(HVIW vw, I4B ir, I4B nr); // 範囲の挿入

ビュー **vw** の **ir** 行の前に初期化された空行を **nr** 行挿入します。

BOOL DeleteViewRow(HVIW vw, I4B ir); // 行の削除

ビュー **vw** の **ir** 行を削除します。DeleteViewRows(vw, ir, 1) と同じです。

BOOL DeleteViewRows(HVIW vw, I4B ir, I4B nr); // 範囲の削除

ビュー **vw** の **ir** 行から **nr** 行分を削除します。

BOOL AddViewRow(HVIW vw); // 行の追加

ビュー **vw** の末尾に初期化された空行を 1 行追加します。

pANY NewViewRow(HVIW vw); // 追加行

ビュー **vw** の末尾に初期化された空行を 1 行追加し、その行のアドレスを返します。

void ArrangeViewColumnsX(HVIW vw, rIDX pa); // 列構成の設定

ビュー **vw** の表示項目を列インデックスのリスト **pa** で設定します。

I4B SearchViewRowX(HVIW vw, I4B i0, FSEL fsel, uANY pa); // サーチ

ビュー **vw** 内で、**i0** 行目から末尾に向かって、選択関数 **fsel** で指定される絞り込み条件に合う行を検索します。戻り値は、条件に合う行が見つければその行のインデックス、見つからなければ -1 です。

I4B CountViewRowsX(HVIW vw, FSEL fsel, uANY pa); // カウント

ビュー **vw** 内で、選択関数 **fsel** で指定される絞り込み条件に合う行をカウントします。戻り値は、条件に合ってカウントされた行数です。

I4B SelectViewRowsX(HVIW vw, FSEL fsel, uANY pa); // 絞り込み

ビュー **vw** 上に、元の表から、選択関数 **fsel** で指定される絞り込み条件に合う行のみを選択します。戻り値は、条件に合って選択された行数です。

この関数は、ビューの表示条件を変更するだけです。SelectTableRowsX 関数とは違い、元の表の行が削除されることはありません。

void SortViewRowsX(HVIW vw, FSRT fcmp, uANY pa); // 並べ替え

比較関数 **fcmp** で指定される並べ替え条件によってビュー **vw** 上の行を並べ替えます。

この関数は、ビューの表示条件を変更するだけです。SortTableRowsX 関数とは違い、元の表が物理的に並べ替えられることはありません。

void ArrangeViewColumns(HVIW vw, rSTR arr); // 列構成の設定

ビュー **vw** の表示項目を列名リスト **arr** で設定します。

I4B SearchViewRow(HVIW vw, I4B i0, rSTR cnd); // サーチ

ビュー **vw** 内で、**i0** 行目から末尾に向かって、絞り込み条件式 **cnd** で指定される絞り込み条件に合う行を検索します。戻り値は、条件に合う行が見つければその行のインデックス、見つからなければ -1 です。

I4B CountViewRows(HVIW vw, rSTR cnd); // カウント

ビュー **vw** 内で、絞り込み条件式 **cnd** で指定される絞り込み条件に合う行をカウントします。戻り値は、条件に合ってカウントされた行数です。

I4B SelectViewRows(HVIW vw, rSTR cnd); // 絞り込み

ビュー **vw** 上に、元の表から、絞り込み条件式 **cnd** で指定される絞り込み条件に合う行のみを選択します。戻り値は、条件に合って選択された行数です。

この関数は、ビューの表示条件を変更するだけです。元の表の行が削除されることはありません。

```
void SortViewRows(HVIW vw, rSTR srt); // 並べ替え
```

並べ替え条件式 `srt` によってビュー `vw` 上の行を並べ替えます。

この関数は、ビューの表示条件を変更するだけです。元の表が物理的に並べ替えられることはありません。

```
void SortTableRowsByView(HVIW vw); // ビュー上の並び順による並べ替え
```

ビュー `vw` 上の現在の並び順に従って元の表を物理的に並べ替えます。ビュー上に選択されていない行は、削除されるのではなく、末尾に送られます。

```
HTBL CreateSummary(HVIW vw, rSTR grp); // 集計表の作成
```

ビュー `vw` に対し、集計項目リスト `grp` を与えて集計表を作成します。

```
I4B SearchViewCellValue(HVIW vw, I4B i0, I4B ic, rANY pv); // 値のサーチ
```

ビュー `vw` の `ic` 列を `i0` 行目から末尾に向かって走査し、`pv` のポイント先の値を検索します。

使用法は、`SearchTableCellValue` 関数と同じです。

```
I4B SearchViewCellText(HVIW vw, I4B i0, I4B ic, rSTR st, I4B flg); // 文字列のサーチ
```

ビュー `vw` の `ic` 列を `i0` 行目から末尾に向かって走査し、文字列 `st` を検索します。

使用法は、`SearchTableCellText` 関数と同じです。

```
BOOL ReadCSV(HVIW vw, rSTR nam, rCSV I cv); // CSV読み込み
```

```
BOOL WriteCSV(HVIW vw, rSTR nam, rCSV I cv); // CSV書き出し
```

ビュー `vw` にファイル名 `nam` のテキストファイルから読み込み／テキストファイルへ書き出します。

`cs` は、テキストファイルの書式を表す `CSV I` 構造体への参照です。

`CSV I` 構造体は、以下のように宣言されています。

```
typedef struct {  
    I1B dlm; // 区切り文字  
    I1B quo; // 引用符文字(引用符なしのときは、0)  
    U2B tsw; // タイトル行(0: 列名、1: なし、2: 列タイトル)  
} CSV I;
```

```
typedef CSV I *pCSV I, *PCSV I;
```

```
typedef const CSV I *rCSV I, *PCCSV I;
```

6.3.5 サイト変数操作関数

以下の関数は、サイト変数を操作します。

```
rSTR SiteName(void); // サイト名
```

```
rSTR SiteTitle(void); // サイトタイトル
```

```
rSTR SiteDescription(void); // サイト説明
```

```
F8B TaxRate(void); // 消費税率
```

```
I4B FYearMonth(void); // 事業年度開始月
```

```
I4B NMonthFYear(void); // 事業年度月数
```

```
rSTR UserPath(void); // ユーザーフォルダ
```

```
void SetCurDate(I4B sr); // 計算日の設定
```

```
I4B CurDate(void); // 計算日
```

```
I4B BFYear(void); // 事業年度開始日
```

```
I4B EFYear(void); // 事業年度終了日
```

```
I4B BMonth(void); // 当月初日
```

```
I4B EMonth(void); // 当月末日
```

計算日は、プログラム起動時に `today` 関数で初期化されますが、`SetCurDate` 関数で再設定できます。

6.3.6 データベース管理関数

以下の関数は、サイト内のデータベースファイル等を操作します。

rSTR CurUser(void); // ユーザー名

ログインユーザー名を返します。

I4B CurAuth(void); // 権限

ログインユーザーの権限レベル（2.2節参照）を返します。

BOOL ConnectDBX(rBDBI db); // D B へ接続

D B 定義情報を与えてデータベースに接続します。

db は、D B 定義情報を記述する BDBI 構造体への参照です。

BDBI 構造体は、以下のように宣言されています。

```
typedef struct {  
    rTBLS ts;    // TBLS 構造体への参照  
    rSTR  nam;   // D B 名  
    rSTR  tit;   // D B タイトル  
    rSTR  dsc;   // D B 説明  
} BDBI;
```

typedef BDBI *pBDBI, *PBDBI;

typedef const BDBI *rBDBI, *PBDBI;

ts は、D B の表構成を定義する TBLS 構造体への参照です。

TBLS 構造体は、TBLI 構造体の配列を記述します。

```
typedef struct {  
    pTBLI ti;    // TBLI 構造体の配列の先頭アドレス  
    I4B  ni;     // 配列の要素数  
} TBLS;
```

typedef TBLS *pTBLS, *PTBLS;

typedef const TBLS *rTBLS, *PCTBLS;

D B に属する表のインデックスは、TBLI 構造体の配列順によって決定されます。またこの配列順は、表の間の参照関係を拘束します。参照先の親表は、参照元の子表よりも常に前に置かなければなりません。

nam は、D B 名にする16byte以内の英字と数字の（C 言語の識別子形式の）文字列です。

tit は、D B タイトルにする32byte以内の任意文字列です。省略すると、nam が tit として使われます。

dsc は、D B 説明にする64byte以内の任意文字列です。省略すると、tit が dsc として使われます。

この関数は、データベースに未接続の状態でのみ実行できます。同時に複数のデータベースに接続することはできません。

戻り値は、関数の成功／失敗を TRUE ／ FALSE で示します。

BOOL ConnectDB(rSTR nam); // D B へ接続

サイトに登録されているデータベース nam に接続します。

この関数は、ConnectDBX 関数を呼び出す D B 定義プロシージャを実行します。

void DisconnectDB(void); // D B への接続解除

現在接続中のデータベースへの接続を解除します。

データベースに未接続の状態でのこの関数を実行しても害はありません。

rSTR CurDBName(void); // 現 D B 名

rSTR CurDBTitle(void); // 現 D B タイトル

rSTR CurDBDescription(void); // 現 D B 説明

現在接続中のデータベースの D B 名／D B タイトル／D B 説明文字列を返します。データベースに未接続なら空文字列を返します。

I4B CurDBStatus(void); // 現 D B 状態

現在データベースに接続中なら 0 以外の値、未接続なら 0 を返します。

I4B NDBTables(void); // 現 D B に属する表の数

現在接続中のデータベースに属する表の数を返します。データベースに未接続なら 0 を返します。

HTBL DBTable(I4B id); // 表インデックスから表ハンドルを求める

現在接続中のデータベース内のインデックス **id** の表の表ハンドルを返します。
データベースに未接続の状態でこの関数を実行するとエラーになります。

rSTR DBTableName(I4B id); // 表インデックスから表名を求める

rSTR DBTableTitle(I4B id); // 表インデックスから表タイトルを求める

rSTR DBTableDescription(I4B id); // 表インデックスから表説明を求める

現在接続中のデータベース内のインデックス **id** の表の表名／表タイトル／表説明文字列を返します。
データベースに未接続の状態でこれらの関数を実行するとエラーになります。

rSTR DBTableFileName(I4B id); // 表インデックスから表ファイル名を求める

現在接続中のデータベース内のインデックス **id** の表の表ファイル名を返します。
データベースに未接続の状態でこの関数を実行するとエラーになります。

I4B DBTableIndex(rSTR nam); // 表名から表インデックスを求める

現在接続中のデータベース内の表 **nam** のインデックスを求めます。データベースに未接続または表名がデータベース内に存在しないときは -1 が返されます。

HTBL DBHTBL(rSTR nam); // 表名から表ハンドルを求める

現在接続中のデータベース内の表 **nam** の表ハンドルを求めます。データベースに未接続または表名がデータベース内に存在しないときはエラーになります。

HOPN OpenTable(rSTR tbs, I4B md); // 表を開く

HOPN OpenTables(rSTR tbs, I4B md); // 表を開く

表名リスト **tbs** を与えて表データをロードします。（２種類の関数名はシノニムです。）

表名リストとは、表名をコンマで区切って並べた文字列です。

モード **md** は、以下のフラグ値を組み合わせて指定します。（ただし、**OM_RDO** と **OM_RDW** は同時に指定できません。）

OM_RDO 読取専用モードで開く

OM_RDW 編集モードで開く

OM_LCK ファイルをロックする

OM_IND 内部使用のため予約

戻り値は、成功時は有効な表オープン情報ハンドル、失敗時は **NULL** です。

この関数は、表の間の参照関係を考慮して表データをロードします。例えば、**EQUDAT** 表が **LEADAT** 表と **MNTDAT** 表を参照している場合、

OpenTable("EQUDAT", OM_RDW);

と指定するだけで **EQUDAT** 表が編集モードでロードされるのと同時に **LEADAT** 表と **MNTDAT** 表が読取専用モードでロードされます。

編集モードで開いている表を再度編集モードで開くことはできませんが、読取専用モードでは同じ表を重複して開くことができます。

HOPN OpenTableX(rIDX pt, I4B md); // 表を開く

HOPN OpenTablesX(rIDX pt, I4B md); // 表を開く

表インデックスのリスト **pt** を与えて表データをロードします。（２種類の関数名はシノニムです。）

BOOL CloseTable(HOPN op, BOOL sv); // 表を閉じる

BOOL CloseTables(HOPN op, BOOL sv); // 表を閉じる

開いた表を閉じます。（２種類の関数名はシノニムです。）

sv は、編集モードで開いた場合、編集内容を保存する／しないを **TRUE** ／ **FALSE** で示します。編集モード以外で開いた場合は、**FALSE** を与えなければなりません。

戻り値は、関数の成功／失敗を **TRUE** ／ **FALSE** で示します。

BOOL SaveTable(HOPN op); // 表の保存

BOOL SaveTables(HOPN op); // 表の保存

編集モードで開いた表の現在の編集内容をファイルに上書保存します。（２種類の関数名はシノニムです。）

上書保存後の表データ内容には、他のユーザーの編集内容が（もしあれば）反映されます。

戻り値は、関数の成功／失敗を **TRUE** ／ **FALSE** で示します。

HTBL OHTBL(HOPN op); // 開いた表

表オープン情報のハンドルから表ハンドルを取得します。複数の表を開いている場合は、0 番目の表の表ハンドルを返します。

I4B NOTables(HOPN op); // 開いた表の数

開いた表の数を返します。

HTBL OTable(HOPN op, I4B i); // 開いた表の i 番目

開いた表の i 番目の表の表ハンドルを返します。

BOOL IsReadOnly(HOPN op); // 読取専用

読取専用モードで開いたかどうかを TRUE / FALSE で返します。

BOOL IsLocked(HOPN op); // ロック中

ペシミスティック・ロックモードで開いたかどうかを TRUE / FALSE で返します。

BOOL IsModified(HOPN op); // 表が更新されている

表を開いた（または最後に上書保存した）時点以降、クライアントのメモリ内で表データに変更が加えられたかどうかを TRUE / FALSE で返します。

BOOL NewDBFiles(void); // 空の表ファイル作成

現在接続中のデータベースに属する表の表ファイルがまだサイト内に存在しない場合、空の表ファイルを作成します。

BOOL BackupDBFiles(rSTR dir); // 現DBに属するファイルの一括コピー

現在接続中のデータベースに属する表の表ファイルを指定フォルダに一括コピーします。

BOOL TrashDBFiles(rSTR dir); // 現DBに属するファイルの一括移動

現在接続中のデータベースに属する表の表ファイル（ただし TF_IMP 属性の表を除く）と DB 定義プロシージャのバイトコードファイルを指定フォルダに一括移動します。

6.4 帳票作成関数等

6.4.1 メッセージボックス関数、ダイアログボックス関数、ステータスメッセージ関数

以下の関数は、メッセージボックスを表示します。

I4B OkBox(HWND hwnd, rSTR st, rSTR ca); // メッセージボックス

「OK」ボタンのメッセージを表示します。

hwnd はフレームウィンドウのハンドル、st はメッセージ文字列、ca はタイトル文字列です。
戻り値は、IDOK です。

I4B OkCanBox(HWND hwnd, rSTR st, rSTR ca); // メッセージボックス

「OK」「キャンセル」ボタンのメッセージを表示します。

戻り値は、IDOK / IDCANCEL のいずれかです。

I4B YesNoBox(HWND hwnd, rSTR st, rSTR ca); // メッセージボックス

「はい」「いいえ」ボタンのメッセージを表示します。

戻り値は、IDYES / IDNO のいずれかです。

I4B YesNoCanBox(HWND hwnd, rSTR st, rSTR ca); // メッセージボックス

「はい」「いいえ」「キャンセル」ボタンのメッセージを表示します。

戻り値は、IDYES / IDNO / IDCANCEL のいずれかです。

I4B ErrorBox(HWND hwnd, rSTR st, rSTR ca); // メッセージボックス

エラーアイコンと「OK」ボタンのメッセージを表示します。

戻り値は、IDOK です。

I4B WarningBox(HWND hwnd, rSTR st, rSTR ca); // メッセージボックス

警告アイコンと「はい」「いいえ」ボタンのメッセージを表示します。

戻り値は、IDYES / IDNO のいずれかです。

I4B AlertBox(HWND hwnd, rSTR st, rSTR ca); // メッセージボックス

警告アイコンと「中止」「再試行」「無視」ボタンのメッセージを表示します。

戻り値は、IDABORT / IDRETRY / IDIGNORE のいずれかです。

以下の関数は、ダイアログボックスを表示します。

BOOL EditRowDlgBox(HWND hwnd, HBUF rb, rSTR tit); // 行編集

行バッファ rb に格納された、表の 1 行分のデータをダイアログボックス上で編集します。

hwnd はフレームウィンドウのハンドル、tit はタイトル文字列です。

このダイアログボックス上では、[Enter] キーは「OK」ボタンの代わりにはならず、[Tab] キーと等しい働きをします。

戻り値は、「OK」ボタンが押されたか、「キャンセル」ボタンが押されたかを TRUE / FALSE で示します。

戻り値が FALSE のとき、rb の内容は元のままです。

以下の関数は、ステータスバーにメッセージを表示します。

void StatMsg(HWND hwnd, rSTR st); // ステータスメッセージ

ステータスバーにメッセージを表示します。

void BusyMsg(HWND hwnd, rSTR st); // ビジーメッセージ

ステータスバーにメッセージを表示し、マウスカーソルを待機状態にします。

void EndBusy(HWND hwnd); // ビジーメッセージ終了

ステータスバーのメッセージを消去し、マウスカーソルを元に戻します。

6.4.2 画像処理関数

以下の関数は、画像データを操作します。画像データオブジェクトは、デバイス独立ビットマップハンドル（HDIB 型）によって参照されます。

HDIB CreateDIB(I4B w, I4B h, I4B flg); // デバイス独立ビットマップの作成

横 **w** ピクセル×縦 **h** ピクセルのデバイス独立ビットマップを作成します。

flg は、以下のフラグ値を組み合わせてピクセル当たりのビット数とピクセルの格納方向を示します。

DIB_24B 24ビット（省略時既定値）

DIB_32B 32ビット

DIB_BUP 下から上（省略時既定値）

DIB_TDN 上から下

戻り値は、成功時は有効なデバイス独立ビットマップハンドル、失敗時は **NULL** です。

flg を **DIB_32B|DIB_TDN** にして画像データオブジェクトを作成すると、次のようにピクセルデータを整数の配列とみなしてアクセスすることが可能になります。

```
void Sample(HDIB bm)
```

```
{ // サンプル
```

```
int i, j, w, h;
```

```
RGBC *qd;
```

```
w = DIB_WID(bm); h = abs(DIB_HEI(bm));
```

```
qd = (RGBC *)DIB_IMG(bm);
```

```
for (i = 0; i < h; i++) {
```

```
    for (j = 0; j < w; j++) {
```

```
        qd[i * w + j] = 0xffffffff;
```

```
    }
```

```
}
```

```
}
```

I4B DIB_WID(HDIB bm); // 横ピクセル数

I4B DIB_HEI(HDIB bm); // 縦ピクセル数(上から下のときは負数)

I4B DIB_BTS(HDIB bm); // ピクセル当たりのビット数

pBIN DIB_IMG(HDIB bm); // ピクセルデータのアドレス

HDIB CreateCompatibleDIB(HDIB bm); // 互換性のあるデバイス独立ビットマップの作成

ビットマップ **bm** と同型（**w**, **h**, **flg** が等しい）のビットマップを作成します。

HDIB DuplicateDIB(HDIB bm); // デバイス独立ビットマップの複製

ビットマップ **bm** を複製（同型のビットマップを作成し、ピクセルデータをコピー）します。

void DeleteDIB(HDIB bm); // デバイス独立ビットマップの削除

ビットマップ **bm** を削除します。

HDIB MoveDIB(HDIB bm, HDIB bx); // デバイス独立ビットマップの移動

ビットマップ **bm** にビットマップ **bx** の内容を上書きしてから **bx** を削除します。 **bm** と **bx** は同型でなくても構いません。戻り値は **bm** です。

pSIZE GetDIBSize(pSIZE sz, HDIB bm); // 画像サイズの取得

SIZE 構造体に横と縦のピクセル数を取得します。

HDIB LoadDIB(HDIB bm, rSTR nam); // BMPファイル読み込み

ビットマップ **bm** にファイル名 **nam** の BMP ファイルから画像データを読み込みます。 **bm** は **NULL** にしておくこともでき、その場合は新しいビットマップが作成されます。

戻り値は、**bm** を与えた場合は **bm**、**NULL** にした場合は新しく作成されたビットマップのハンドルです。

BOOL SaveDIB(HDIB bm, rSTR nam); // BMPファイル書き出し

ビットマップ **bm** からファイル名 **nam** の BMP ファイルへ画像データを書き出します。

HDIB LoadBMP(HDIB bm, rSTR nam); // BMPファイル読込

BMPファイルから画像データを読み込みます。

LoadDIB 関数との違いは、ビットマップ形式をメモリ内での操作に適した **DIB_32B|DIB_TDN** 形式に変換して読み込むことです。

BOOL SaveBMP(HDIB bm, rSTR nam); // BMPファイル書出

BMPファイルへ画像データを書き出します。

SaveDIB 関数との違いは、ビットマップ形式をファイル保存に適した **DIB_24B|DIB_BUP** 形式に変換して書き出すことです。

HDIB LoadJPG(HDIB bm, rSTR nam); // 画像ファイル読込

JPEGファイル等から画像データを読み込みます。画像形式はファイル名の拡張子によって判断されます。

サポートする拡張子は、.JPG, .JPEG, .JPE, .JFIF, .TIF, .TIFF, .GIF, .PNG, .BMP, .DIB です。

読み込んだデータは、**DIB_32B|DIB_TDN** 形式のビットマップになります。

BOOL SaveJPG(HDIB bm, rSTR nam); // 画像ファイル書出

JPEGファイル等へ画像データを書き出します。画像形式はファイル名の拡張子によって判断されます。

BOOL TrimDIB(HDIB bm, I4B x, I4B y, I4B w, I4B h, RGBC cb); // トリミング

ビットマップbmをトリミングします。x, y, w, hは、トリミング範囲を示します。範囲指定は、元の画像より大きくても構いません。cbは、元の画像をはみだす範囲の色指定です。

BOOL PasteDIB(HDIB bm, HDIB bx, I4B x, I4B y); // 貼り付け

ビットマップbxをビットマップbmに貼り付けます。

BOOL ResizeDIBX(HDIB bm, I4B w, I4B h, I4B sw); // 拡大縮小

ビットマップbmを拡大縮小します。swは、拡大縮小のアルゴリズムを以下の定数で示します。

DIB_MEA 面積平均法

DIB_LIN 線形補完法

DIB_SMO スムースネス優先

DIB_SHA シャープネス優先

BOOL ResizeDIB(HDIB bm, I4B w, I4B h); // 拡大縮小

ビットマップbmを拡大縮小します。ResizeDIBX(bm, w, h, DIB_MEA)と同じです。

BOOL InvertDIB(HDIB bm, I4B sw); // 鏡映反転

ビットマップbmを鏡映反転します。swは、反転の方向を以下の定数で示します。

DIB_HRZ 水平方向

DIB_VRT 垂直方向

BOOL RotateDIB(HDIB bm, F8B dg, RGBC cb); // 任意角度回転

ビットマップbmを任意角度回転します。dgは、回転角度（単位は度、反時計回り正）です。

BOOL TransformDIB(HDIB bm, rXFMQ q, RGBC cb); // 自由変形

ビットマップbmを任意の四辺形に変形します。qは、四隅ピクセルの移動先を示すXFMQ構造体への参照です。

typedef struct {

I4B x1, y1; // 左上隅(0, 0)の移動先

I4B x2, y2; // 右上隅(w-1, 0)の移動先

I4B x3, y3; // 左下隅(0, h-1)の移動先

I4B x4, y4; // 右下隅(w-1, h-1)の移動先

} XFMQ;

typedef XFMQ *pXFMQ, *PXFMQ;

typedef const XFMQ *rXFMQ, *PCXFMQ;

void SetXFMQ(pXFMQ q, I4B x1, I4B y1, I4B x2, I4B y2, I4B x3, I4B y3, I4B x4, I4B y4);

XFMQ構造体に値を設定します。

BOOL ConvertDIB(HDIB bm, I4B flg); // デバイス独立ビットマップのフォーマット変換

ビットマップbmの形式を変換します。

6.4.3 帳票作成関数

以下の関数は、帳票作成をサポートします。ほとんどの帳票作成関数は、レポート情報（帳票作成の環境を管理するオブジェクト）のハンドル（HREP 型）を引数に取ります。

HWND hWND(HREP rp); // 親ウィンドウのハンドル

ReportMain 関数を呼び出した親ウィンドウのハンドルを返します。

void SetReportInfo(HREP rp, rSTR tit, rSTR usr, I4B sr, I4B tm); // レポート情報の設定

帳票のタイトル **tit**、作成者名 **usr**、作成日 **sr**、作成時刻 **tm** を設定します。

既定値は、帳票作成プロシージャのタイトル、ログインユーザー名、描画開始日時です。

rSTR ReportTitle(HREP rp); // タイトル

帳票のタイトル文字列を返します。

rSTR ReportAuthor(HREP rp); // 作成者

帳票の作成者名文字列を返します。

I4B ReportDate(HREP rp); // 作成日

帳票の作成日（日付シリアル値）を返します。

I4B ReportTime(HREP rp); // 作成時刻

帳票の作成時刻（時刻シリアル値）を返します。

void SetReportExtra(HREP rp, uANY pa); // レポートの付加情報設定

レポート情報に任意の付加情報を設定します。

uANY ReportExtra(HREP rp); // レポートの付加情報

レポートの付加情報を返します。

void SetNPages(HREP rp, I4B npg); // ページ数設定

I4B NPages(HREP rp); // ページ数

帳票のページ数を設定／取得します。

void SetPaperSize(HREP rp, F8B w, F8B h, F8B scl); // 出力用紙サイズと倍率の設定

F8B PaperWidth(HREP rp); // 出力用紙横サイズ

F8B PaperHeight(HREP rp); // 出力用紙縦サイズ

出力用紙サイズを設定／取得します。既定の用紙サイズは、A 4 縦（210×297）です。

出力用紙サイズは、ユーザーが印刷用紙設定ダイアログボックスで変更することができます。

void SetPageSize(HREP rp, F8B w, F8B h); // 原稿用紙サイズ設定

F8B PageWidth(HREP rp); // 原稿用紙横サイズ

F8B PageHeight(HREP rp); // 原稿用紙縦サイズ

原稿用紙サイズを設定／取得します。既定の用紙サイズは、A 4 縦（210×297）です。

SetPageSize(rp, w, h) は、SetPaperSize(rp, w, h, 1.0) と同じです。

F8B PageLeft(HREP rp); // 原稿用紙左端

F8B PageTop(HREP rp); // 原稿用紙上端

原稿用紙左端／上端の x / y 座標を返します。既定値は、0 / 0 です。SetOrigin 関数で変更できます。

F8B PageRight(HREP rp); // 原稿用紙右端

F8B PageBottom(HREP rp); // 原稿用紙下端

原稿用紙右端／下端の x / y 座標を返します。既定値は、原稿用紙サイズの w / h です。

void SetPageMargin(HREP rp, F8B lm, F8B tm, F8B rm, F8B bm); // 用紙の余白サイズ設定

用紙の上下左右マージン（左余白 **lm**、上余白 **tm**、右余白 **rm**、下余白 **bm**）を設定します。

上下左右マージンは、ユーザーが印刷用紙設定ダイアログボックスで変更することができます。

用紙のマージン設定は、以下に列挙するページ設定情報取得関数に影響しますが、描画関数の描画領域を制限するものではありません。ページ設定情報を参照して描画するかどうかは自由です。

```
F8B BodyWidth(HREP rp); // 本文領域横サイズ
F8B BodyHeight(HREP rp); // 本文領域縦サイズ
F8B BodyLeft(HREP rp); // 本文領域左端
F8B BodyTop(HREP rp); // 本文領域上端
F8B BodyRight(HREP rp); // 本文領域右端
F8B BodyBottom(HREP rp); // 本文領域下端
```

```
void SetHeaderHeight(HREP rp, F8B hh); // ヘッダ領域高さ設定
上マージン内のヘッダ領域の縦サイズを設定します。既定値は、tm-5 です。
```

```
void SetFooterHeight(HREP rp, F8B fh); // フッタ領域高さ設定
下マージン内のフッタ領域の縦サイズを設定します。既定値は、bm-5 です。
```

```
F8B HeaderWidth(HREP rp); // ヘッダ領域横サイズ、BodyWidthに同じ
F8B HeaderHeight(HREP rp); // ヘッダ領域高さ
F8B HeaderLeft(HREP rp); // ヘッダ領域左端、BodyLeftに同じ
F8B HeaderTop(HREP rp); // ヘッダ領域上端
F8B HeaderRight(HREP rp); // ヘッダ領域右端、BodyRightに同じ
F8B HeaderBottom(HREP rp); // ヘッダ領域下端、BodyTopに同じ
F8B FooterWidth(HREP rp); // フッタ領域横サイズ、BodyWidthに同じ
F8B FooterHeight(HREP rp); // フッタ領域高さ
F8B FooterLeft(HREP rp); // フッタ領域左端、BodyLeftに同じ
F8B FooterTop(HREP rp); // フッタ領域上端、BodyBottomに同じ
F8B FooterRight(HREP rp); // フッタ領域右端、BodyRightに同じ
F8B FooterBottom(HREP rp); // フッタ領域下端
```

```
void SetReqBodyWidth(HREP rp, F8B w, I4B sw); // 横幅合わせ基準値設定
横幅合わせ基準値（本文領域の横サイズとして要求する幅）wを与えて、印刷用紙設定ダイアログボックスの「横幅に合わせる」ボタンを使用可能にします。swをRSW_AUTにすると、デフォルトで「横幅に合わせる」ボタンが選択状態になります。swをRSW_SKPにすると、印刷用紙設定ダイアログボックス入力がスキップされます。
```

```
F8B ReqBodyWidth(HREP rp); // 横幅合わせ基準値
横幅合わせ基準値を返します。
```

```
pCELL PageCell(pCELL pc, HREP rp); // 原稿用紙矩形
pCELL BodyCell(pCELL pc, HREP rp); // 本文領域矩形
pCELL HeaderCell(pCELL pc, HREP rp); // ヘッダ領域矩形
pCELL FooterCell(pCELL pc, HREP rp); // フッタ領域矩形
原稿用紙領域／本文領域／ヘッダ領域／フッタ領域を CELL 構造体 to 取得します。
CELL 構造体は、以下のように宣言されています。
```

```
typedef struct {
    F8B x1; // 左端x座標
    F8B y1; // 上端y座標
    F8B x2; // 右端x座標
    F8B y2; // 下端y座標
} CELL;
typedef CELL *pCELL, *PCCELL;
typedef const CELL *rCELL, *PCCELL;
```

```
void SetOrigin(HREP rp, F8B xorg, F8B yorg); // 原点の設定
F8B OriginX(HREP rp); // 原点x座標
F8B OriginY(HREP rp); // 原点y座標
原稿用紙上の原点座標（左上隅からの x y 方向移動量）を設定／取得します。
```

```
void SetScale(HREP rp, F8B xscl, F8B yscl); // 倍率の設定
F8B ScaleX(HREP rp); // x方向倍率
F8B ScaleY(HREP rp); // y方向倍率
座標値の拡大縮小倍率を設定／取得します。
```

pCELL SetCell(pCELL pc, F8B x1, F8B y1, F8B x2, F8B y2); // セル矩形の設定
セル矩形 pc の各座標を設定します。

pCELL MoveCell(pCELL pc, F8B dx, F8B dy); // セル矩形の移動
移動量を指定してセル矩形 pc を移動します。

pCELL ResizeCell(pCELL pc, F8B dx, F8B dy); // セル矩形のサイズ変更
右下隅の移動量を指定してセル矩形 pc のサイズを変更します。

pCELL ReshapeCell(pCELL pc, F8B lm, F8B tm, F8B rm, F8B bm); // セル矩形の変形
各座標の移動量を指定してセル矩形 pc を変形します。移動量は、内側に向かう移動を正とします。

void SetCellMargin(HREP rp, F8B hcmg, F8B vcmg); // セル内パディングの設定
F8B CellMarginH(HREP rp); // セル内左右パディング
F8B CellMarginV(HREP rp); // セル内上下パディング
セル内の左右パディング、上下パディングを設定／取得します。

I4B SetLineStyle(HREP rp, I4B lst); // 線種設定
線種を設定します。lst は、線種を表す以下の定数名のいずれかとします。

LST_NUL 線なし
LST_THN 細線
LST_REG 標準
LST_MED 中太
LST_THK 太線
LST_XTK 極太
LST_DSH 破線
LST_DOT 点線
LST_DSD 一点鎖線
LST_DDD 二点鎖線

戻り値は、関数を呼び出す前に設定されていた線種です。

BGRG SetLineColor(HREP rp, BGRG col); // 線色設定
線の色を COLORREF 値 (0xbbggrr) で設定します。
戻り値は、関数を呼び出す前に設定されていた色です。

void SetTextFont(HREP rp, rSTR face, F8B ch, I4B flg, BGRG col); // テキストフォント、文字色設定
フォント名 face、文字高さ ch、文字属性 flg、文字色 col を設定します。
flg は、以下のフラグ値の組み合わせで示します。

TFF_BLD ボールド体
TFF_ITA イタリック体
TFF_UND アンダーライン
TFF_STO 取り消し線
TFF_VRT 縦書き文字

void SetTextStyle(HREP rp, F8B cw, F8B ch, F8B cd, rSTR face); // テキスト文字サイズ、フォント設定
平均文字幅 cw、文字高さ ch、文字間スペース幅 cd、フォント名 face を設定します。cw は、0 にしておくとも高さに応じた適正幅になります。cd も通常は 0 にするのが適正です。face を NULL にすると、現在設定されているフォント名が引き続き使われます。

pWHSZ TextSize(pWHSZ sz, HREP rp, rSTR st, I4B cb); // 文字列のサイズ
文字列 st のサイズを WHSZ 構造体を取得します。
WSHZ 構造体は、以下のように宣言されています。
typedef struct {
 F8B w; // 幅
 F8B h; // 高さ
} WHSZ;
typedef WHSZ *pWHSZ, *PWHSZ;
cb を指定すると、st から cb バイト分の部分文字列のサイズが得られます。

pWHSZ AveCharSize(pWHSZ sz, HREP rp); // 平均文字サイズ
半角英字の平均文字サイズを取得します。

pWHSZ ZenCharSize(pWHSZ sz, HREP rp); // 全角文字サイズ
全角文字のサイズを取得します。

void DrawLine(HREP rp, F8B x1, F8B y1, F8B x2, F8B y2); // 線分の描画
点 x1, y1 から点 x2, y2 まで線分を描画します。

void DrawCirc(HREP rp, F8B xc, F8B yc, F8B rc, F8B t1, F8B t2); // 円(円弧)の描画
中心 xc, yc、半径 rc、始点 t1 度、終点 t2 度（反時計回り）の円弧を描画します。始点と終点を同じにすると円になります。

I4B RST(I4B l1st, I4B t1st, I4B r1st, I4B b1st); // 罫線スタイル
セル矩形の四辺の線種を指定する整数値を与えます。l1st は左辺、t1st は上辺、r1st は右辺、b1st は下辺の線種です。

void DrawCell(HREP rp, rCELL pc, I4B rst); // 矩形の描画
セル矩形の四辺を描画します。rst は、四辺の線種を指定する整数値で、RST 関数または以下の定数名で与えます。

RST_NUL 線なし
RST_DEF 現在の設定線種
RST_THN 細線
RST_REG 標準
RST_MED 中太
RST_THK 太線
RST_XTK 極太
RST_DSH 破線
RST_DOT 点線
RST_DSD 一点鎖線
RST_DDD 二点鎖線

線色は、現在の設定が用いられます。

void FillCell(HREP rp, rCELL pc, I4B rst, BGRC col); // 矩形の塗りつぶし
セル矩形の四辺を描画し、内部を col 色で塗りつぶします。
線色は、現在の設定が用いられます。

void FillPoly(HREP rp, rXYPT pt, I4B np, BGRC col); // 多角形の塗りつぶし
多角形を描画し、内部を col 色で塗りつぶします。pt は多角形の頂点座標を与える XYPT 構造体の配列の先頭アドレス、np は配列の要素数です。
XYPT 構造体は、以下のように宣言されています。

```
typedef struct {  
    F8B x;    // x座標  
    F8B y;    // y座標  
} XYPT;  
typedef XYPT *pXYPT, *PXYPT;  
typedef const XYPT *rXYPT, *PCXYPT;  
線種と線色は、現在の設定が用いられます。
```

void FillCirc(HREP rp, F8B xc, F8B yc, F8B rc, F8B t1, F8B t2, BGRC col); // 円(扇形)の塗りつぶし
円(扇形)を描画し、内部を col 色で塗りつぶします。
線種と線色は、現在の設定が用いられます。

void CellText(HREP rp, pCELL pc, rSTR st, I4B fmt); // 文字列の描画
セル矩形内に文字列を描画します。fmt は、矩形内での文字列の配置を以下のフラグ値を組み合わせて示します。
DT_FRC 上寄せ
DT_LFT 左寄せ（省略時既定値）
DT_CEN 水平方向中央揃え
DT_RGT 右寄せ
DT_MID 垂直方向中央揃え（省略時既定値）

DT_BTМ 下寄せ
DT_CAL テキストサイズの計算
DT_PEL 必要ならテキストの途中を省略
DT_EEL 必要ならテキストの最後を省略
DT_MUL 自動的に折り返す
DT_JST 均等割付
DT_AUT 自動的に幅を縮小する
DT_PRE プリフィクス文字を処理する
DT_CLP クリッピングする

void VCellText(HREP rp, pCELL pc, rSTR st, I4B fmt); // 縦書き文字列の描画

セル矩形内に縦書き文字列を描画します。fmt は、矩形内での文字列の配置を以下のフラグ値で示します。

VT_LFT 左寄せ
VT_CEN 水平方向中央揃え（省略時既定値）
VT_RGT 右寄せ
VT_TOP 上寄せ（省略時既定値）
VT_MID 垂直方向中央揃え
VT_BTМ 下寄せ
VT_CAL テキストサイズの計算

この関数は、SetTextFont 関数で次のように縦書き用のフォントを選択し、全角文字列を与えた場合のみ有効です。

SetTextFont(rp, "MS ゴシック", 4.0, TFF_VRT, 0);

折り返しや均等割付の指定はできません。文字列が長いときは、矩形内に収まるように自動的に縮小されます。

void SetGrid0(pGRID gr, F8B x0, F8B y0, I4B nc, I4B nr, F8B dcw, F8B drh);

void SetGrid1(pGRID gr, F8B x0, F8B y0, I4B nc, I4B nr, F8B *cw, F8B drh);

void SetGrid2(pGRID gr, F8B x0, F8B y0, I4B nc, I4B nr, F8B dcw, F8B *rh);

void SetGrid3(pGRID gr, F8B x0, F8B y0, I4B nc, I4B nr, F8B *cw, F8B *rh);

用紙上にグリッドを設定します。

GRID 構造体は、以下のように宣言されています。

```
typedef struct {
    F8B    x0;    // 左端x座標
    F8B    y0;    // 上端y座標
    F8B    dcw;   // 列幅(横等間隔グリッドの場合)
    F8B    drh;   // 行ピッチ(縦等間隔グリッドの場合)
    F8B    *cw;   // 列幅配列の先頭アドレス(横任意間隔グリッドの場合)
    I4B    nc;    // 列数
    F8B    *rh;   // 行ピッチ配列の先頭アドレス(縦任意間隔グリッドの場合)
    I4B    nr;    // 行数
} GRID;
typedef GRID *pGRID, *PGRID;
typedef const GRID *rGRID, *PCGRID;
```

pCELL GridCell(pCELL pc, rGRID gr, I4B ir, I4B ic); // グリッドのセル矩形

グリッド gr の ir 行 ic 列のセル矩形を pc に取得します。

void DrawGridCell(HREP rp, rGRID gr, I4B ir, I4B ic, I4B rst); // グリッドのセル矩形の描画

グリッド gr の ir 行 ic 列のセル矩形の四辺を描画します。rst には、DrawCell 関数の線種指定が使える他、以下の定数名が使用可能です。

RST_I_R 外枠が標準、内側が細線
RST_I_M 外枠が中太、内側が細線
RST_I_B 外枠が太線、内側が細線
RST_R_M 外枠が中太、内側が標準
RST_R_B 外枠が太線、内側が標準
RST_R_X 外枠が極太、内側が標準

void GridCellText(HREP rp, rGRID gr, I4B ir, I4B ic, I4B rst, rSTR st, I4B fmt); // セルの描画

グリッドのセル矩形の四辺を描画し、矩形内に文字列を描画します。

DrawGridCell(rp, gr, ir, ic); CellText(rp, GridCell(&cel, gr, ir, ic), st, fmt) と同じです。

```
void CellStamp(HREP rp, rCELL pc, rSTMP ps); // 印影の描画
```

セル矩形内に印影を描画します。ps は、印影の形式を与える STMP 構造体への参照です。

印影の色とフォント名は、SetTextFont 関数で指定します。文字のサイズは、印影の形式と直径に応じて自動的に調整されます。

```
void SetStamp0(pSTMP ps, F8B dc, rSTR st);
```

```
void SetStamp1(pSTMP ps, F8B dc, rSTR st);
```

```
void SetStamp2(pSTMP ps, F8B dc, rSTR st1, rSTR st2);
```

```
void SetStamp3(pSTMP ps, F8B dc, rSTR st1, rSTR st2, rSTR st3);
```

それぞれの形式の印影情報を設定します。

STMP 構造体は、以下のように宣言されています。

```
typedef struct {  
    I4B    sty; // 形式 (0:縦書き、1:横書き、2:二段、3:三段)  
    rSTR    st1; // 文字列  
    rSTR    st2; // 二日目文字列  
    rSTR    st3; // 三日目文字列  
    F8B    dc;  // 直径  
} STMP;  
typedef STMP *pSTMP, *PSTMP;  
typedef const STMP *rSTMP, *PCSTMP;
```

```
void CellImage(HREP rp, pCELL pc, HDIB bm, I4B fmt); // 画像の貼り付け
```

セル矩形内に画像を貼り付けます。fmt は、矩形内での画像の配置を以下のフラグ値で示します。

```
DT_FRC  上寄せ  
DT_LFT  左寄せ (省略時既定値)  
DT_CEN  水平方向中央揃え  
DT_RGT  右寄せ  
DT_MID  垂直方向中央揃え (省略時既定値)  
DT_BTM  下寄せ  
DT_CAL  画像サイズの計算  
DT_AJW  縦横比を維持してセル矩形の幅に合わせる  
DT_AJH  縦横比を維持してセル矩形の高さに合わせる  
DT_GRY  グレー表示する  
DT_STB  セル矩形の幅と高さに合わせる (縦横比は維持しない)  
DT_CLP  クリッピングする
```

7 ネイティブアプリケーションの作成

スクリプト言語によるストアドプロージャプログラミングでは、表ウィンドウ等のユーザーインタフェース部品のカスタマイズはできません。Bolyaiのライブラリを使い、自由なユーザーインタフェースのアプリケーションを作成するには、ネイティブのWindowsアプリケーションを開発できる言語製品（Cコンパイラ）を別途用意してBolyaiライブラリの構成ファイルを再コンパイルする必要があります。

ここでは、Bolyaiライブラリのモジュール構成および各モジュールの提供するネイティブアプリケーション用API群について簡単に解説します。

7.1 Bolyai-SDK

配布メディアのSDKフォルダには、以下のファイルが含まれています。

BOLALG.C	BOLALG.LIBのソースコードファイル
BOLALG.H	BOLALG.LIBのヘッダファイル
BOLBAS.C	BOLBAS.LIBのソースコードファイル
BOLBAS.H	BOLBAS.LIBのヘッダファイル
BOLCOM.C	BOLCOM.LIBのソースコードファイル
BOLCOM.H	BOLCOM.LIBのヘッダファイル
BOLDBE.C	BOLDBE.LIBのソースコードファイル
BOLDBE.H	BOLDBE.LIBのヘッダファイル
BOLEXT.C	BOLEXT.LIBのソースコードファイル
BOLEXT.H	BOLEXT.LIBのヘッダファイル
BOLFRO.C	BOLFRO.LIBのソースコードファイル
BOLFRO.H	BOLFRO.LIBのヘッダファイル
BOLGUI.C	BOLGUI.LIBのソースコードファイル
BOLGUI.H	BOLGUI.LIBのヘッダファイル
BOLPLS.CPP	BOLPLS.LIBのソースコードファイル
BOLPLS.H	BOLPLS.LIBのヘッダファイル
BOLYAI.BMP	BOLYAI.EXEのリソース（ビットマップファイル）
BOLYAI.C	BOLYAI.EXEのソースコードファイル
BOLYAI.H	BOLYAI.EXEのヘッダファイル
BOLYAI.ICO	BOLYAI.EXEのリソース（アイコンファイル）
BOLYAI.RC	BOLYAI.EXEのリソーススクリプトファイル

BOLYAI.EXEを修正して再コンパイルするのに必要なファイルは、上記が全てです。

プログラムは、以下のモジュールで構成されます。

BOLALG.LIB	ユーティリティ
BOLBAS.LIB	バイトコードインタプリタ
BOLCOM.LIB	コンパイラ
BOLDBE.LIB	データベースエンジン
BOLEXT.LIB	数値計算、画像処理
BOLFRO.LIB	ユーザーインタフェース部品（汎用）
BOLGUI.LIB	ユーザーインタフェース部品（専用）
BOLPLS.LIB	GDI+ラッパー
BOLYAI.EXE	アプリケーション本体

モジュール間の依存関係は、以下の通りです。

BOLALG	→
BOLBAS	→ BOLALG
BOLCOM	→ BOLALG, BOLBAS
BOLDBE	→ BOLALG, BOLBAS
BOLEXT	→ BOLALG, BOLBAS
BOLFRO	→ BOLALG, BOLBAS, BOLEXT
BOLGUI	→ BOLALG, BOLBAS, BOLDBE, BOLEXT, BOLFRO
BOLPLS	→ BOLALG, BOLBAS, BOLEXT, BOLFRO
BOLYAI	→ BOLALG, BOLBAS, BOLCOM, BOLDBE, BOLEXT, BOLFRO, BOLGUI, BOLPLS

リンク時に必要な追加の依存ファイルは、imm32.libとGdiPlus.libです。

7.2 BOLALG. LIB

BOLALG. LIB は、ユーティリティ関数群を提供します。

このモジュールを使用するアプリケーション本体は、開始時と終了時に以下の関数を呼ばなければなりません。

`BOOL InitBOLALG(uANY pa); // 開始処理`

BOLALG. LIB モジュールを初期化します。pa は、ダミーの引数です。0 を与えてください。

`void TermBOLALG(void); // 終了処理`

BOLALG. LIB モジュールの終了処理を行います。

7.2.1 メモリブロック操作関数

以下の関数は、メモリブロックオブジェクトを操作します。メモリブロックオブジェクトは、汎用の配列オブジェクトで、メモリブロックハンドル（HBLK 型）によって参照されます。

`HBLK CreateArray(I4B cap, I4B rsz); // 動的配列の作成`

`#define CreateDArray(cap, rsz) CreateArray(cap, rsz)`

最大要素数の初期値 cap と要素サイズ rsz を与えて動的な（データ領域の再割り当てが可能な）メモリブロックを作成します。戻り値は、成功時は有効なメモリブロックハンドル、失敗時は NULL です。

メモリブロックハンドルは、以下のように定義された HBLK_ 構造体へのポインタです。

```
typedef struct {
    pANY ptr; // 配列の先頭アドレス
    I4B len; // 配列の要素数
    I4B cap; // 配列の最大要素数
    I4B rsz; // 配列の要素サイズ
} HBLK_;
typedef HBLK_ *HBLK;
```

`void DeleteArray(HBLK mb); // メモリブロックの削除`

`#define DeleteDArray(mb) DeleteArray(mb)`

メモリブロックを削除します。

`HBLK CreateMemBlk(I4B cap, I4B rsz); // 静的配列の作成`

最大要素数 cap と要素サイズ rsz を与えて静的なメモリブロックを作成します。戻り値は、成功時は有効なメモリブロックハンドル、失敗時は NULL です。

`#define CreateBinBlk(cap) CreateMemBlk(cap, 1)`

バイト列格納域を作成します。

`#define CreateStrBlk(cap) CreateBinBlk(cap)`

文字列格納域を作成します。

`#define DeleteMemBlk(mb) DeleteArray(mb)`

`#define DeleteBinBlk(bb) DeleteMemBlk(bb)`

`#define DeleteStrBlk(sb) DeleteBinBlk(sb)`

メモリブロックを削除します。

`pANY ItemPtr(HBLK mb, I4B ir); // 要素のアドレス`

メモリブロック mb の ir 番目の要素のアドレスを返します。

`BOOL InsertItem(HBLK mb, I4B ir, rANY pi); // 要素の挿入`

メモリブロック mb の ir 番目に要素を挿入します。

`BOOL DeleteItem(HBLK mb, I4B ir); // 要素の削除`

メモリブロック mb の ir 番目の要素を削除します。

`BOOL AddItem(HBLK mb, rANY pi); // 要素の追加`

メモリブロック mb の末尾に要素を追加します。

void MoveItemToFirst(HBLK mb, I4B ir); // 要素を先頭に移動
メモリブロック **mb** の **ir** 番目の要素を先頭に移動します。

void MoveItemToLast(HBLK mb, I4B ir); // 要素を末尾に移動
メモリブロック **mb** の **ir** 番目の要素を末尾に移動します。

HBLK SelectBinBlk(HBLK bb); // バイト列格納域の選択
メモリブロック **bb** を既定のバイト列格納域として選択します。戻り値は、関数を呼び出す前に選択されていたバイト列格納域のハンドル（選択されていなければ **NULL**）です。

rANY stobin(rANY pv, I4B sz); // バイト列の格納
アドレス **pv** にあるサイズ **sz** のオブジェクトを既定のバイト列格納域にコピーし、その格納先アドレスを返します。

HBLK SelectStrBlk(HBLK sb); // 文字列格納域の選択
メモリブロック **sb** を既定の文字列格納域として選択します。戻り値は、関数を呼び出す前に選択されていた文字列格納域のハンドル（選択されていなければ **NULL**）です。

rSTR stostr(rSTR st); // 文字列の格納
文字列 **st** を既定の文字列格納域にコピーし、その格納先アドレスを返します。同じ文字列を重複して格納することではなく、同じ文字列に対しては常に同じアドレスを返します。

rSTR dirnam(rSTR nam); // ディレクトリ名の抽出
パス名 **nam** からディレクトリ名を取り出します。変換結果の文字列は既定の文字列格納域に格納されます。

rSTR basnam(rSTR nam); // ディレクトリ名を外す
パス名 **nam** からファイル名を取り出します。変換結果の文字列は既定の文字列格納域に格納されます。

rSTR basnamx(rSTR nam); // ディレクトリ名と拡張子を外す
パス名 **nam** から拡張子抜きファイル名を取り出します。変換結果の文字列は既定の文字列格納域に格納されます。

rSTR adddir(rSTR dir, rSTR nam); // ディレクトリ名の追加
ファイル名 **nam** の前にディレクトリ名 **dir** を追加します。変換結果の文字列は既定の文字列格納域に格納されます。

rSTR addext(rSTR nam, rSTR ext); // 拡張子の追加
拡張子抜きファイル名 **nam** に拡張子 **ext** を追加します。変換結果の文字列は既定の文字列格納域に格納されます。

HBLK CreateTXT(I4B cap); // テキストブロックの作成
テキストブロック（文字列を保持する動的配列）を作成します。文字列 **(pSTR)xb->ptr** は、ヌル文字で終了しており、**xb->len** は **strlen((pSTR)xb->ptr) + 1** と一致します。以下の関数は、この条件を維持してテキストを操作します。

#define DeleteTXT(xb) DeleteArray(xb)
#define UnloadTXT(xb) DeleteTXT(xb)
テキストブロックを削除します。

HBLK LoadTXT(HBLK xb, rSTR nam); // テキストファイル読込
テキストブロック **xb** にファイル名 **nam** のテキストファイルからテキストを読み込みます。**xb** は **NULL** にしておくこともでき、その場合は新しいテキストブロックが作成されます。

BOOL SaveTXT(HBLK xb, rSTR nam); // テキストファイル書出
テキストブロック **xb** からファイル名 **nam** のテキストファイルにテキストを書き出します。

U4B RowExt(HBLK xb); // 行数と最大桁数
テキストブロックの行数と最大桁数を返します。戻り値の下位ワードが行数、上位ワードが最大桁数です。

U4B PtrRow(HBLK xb, rSTR pc); // 位置情報
テキストブロック中のポインタ **pc** の位置が何行目の何桁目かを返します。戻り値の下位ワードが行インデックス、上位ワードが桁インデックスです。

pSTR RowPtr (HBLK xb, U4B rc); // 行のアドレス

テキストブロック中の指定行の指定桁の文字のアドレスを返します。rc の下位ワードが行インデックス、上位ワードが桁インデックスです。rc に行インデックスを与えると、行のアドレスを返します。

I4B RowLen (HBLK xb, U4B rc); // 行の桁数

テキストブロック中の指定行の指定桁の文字から行末までの桁数を返します。rc の下位ワードが行インデックス、上位ワードが桁インデックスです。rc に行インデックスを与えると、行の桁数を返します。

U4B CharPos (HBLK xb, U4B rc); // 位置が全角文字の第2バイト目なら第1バイト目に補正

rc が示す位置が全角文字の第2バイト目なら第1バイト目に補正します。

U4B PrevPos (HBLK xb, U4B rc); // 前の文字位置

rc が示す文字位置の直前の文字位置を返します。

U4B NextPos (HBLK xb, U4B rc); // 次の文字位置

rc が示す文字位置の直後の文字位置を返します。

void CopyStr (pSTR st, HBLK xb, U4B r1, U4B r2); // 文字列のコピー

テキストブロック中の文字位置 r1 から r2 の直前までの部分を切り出して格納先 st にコピーします。

void InsertStr (HBLK xb, U4B rc, rSTR st); // 文字列の挿入

テキストブロック中の文字位置 rc に文字列 st を挿入します。

void DeleteStr (HBLK xb, U4B r1, U4B r2); // 文字列の削除

テキストブロック中の文字位置 r1 から r2 の直前までの部分を削除します。

void AddStr (HBLK xb, rSTR st); // 文字列の追加

テキストブロックの末尾に文字列 st を追加します。

U4B SearchStr (HBLK xb, U4B r0, rSTR st, I4B flg); // 文字列の検索

テキストブロック中の文字位置 r1 から末尾に向かって文字列 st を検索します。比較条件 flg は、strfindx 関数と同じです。戻り値は、文字列が見つければその文字位置、見つからなければ (U4B)-1 です。

void ReplaceStr (HBLK xb, rSTR s1, rSTR s2); // 文字列の置換

テキストブロック中の文字列 s1 を全て s2 に置換します。

7.2.2 字句解析関数

以下の関数は、与えられた字句規則に基づいて字句解析を行います。

HLEX CreateParsEnv (rLEXI li); // 字句解析情報の作成

字句規則を与えて字句解析情報（字句解析のコンテキストを管理するオブジェクト）を作成します。

li は、字句規則を定義する LEXI 構造体への参照です。戻り値は、成功時は有効な字句解析情報のハンドル、失敗時は NULL です。LEXI 構造体は、以下のように宣言されています。

```
typedef struct {  
    FTKN fspc; // 空白  
    FTKN fcns; // 定数リテラル  
    FTKN fstr; // 文字列リテラル  
    FTKN fkwd; // キーワードと識別子  
    FTKN fopr; // 演算子と区切り  
    uANY pa;   // 任意の付加情報  
} LEXI;
```

typedef LEXI *pLEXI, *PLEXI;

typedef const LEXI *rLEXI, *PCLEXI;

字句規則は、次の宣言で表される型のコールバック関数のセットで定義します。

```
typedef I4B (*FTKN) (rSTR *pp, pBIN bf, uANY pa);
```

この関数は、文字列上の現在位置を示すポインタ pp のアドレスを受け取って、現在位置に指定種別のトークンがあるかどうかを調べます。指定種別のトークンがあれば、その値を bf に格納し、現在位置を先に進めてトークン種別コードを返します。指定種別のトークンがなければ、現在位置はそのままにして 0 を返します。（空白やコメントを読み飛ばす場合、現在位置を先に進めて 0 を返すこともあります。）コールバック関数は上から順に呼び出され、全ての関数が 0 を返すと字句解析エラーとなります。

void DeleteParsEnv (HLEX le); // 字句解析情報の削除
字句解析情報を削除します。

HLEX SelectParsEnv (HLEX le); // 字句解析情報の選択
字句解析情報を選択します。戻り値は、関数を呼び出す前に選択されていた字句解析情報のハンドル（選択されていなければ **NULL**）です。

void start (rSTR pt); // 開始位置の設定
字句解析の開始位置を与えます。

void restart (rSTR pt); // 現在位置の再設定
現在位置を元に戻して字句解析をやり直します。

rSTR cpos (void); // 現在位置
現在位置を返します。

pANY buf (void); // 読込バッファ
直前に読み込まれたトークンの値が格納されているバッファのアドレスを返します。

I4B token (void); // トークン
トークンを読み込み現在位置を先に進めます。戻り値はトークン種別コードです。

BOOL token_is (I4B cd); // トークン (先読み)
現在位置に指定種別のトークンがある場合のみ、トークンを読み込み現在位置を先に進めます。

7.3 BOLBAS.LIB

BOLBAS.LIB は、スクリプト言語のバイトコードインタプリタ機能を提供します。

このモジュールを使用するアプリケーション本体は、開始時と終了時に以下の関数を呼ばなければなりません。

BOOL InitBOLBAS(uANY pa); // 開始処理

BOLBAS.LIB モジュールを初期化します。pa は、ダミーの引数です。0 を与えてください。

void TermBOLBAS(void); // 終了処理

BOLBAS.LIB モジュールの終了処理を行います。

以下の関数は、バイトコードインタプリタ機能を提供します。

BOOL RegisterAPIs(rBFNS bf); // ビルトイン関数の登録

C 言語で記述された A P I 関数をスクリプト言語から呼び出して実行できるように登録します。

スクリプト言語から A P I を呼ぶには、バイトコードインタプリタの仮想スタック上から C 関数を呼ぶラッパー関数を A P I ごとに用意する必要があります。ラッパー関数は、次の宣言で表される型の関数として定義します。

```
typedef void (*BFNC) (pBIN sp);
```

sp は仮想スタックのトップを指すポインタです。例えば、strcpyn のラッパー関数は、

```
void fi_strcpyn(pBIN sp)
```

```
{
    *(pP4B) (sp + 8) = strcpyn(*(pP4B) sp, *(rP4B) (sp + 4), *(rI4B) (sp + 8));
}
```

のように定義されています。

bf は、ラッパー関数のアドレスの配列を記述する BFNS 構造体への参照です。

BFNS 構造体は、以下のように宣言されています。

```
typedef struct {
    BFNC *fn; // 関数ポインタの配列の先頭アドレス
    I4B ni; // 配列の要素数
} BFNS;
```

```
typedef BFNS *pBFNS, *PBFNS;
```

```
typedef const BFNS *rBFNS, *PCBFNS;
```

ビルトイン関数の序数は、この関数での登録順で決定されます。

HBCD LoadBCD(rSTR nam, I4B app, I4B pro, I4B ssz); // バイトコードファイル読込

ファイル名 nam のバイトコードファイルを読み込みます。

app と pro は、アプリケーションとプロシージャ種別の確認が必要な場合に指定します。

ssz は、スタックサイズの指定です。0 にしておくと既定の128KBとなります。

戻り値は、成功時は実行可能なメモリ内のバイトコードのハンドル、失敗時は NULL です。

void DeleteBCD(HBCD hb); // バイトコードの削除

```
#define UnloadBCD(hb) DeleteBCD(hb)
```

メモリ内のバイトコードを削除します。

void CallBCD(pANY pv, HBCD hb, I4B ent, rANY arg); // バイトコードの実行

メモリ内のバイトコードの序数 ent の関数を実行します。

void ExecBCD(pANY pv, HBCD hb, rANY arg); // バイトコードの実行

メモリ内のバイトコードのメイン関数を実行します。

7.4 BOLCOM.LIB

BOLCOM.LIB は、スクリプト言語のコンパイラ機能を提供します。

このモジュールを使用するアプリケーション本体は、開始時と終了時に以下の関数を呼ばなければなりません。

```
BOOL InitBOLCOM(uANY pa); // 開始処理
```

BOLCOM.LIB モジュールを初期化します。pa は、ダミーの引数です。0 を与えてください。

```
void TermBOLCOM(void); // 終了処理
```

BOLCOM.LIB モジュールの終了処理を行います。

コンパイラ機能を提供するのは、次の関数です。

```
BOOL CompileBCD(rCPLI cp); // コンパイル
```

ソースコードをコンパイルしてバイトコードを作成します。

cp は、コンパイラに対する設定事項を記述する CPLI 構造体への参照です。

CPLI 構造体は、以下のように宣言されています。

```
typedef struct {  
    rSTR hfn; // ヘッダファイル名  
    rSTR sfm; // ソースコードファイル名  
    rSTR ofn; // バイトコードファイル名  
    rSTR mai; // メイン関数宣言  
    I4B app; // アプリケーション識別コード  
    I4B pro; // プロシージャ種別識別コード  
    rSTR tit; // タイトル文字列  
    rSTR dsc; // 説明文字列  
    HBLK hxb; // ヘッダファイルのテキストブロックハンドル  
    HBLK sxb; // ソースコードファイルのテキストブロックハンドル  
    HBCD *ph; // バイトコードのハンドルへのポインタ  
    I4B opt; // デバッグ用  
} CPLI;
```

```
typedef CPLI *pCPLI, *PCPLI;
```

```
typedef const CPLI *rCPLI, *PCPCPLI;
```

hfn は、ヘッダファイル名です。この場合は“BOLYAI_C.H”を与えます。ヘッダファイルがすでにロードされておりそのテキストブロックハンドルがあるときは、hfn は NULL にして hxb にヘッダファイルのテキストブロックハンドルを渡します。hfn を与えるときは、hxb は NULL にします。

sfm は、ソースコードファイル名です。ソースコードファイルがすでにロードされておりそのテキストブロックハンドルがあるときは、sfm は NULL にして sxb にソースコードファイルのテキストブロックハンドルを渡します。

sfm を与えるときは、sxb は NULL にします。

ofn は、出力先のバイトコードファイル名です。出力をファイルに書き出さず直接メモリ内に実行可能なバイトコードを作成するときは、ofn は NULL にして ph に出力を受け取る HBCD 型の変数のアドレスを渡します。

ofn も ph も NULL にすると、文法のチェックのみ行うことになります。

mai には、メイン関数宣言を文字列で与えます。

app と pro は、アプリケーションとプロシージャ種別の識別に用います。

tit は、タイトルにする32byte以内の任意文字列、dsc は、説明にする64byte以内の任意文字列です。

opt は、0 にしてください。

戻り値は、コンパイルの成功／失敗を TRUE / FALSE で示します。

7.5 BOLDBE.LIB

BOLDBE.LIB は、データベースエンジン機能を提供します。

このモジュールを使用するアプリケーション本体は、開始時と終了時に以下の関数を呼ばなければなりません。

BOOL InitBOLDBE(uANY pa); // 開始処理

BOLDBE.LIB モジュールを初期化します。pa は、ダミーの引数です。0 を与えてください。

void TermBOLDBE(void); // 終了処理

BOLDBE.LIB モジュールの終了処理を行います。

7.5.1 データベース管理関数（ネイティブアプリケーション用）

以下の関数は、サイト内のデータベースファイル等を操作します。

HTBL CreateBSATable(I4B cap, BOOL ld); // ユーザー管理表の作成

ユーザー管理表を作成します。ユーザー管理表のファイル名は、**サイト名.BSA** です。

cap は表の最大行数の初期値、ld はデータをファイルから読み込むかどうかの指定です。ld が FALSE のときは空の表が作成されます。

HTBL CreateBSDTable(I4B cap, I4B pro); // バイトコードリストの作成

サイト内にあるバイトコードファイルのリストを表として作成します。

cap は表の最大行数の初期値、pro はプロシージャ種別の指定です。pro が 0 のときは空の表が作成されます。

HTBL CreateBSUTable(I4B cap, BOOL ld); // ログイン中ユーザーリストの作成

サイトにログイン中のユーザー名のリストを表として作成します。表のファイル名は、**サイト名.BSU** です。

cap は表の最大行数の初期値、ld はデータをファイルから読み込むかどうかの指定です。ld が FALSE のときは空の表が作成されます。

HTBL CreateBDDTable(I4B cap, BOOL ld); // DB内の表リスト作成

現在接続中のデータベース内の表のリストを表として作成します。

データベースに未接続、または ld が FALSE のときは空の表が作成されます。

HTBL CreateBDUTable(I4B cap, BOOL ld); // DB接続中ユーザーリストの作成

データベースに接続中のユーザー名のリストを表として作成します。表のファイル名は、**DB名.BDU** です。

データベースに未接続、または ld が FALSE のときは空の表が作成されます。

HTBL CreateLCKTable(I4B cap, BOOL ld); // ロック情報管理表の作成

データベース内の表のロック情報を表として作成します。表のファイル名は、**DB名.LCK** です。

データベースに未接続、または ld が FALSE のときは空の表が作成されます。

BOOL LoginSite(rSTR usr, rSTR pwd); // ログイン

ユーザー名 **usr** パスワード **pwd** でサイトにログインします。戻り値は、ログインの成功／失敗を示します。

void LogoutSite(void); // ログアウト

サイトからログアウトします。サイトにログインしていない状態でこの関数を実行しても害はありません。

void ForceLogoutSite(rSTR usr); // 強制ログアウト

ユーザー **usr** をサイトから強制的にログアウトさせます。

HOPN CreateHOPN(HTBL tb, BOOL rw); // 表オープン情報の作成

表ハンドル **tb** から表オープン情報ハンドルを作成します。

rw は、編集モード／読取専用モードの別を TRUE / FALSE で指定します。

HOPN OpenBSATable(BOOL rw); // ユーザー管理表を開く

ユーザー管理表を開き表オープン情報ハンドルを返します。

rw は、編集モード／読取専用モードの別を TRUE / FALSE で指定します。

BOOL CallCommandMain(rSTR nam, HWND hwnd); // 一括処理バイトコードの実行
一括処理プロシージャのバイトコードをファイルからロードしてメイン関数を実行します。
実行が終了したらバイトコードはアンロードされます。
nam はプロシージャ名 (バイトコードのファイル名から拡張子を外したもの) です。

BOOL LoadReportBCD(rSTR nam); // 帳票作成バイトコードファイル読込
帳票作成プロシージャのバイトコードをファイルからロードします。
nam はプロシージャ名 (バイトコードのファイル名から拡張子を外したもの) です。

void UnloadReportBCD(void); // 帳票作成バイトコードの削除
ロードした帳票作成プロシージャのバイトコードをアンロードします。

BOOL CallReportMain(pANY rp, I4B msg, I4B pg, uANY pa); // 帳票作成バイトコードの実行
帳票作成プロシージャのバイトコードのメイン関数を実行します。
この関数は、帳票作成プロシージャのバイトコードがロードされた状態でのみ実行できます。

7.5.2 データファイル名

本ソフトで使用するデータファイル名をまとめると以下のようになります。

(サイト内のファイル)

BOLENV.INI	環境設定情報
サイト名.BSA	ユーザー管理表
サイト名.BSU	サイトにログイン中のユーザー名のリスト
サイト名.LOG	ユーザーのログイン／ログアウト履歴
DB名.BCF	DB定義プロシージャのバイトコード
DB名.BDU	データベースに接続中のユーザー名のリスト
DB名.LCK	ファイルのロック情報
表名.BTB	表データ
表名.BTC	表データ (暗号化)
プロシージャ名.BC0	一括処理プロシージャのバイトコード
プロシージャ名.BCR	帳票作成プロシージャのバイトコード

(クライアントのユーザーフォルダ内のファイル)

BOLYAI.INI	オプション設定情報
サイト名.LOG	エラーログ
表名.VIW	ユーザー定義のビューセット

7. 6 BOLEXT.LIB

BOLEXT.LIB は、数値計算と画像処理のための関数群を提供します。

このモジュールを使用するアプリケーション本体は、開始時と終了時に以下の関数を呼ばなければなりません。

```
BOOL InitBOLEXT(uANY pa); // 開始処理
```

BOLEXT.LIB モジュールを初期化します。pa は、ダミーの引数です。0 を与えてください。

```
void TermBOLEXT(void); // 終了処理
```

BOLEXT.LIB モジュールの終了処理を行います。

7.7 BOLFRO.LIB

BOLFRO.LIB は、汎用のユーザーインタフェース機能を提供します。

このモジュールを使用するアプリケーション本体は、開始時と終了時に以下の関数を呼ばなければなりません。

BOOL InitBOLFRO(uANY pa); // 開始処理

BOLFRO.LIB モジュールを初期化します。pa は、ダミーの引数です。0 を与えてください。

void TermBOLFRO(void); // 終了処理

BOLFRO.LIB モジュールの終了処理を行います。

7.7.1 パレット／フォント／ビットマップ関数

以下の関数は、アプリケーション用の既定のパレットを操作します。既定のパレットには 64 色があらかじめ設定されておりそれぞれの色には文字コードが与えられています。既定のパレット定義は、配布メディアの SDK フォルダ内の COLOR.HTM ファイルで確認できます。

void SetAppColor(I4B cid, BGRC col); // パレット色設定

パレットに色を設定します。cid は色のインデックスです。

BGRC AppColor(I4B cid); // 色 (COLORREF)

パレットの色を返します。

char AppColorChar(I4B cid); // 色を表す文字

パレットの色を表す文字を返します。

I4B AppColorIndex(I1B ch); // 文字が表す色のインデックス

文字が表す色のインデックスを返します。

I4B NearestAppColorIndex(BGRC col); // 最も近い色のインデックス

パレット内で与えられた色に最も近い色のインデックスを返します。

HPEN AppPen(I4B cid); // ペン

色のインデックスを指定してその色のペンのハンドルを取得します。

HBRUSH AppBrush(I4B cid); // ブラシ

色のインデックスを指定してその色のブラシのハンドルを取得します。

以下の関数は、アプリケーション用の既定のフォントを操作します。

void SetAppFont(I4B fid, HFONT hfnt); // フォントの登録

フォントを登録します。fid はフォントのインデックスです。

HFONT AppFont(I4B fid); // フォント

フォントのハンドルを取得します。

以下の関数は、メニュー等に用いるビットマップを操作します。

HDIB CreateDIBFromSTR(I4B w, I4B h, rSTR bi); // 文字配列によるデバイス独立ビットマップの作成

HBITMAP CreateDDBFromSTR(I4B w, I4B h, rSTR bi); // 文字配列による画面表示用ビットマップの作成

既定のパレットの色を表す文字コードを並べた文字配列によってビットマップを作成します。

横 w ピクセル×縦 h ピクセルのビットマップイメージを文字配列 bi[w*h] で表します。

void SetAppDDB(I4B bid, HBITMAP hbmp); // 画面表示用ビットマップの登録

ビットマップを登録します。bid はビットマップのインデックスです。

HBITMAP AppDDB(I4B bid); // 画面表示用ビットマップ

ビットマップのハンドルを取得します。

`void SetMenuItemDDB(I4B idm, HBITMAP hbmp);` // メニュー用ビットマップの設定
メニュー項目 I Dに画面表示用ビットマップのハンドルを設定します。

`HBITMAP MenuItemDDB(I4B idm);` // メニュー用ビットマップ
メニュー項目 I Dに設定された画面表示用ビットマップのハンドルを返します。

7.7.2 メニュー／ツールバー／ステータスバー関数

以下の関数は、メニュー、ツールバー、ステータスバーを作成します。

`HMENU CreateMenuBar(rMNUS ms);` // メニューの作成

定義情報を与えてメニューを作成します。

`ms` は、メニューの構成を定義する `MNUS` 構造体への参照です。

`MNUS` 構造体は、`MNUI` 構造体の配列を記述します。

```
typedef struct {
    I4B   flg; // 項目の属性
    I4B   idm; // 項目 I D
    rSTR  txt; // 項目文字列
    U4B   rsv; // 未使用
} MNUI;
typedef MNUI *pMNUI, *PMNUI;
typedef const MNUI *rMNUI, *PCMNUi;
typedef struct {
    pMNUI mi; // MNUI 構造体の配列の先頭アドレス
    I4B   ni; // 配列の要素数
} MNUS;
typedef MNUS *pMNUS, *PMNUS;
typedef const MNUS *rMNUS, *PCMNUi;
項目の属性は、以下のマクロで指定します。
mf_   通常の項目
mfG   グレー表示項目
mfP   ポップアップ項目
mfE   最後の項目
mfPE  ポップアップ項目で最後の項目
```

`HWND CreateToolBar(HWND hwnd, I4B id, rTOBS ts);` // ツールバーの作成

`HWND CreateStatBar(HWND hwnd, I4B id, rTOBS ss);` // ステータスバーの作成

定義情報を与えてツールバー／ステータスバーを作成します。

`id`はウィンドウ I D、`ts`／`ss`はバーの構成を定義する `TOBS` 構造体への参照です。

`TOBS` 構造体は、`TOBI` 構造体の配列を記述します。

```
typedef struct {
    I4B   flg; // 項目の属性
    I4B   idm; // 項目 I D
    rSTR  txt; // 項目文字列
    U4B   rsv; // 未使用
} TOBI;
typedef TOBI *pTOBI, *PTOBI;
typedef const TOBI *rTOBI, *PCTOBI;
typedef struct {
    pTOBI pi; // TOBI 構造体の配列の先頭アドレス
    I4B   ni; // 配列の要素数
} TOBS;
typedef TOBS *pTOBS, *PTOBS;
typedef const TOBS *rTOBS, *PCTOBS;
項目の属性は、以下のマクロで指定します。
tbfV   縦線のセパレータ
tbfB   ボタン
tbfS(w) 幅 w ドットのスペース
tbfL(w) 幅 w ドットの左寄せ文字列項目
tbfC(w) 幅 w ドットの中央揃え文字列項目
tbfR(w) 幅 w ドットの右寄せ文字列項目
```

7.7.3 ダイアログボックス関数

以下の関数は、ダイアログボックスを作成します。

I4B DlgBoxInput(HWND hwnd, rDLGS ds, DLGPROC lpfn, iANY lpa); // モーダルダイアログボックス

ダイアログボックスを画面に表示し、ユーザーの入力を待ちます。

hwnd はオーナーウィンドウのハンドル、ds はダイアログボックスの構成を定義する DLGS 構造体への参照、lpfn はダイアログプロシージャのアドレス、lpa は WM_INITDIALOG メッセージのパラメータです。

DLGS 構造体は、DLGI 構造体の配列とその他の付加情報を記述します。

```
typedef struct {
    rSTR cls; // コントロールのウィンドウクラス
    I4B sty; // コントロールのウィンドウスタイル
    U2B x, y; // コントロールの左上隅座標(ダイアログ単位)
    U2B w, h; // コントロールの幅と高さ(ダイアログ単位)
    I4B id; // 項目 I D
    rSTR txt; // 項目文字列
} DLGI;
typedef DLGI *pDLGI, *PDLGI;
typedef const DLGI *rDLGI, *PCDLGI;
typedef struct {
    pDLGI di; // DLGI 構造体の配列の先頭アドレス
    I4B ni; // 配列の要素数
    U2B x, y; // ダイアログボックスの左上隅座標(ダイアログ単位)
    U2B w, h; // ダイアログボックスの幅と高さ(ダイアログ単位)
    rSTR tit; // ダイアログボックスのタイトル文字列
} DLGS;
typedef DLGS *pDLGS, *PDLGS;
typedef const DLGS *rDLGS, *PCDLGS;
```

I4B PropSheetInput(HWND hwnd, rPSHS ps, I4B ip); // プロパティシート

複数のダイアログボックスをまとめてプロパティシートのように動作させます。

ps はプロパティシートの構成を定義する PSHS 構造体への参照、ip は初期ページの指定です。

PSHS 構造体は、PSHI 構造体の配列を記述します。

```
typedef struct {
    rDLGS ds; // DLGS 構造体への参照
    DLGPROC lpfn; // ダイアログプロシージャ
    iANY lpa; // パラメータ
    U4B psz; // オブジェクトのサイズ(パラメータがポインタのとき)
} PSHI;
typedef PSHI *pPSHI, *PPSHI;
typedef const PSHI *rPSHI, *PCPSHI;
typedef struct {
    pPSHI pp; // PSHI 構造体の配列の先頭アドレス
    I4B np; // 配列の要素数
} PSHS;
typedef PSHS *pPSHS, *PPSHS;
typedef const PSHS *rPSHS, *PCPSHS;
```

BOOL OpenFileDialog(HWND hwnd, pOFNS os); // 開くダイアログボックス

BOOL SaveFileDialog(HWND hwnd, pOFNS os); // 保存ダイアログボックス

Windows の A P I 関数 `GetOpenFileName` / `GetSaveFileName` のラッパー関数です。

OFNS 構造体は、以下のように宣言されています。

```
typedef struct {
    rSTR fil; // フィルター文字列(lpstrFilter)
    I4B idx; // フィルター文字列の既定値(nFilterIndex)
    pSTR nam; // ファイル名(lpstrFile)
    I4B flg; // 初期化フラグ(Flags)
} OFNS;
typedef OFNS *pOFNS, *POFNS;
```


BOOL PrintDlgBox(HWND hwnd, PPRDS ps); // 印刷ダイアログボックス

WindowsのAPI関数PrintDlgのラッパー関数です。

PRDS構造体は、以下のように宣言されています。

```
typedef struct {
    HDC   hdc; // デバイスコンテキスト(hDC)
    I4B   flg; // 初期化フラグ(Flags)
    U2B   nf, nt; // 開始ページ(nFromPage), 終了ページ(nToPage)
    U2B   np, nc; // ページ最大値(nMaxPage), 部数(nCopies)
    U2B   psz, ori; // 用紙サイズ(dmPaperSize), 用紙の向き(dmOrientation)
    U2B   wid, len; // 用紙の幅(dmPaperWidth), 用紙の長さ(dmPaperLength) 1/10mm単位
} PRDS;
typedef PRDS *pPRDS, *PPRDS;
```

BOOL PageSetupDlgBox(HWND hwnd, PSDS ps); // 印刷用紙設定ダイアログボックス

印刷用紙の設定をします。

PSDS構造体は、以下のように宣言されています。

```
typedef struct {
    I4B   adj; // フラグ
    I4B   apw; // 横幅合わせ基準値
    I4B   opw, oph; // 出力用紙サイズ 1/1000mm単位
    I4B   ppw, pph; // 原稿用紙サイズ
    I4B   lpm, tpm, rpm, bpm; // 用紙の余白サイズ
} PSDS;
typedef PSDS *pPSDS, *PPSDS;
typedef const PSDS *rPSDS, *PCPSDS;
```

7.7.4 帳票作成関数（ネイティブアプリケーション用）

以下の関数は、レポート情報オブジェクトを操作（帳票作成プロシージャを駆動）します。

HREP CreateReport(rREPI ri); // レポート情報作成

レポート情報を作成し、帳票作成プロシージャにREPM_CREATEメッセージを送ります。

riは、作成情報を与えるREPI構造体への参照です。

REPI構造体は、以下のように宣言されています。

```
typedef struct {
    rSTR   tit; // タイトル文字列
    I4B   flg; // 用紙サイズと用紙の余白サイズの初期値指定
    FREP   frep; // 帳票作成プロシージャ
    uANY   pa; // 付加情報
} REPI;
typedef REPI *pREPI, *PREPI;
typedef const REPI *rREPI, *PCREPI;
```

flgは、用紙サイズと用紙の余白サイズの初期値を以下のフラグ値の組み合わせで指定します。

PSZ_A3P	A 3 縦	PSZ_A3L	A 3 横
PSZ_B4P	B 4 縦	PSZ_B4L	B 4 横
PSZ_A4P	A 4 縦	PSZ_A4L	A 4 横
PSZ_B5P	B 5 縦	PSZ_B5L	B 5 横
PSZ_A5P	A 5 縦	PSZ_A5L	A 5 横
PSZ_B6P	B 6 縦	PSZ_B6L	B 6 横
PSZ_A6P	A 6 縦	PSZ_A6L	A 6 横
PMG_REG	標準（省略時既定値）		
PMG_MWD	やや広い	PMG_WID	広い
PMG_XWD	かなり広い	PMG_XXW	とても広い
PMG_MNR	やや狭い	PMG_NAR	狭い
PMG_XNR	かなり狭い	PMG_XXN	とても狭い

frepは、次の宣言で表される型のコールバック関数のアドレスです。

```
typedef pANY (*FREP) (HREP rp, I4B msg, I4B pg, uANY pa);
```

この関数（帳票作成プロシージャ）が、ReportMain関数に相当します。

void DeleteReport(HREP rp); // レポート情報削除

帳票作成プロシージャに REPM_DESTROY メッセージを送って、レポート情報を削除します。

void StartPrint(HREP rp, HDC hdc, I4B dpi); // 描画開始処理

描画環境をセットアップし、帳票作成プロシージャに REPM_STARTPRINT メッセージを送ります。

hdc は、描画先のデバイスコンテキストのハンドルです。

dpi は、解像度の指定です。0 にしておくとデバイスの解像度が用いられます。異なる解像度を指定すると描画内容全体をスケーリングする効果があります。

void EndPrint(HREP rp); // 描画終了処理

帳票作成プロシージャに REPM_ENDPRINT メッセージを送ります。

void DrawPage(HREP rp, I4B pg); // ページ描画

帳票作成プロシージャに REPM_DRAWPAGE メッセージを送ります。

void SkipPage(HREP rp, I4B pg); // ページスキップ

帳票作成プロシージャに REPM_SKIPPAGE メッセージを送ります。

BOOL ReportSetup(HWND hwnd, HREP rp); // 印刷データ設定

帳票作成プロシージャに REPM_SETUP メッセージを送ります。

BOOL PageSetup(HWND hwnd, HREP rp); // 印刷用紙設定

印刷用紙設定ダイアログボックスを表示し、ユーザーの入力を待ちます。

「OK」ボタンが押されたら、帳票作成プロシージャに REPM_PAGESETUP メッセージを送ります。

BOOL PrintReport(HWND hwnd, HREP rp); // 印刷

印刷ダイアログボックスを表示し、ユーザーの入力を待ちます。

「OK」ボタンが押されたら、印刷を実行します。

7.8 BOLGUI.LIB

BOLGUI.LIB は、専用のユーザーインタフェース機能を提供します。

このモジュールを使用するアプリケーション本体は、開始時と終了時に以下の関数を呼ばなければなりません。

`BOOL InitBOLGUI(uANY pa); // 開始処理`

BOLGUI.LIB モジュールを初期化します。pa は、ダミーの引数です。0 を与えてください。

`void TermBOLGUI(void); // 終了処理`

BOLGUI.LIB モジュールの終了処理を行います。

7.8.1 開始メニュー

開始メニューは、次の関数で作成します。

`I4B MenuDlgBox(HWND hwnd, pMNDI pa, rSTR tit); // 開始メニュー`

開始メニューダイアログボックスを作成します。

hwnd はオーナーウィンドウのハンドル、pa はメニューの構成を定義する MNDI 構造体への参照、tit はタイトル文字列です。

MNDI 構造体は、MNDI 構造体の配列とその他の付加情報を記述します。

```
typedef struct {
    I4B flg; // 項目の属性
    I4B idm; // 項目 I D
    rSTR txt; // 項目文字列
    I4B bid; // ビットマップのインデックス
} MNDI;
typedef MNDI *pMNDI, *PMNDI;
typedef const MNDI *rMNDI, *PCMNDI;
typedef struct {
    pMNDI mi; // MNDI 構造体の配列の先頭アドレス
    I4B ni; // 配列の要素数
    I4B flg; // オプション指定
    I4B ip; // 初期ページ
    rSTR msg1; // 下部領域の表示文字列
    rSTR msg2; // 下部領域の表示文字列
} MNDS;
typedef MNDS *pMNDS, *PMNDS;
typedef const MNDS *rMNDS, *PCMNDS;
```

項目は、8 項目が 1 ページ分となり、ページを切り替えるコマンド `IDJMP + ip` でページ間を移動できます。

7.8.2 テキストウィンドウ

テキストウィンドウ（クラス名 “BOLTXT”）は、テキストブロックを画面上に表示して編集するための機能を提供するウィンドウクラスです。このウィンドウは、フレームウィンドウの子ウィンドウとして働きます。

コマンド選択に対する応答等のウィンドウの具体的な振る舞いは、アプリケーション側でサブクラス化して実装します。

既定のウィンドウプロシージャは、次の関数です。

`iANY DefBOLTXTProc(HWND hwnd, U4B umsg, uANY wpa, iANY lpa); // ウィンドウプロシージャ`

主なウィンドウメッセージは、以下の通りです。

<code>TXCM_SETMODE</code>	状態フラグ設定
<code>TXCM_GETMODE</code>	状態フラグ取得
<code>TXCM_SETTEXT</code>	テキストブロックハンドルの設定
<code>TXCM_GETTEXT</code>	テキストブロックハンドルの取得
<code>TXCM_SETCUR</code>	カーソル位置の設定
<code>TXCM_GETCUR</code>	カーソル位置の取得
<code>TXCM_SETTOP</code>	表示開始位置の設定
<code>TXCM_GETTOP</code>	表示開始位置の取得
<code>TXCM_UPDATE</code>	表示更新
<code>TXCM_MOVECUR</code>	カーソルの相対移動
<code>TXCM_SEARCH</code>	文字列検索

7.8.3 印刷プレビューウィンドウ

印刷プレビューウィンドウ（クラス名“BOLPRN”）は、印刷イメージを画面上で確認するための機能を提供するウィンドウクラスです。このウィンドウは、フレームウィンドウの子ウィンドウとして働きます。コマンド選択に対する応答等のウィンドウの具体的な振る舞いは、アプリケーション側でサブクラス化して実装します。

既定のウィンドウプロシージャは、次の関数です。

```
iANY DefBOLPRNProc(HWND hwnd, U4B umsg, uANY wpa, iANY lpa); // ウィンドウプロシージャ
```

主なウィンドウメッセージは、以下の通りです。

PRCM_SETMODE	状態フラグ設定
PRCM_GETMODE	状態フラグ取得
PRCM_SETREPORT	レポート情報ハンドルの設定
PRCM_GETREPORT	レポート情報ハンドルの取得
PRCM_SETSCALE	表示倍率の設定
PRCM_GETSCALE	表示倍率の取得
PRCM_SETPAGE	表示ページの設定
PRCM_GETPAGE	表示ページの取得

7.8.4 表ウィンドウ

表ウィンドウ（クラス名“BOLTBL”）は、表データを画面上に表示して編集するための機能を提供するウィンドウクラスです。このウィンドウは、フレームウィンドウの子ウィンドウまたはダイアログボックス内のコントロールとして働きます。

コマンド選択に対する応答等のウィンドウの具体的な振る舞いは、アプリケーション側でサブクラス化して実装します。

既定のウィンドウプロシージャは、次の関数です。

```
iANY DefBOLTBLProc(HWND hwnd, U4B umsg, uANY wpa, iANY lpa); // ウィンドウプロシージャ
```

主なウィンドウメッセージは、以下の通りです。

TBCM_SETMODE	状態フラグ設定
TBCM_GETMODE	状態フラグ取得
TBCM_SETTABLE	表ハンドルの設定
TBCM_GETTABLE	表ハンドルの取得
TBCM_SETCUR	カレントセルの設定
TBCM_GETCUR	カレントセルの取得
TBCM_SETTOP	表示開始位置の設定
TBCM_GETTOP	表示開始位置の取得
TBCM_SETSEL	範囲選択(選択解除)
TBCM_GETSEL	選択状態の取得
TBCM_UPDATE	表示更新
TBCM_EDITCELL	カレントセルを編集
TBCM_MOVECUR	カレントセルの相対移動
TBCM_ASKCELLFORMAT	セルの描画属性の問い合わせ
TBCM_GETCELLFORMAT	セルの描画属性の取得
TBCM_SETCELLTEXT	セルテキストの設定
TBCM_GETCELLTEXT	セルテキストの取得
TBCM_CHANGEVIEW	ビューの変更
TBCM_GETVIEW	ビューハンドルの取得
TBCM_SETCURROW	行バッファの内容をカレント行にコピー
TBCM_GETCURROW	カレント行の内容を行バッファにコピー
TBCM_INSERT	行ブロックの挿入
TBCM_DELETE	行ブロックの削除
TBCM_CLEAR	行ブロックのクリア
TBCM_SEARCH	文字列検索
TBCM_NSEARCH	次の文字列検索

7.8.5 フレームウィンドウ

フレームウィンドウ（クラス名“BOLFRM”）は、アプリケーションのメインウィンドウとして働き、複数のドキュメントウィンドウを切り替えて表示する機能を提供するウィンドウクラスです。

このウィンドウは、表ウィンドウ等のドキュメントウィンドウの親ウィンドウとなります。

コマンド選択に対する応答等のウィンドウの具体的な振る舞いは、アプリケーション側でサブクラス化して実装します。

既定のウィンドウプロシージャは、次の関数です。

```
iANY DefBOLFRMProc(HWND hwnd, U4B umsg, uANY wpa, iANY lpa); // ウィンドウプロシージャ
```

主なウィンドウメッセージは、以下の通りです。

FRMM_GETNCHILDREN	子ウィンドウの数
FRMM_GETCHILD	子ウィンドウのインデックスからハンドルを求める
FRMM_GETINDEX	子ウィンドウのハンドルからインデックスを求める
FRMM_GETACTIVE	現在アクティブな子ウィンドウのハンドルを求める
FRMM_ARRANGE	子ウィンドウの再配置
FRMM_ACTIVATE	子ウィンドウをアクティブ化する
FRMM_CREATECHILD	子ウィンドウを開く
FRMM_DESTROYCHILD	子ウィンドウの削除
FRMM_GETNEWID	未使用の子ウィンドウ I Dを求める
FRMM_GETCHILDBYNAME	子ウィンドウの名前からハンドルを求める
FRMM_SETFLAGS	状態フラグ設定
FRMM_GETFLAGS	状態フラグ取得

フレームウィンドウは、次の関数で作成します。

```
HWND CreateBOLFRMWindow(HINSTANCE hinst, rFRMI fi); // フレームウィンドウの作成
```

フレームウィンドウを作成します。

fi は、作成情報を与える FRMI 構造体への参照です。

FRMI 構造体は、以下のように宣言されています。

```
typedef struct {  
    rSTR  nam; // タイトル文字列  
    pWPOS wp;  // ウィンドウの初期配置情報  
    pMNUS ms;  // メニュー定義情報  
    I4B   tid; // ツールバーのウィンドウ I D  
    pTOBS ts;  // ツールバー定義情報  
    I4B   sid; // ステータスバーのウィンドウ I D  
    pTOBS ss;  // ステータスバー定義情報  
    uANY  ext; // 任意の付加情報  
} FRMI;
```

```
typedef FRMI *pFRMI, *PFRMI;
```

```
typedef const FRMI *rFRMI, *PCFRMI;
```

ウィンドウの初期配置情報は、WPOS 構造体で与えます。

```
typedef struct {  
    I4B  wx, wy; // 左上隅座標  
    I4B  ww, wh; // 幅と高さ  
    BOOL mx;     // 最大化する／しない  
} WPOS;  
typedef WPOS *pWPOS, *PWPOS;  
typedef const WPOS *rWPOS, *PCWPOS;
```

7.9 BOLPLS.LIB

BOLPLS.LIB は、SaveJPG / LoadJPG 関数を実装するために G D I + を使用しています。
このモジュールを使用するアプリケーション本体は、開始時と終了時に以下の関数を呼ばなければなりません。

`BOOL InitBOLPLS(uANY pa); // 開始処理`

BOLPLS.LIB モジュールを初期化します。pa は、ダミーの引数です。0 を与えてください。

`void TermBOLPLS(void); // 終了処理`

BOLPLS.LIB モジュールの終了処理を行います。