StkData

1. StkDataとは

1.1 StkData概要

StkDataは静的ライブラリとして提供されるリポジトリ管理プログラムで，StkData利用プログラムに対して以下の機能を提供します。

- テーブルの追加
- テーブルの削除
- レコードの追加
- レコードの削除
- レコードの更新
- レコードの検索／取得
- レコードのソート
- テーブル情報，レコード情報の取得
- テーブル／レコードのファイルへの書き込み
- テーブル／レコードのファイルからの読み取り

1.2 StkDataの特徴

StkDataは以下の特徴を持ちます。

- StkDataの全ての機能は静的ライブラリで提供されるため，StkData利用プログラムに容易に組み込むことができます。このためユーザーは他のDBMSを利用する場合に比べ，事前インストール／セットアップの手間から解放されます。
- StkDataが管理する全てのデータは常に仮想メモリー上に展開され処理されます。また，全てのStkData提供機能(API)は静的ライブラリのネイティブ関数として呼び出すことができます。これによりデータアクセスに伴うオーバーヘッドは最小限となり，高速なデータアクセスが可能となります。

1.3 要件

StkDataは以下の環境で動作します。

- 動作に必要なCPU:
  1GHz以上のクロック周波数で動作するCPU

- 動作に必要なメモリ容量:
  512MBytes以上

- 導入に必要な空きディスク容量:
  10MBytes以上

- サポートするオペレーティングシステム:
  Windows:
  Windows 2000 Professional SP4
  Windows XP Professional SP2/SP3
  Windows XP Professional x64 SP2
  Windows Vista Business/Ultimate SPなし/SP1/SP2
  Windows 7 Professional/Ultimate SPなし
  Windows Server 2003 Standard SPなし/SP1/SP2
  Windows Server 2008 Standard SPなし/SP2

- 前提となるソフトウェア:

Windows:

Microsoft Visual Studio 2008 (Visual C++)

2. StkDataの導入

2.1 ファイル一覧

StkDataを構成するファイルは以下のとおりです。

ファイル名	説明	ビルド時	実行時
stkdata.h	StkDataのクラス，関数の定義を記述したヘッダファイルです。StkData利用プログラムのソースコードから#include文で読込んでください。	要	不要
stkdata.lib	StkDataの機能を提供する静的ライブラリ（StkData本体）です。本ファイルはStkData利用プログラムのビルド時に，StkData利用プログラムに取り込まれます。	要	不要

2.2 ビルド方法

StkData利用プログラムのソースコードからstkdata.hをインクルードしてください。また，StkData利用プログラムのビルド時にstkdata.libを静的リンクするようプロジェクトの設定を変更してください。

StkData利用プログラムはWin32用アプリケーションとしてビルドされる必要があります。
StkDataは，StkData利用プログラムがMicrosoftのVisual C++で記載されることを想定しています。
推奨バージョンは以下のとおりです。

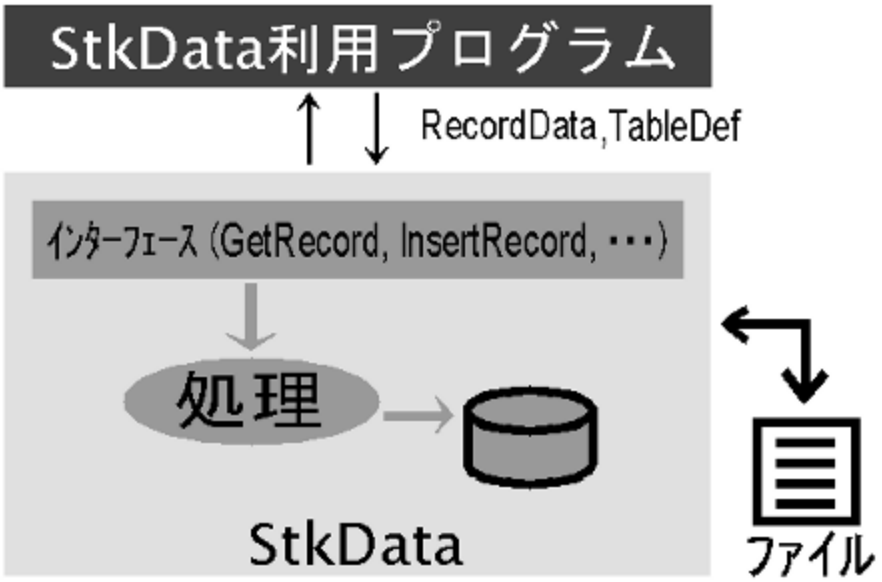
- Microsoft Visual Studio 2005 (Visual C++)
- Microsoft Visual Studio 2008 (Visual C++)

上記以外でも，MicrosoftのWindowsオペレーティングシステム用のプログラムを生成可能なC++言語であれば StkData利用プログラムをビルドすることが可能です。

3. StkData詳細説明

3.1 StkData概念図

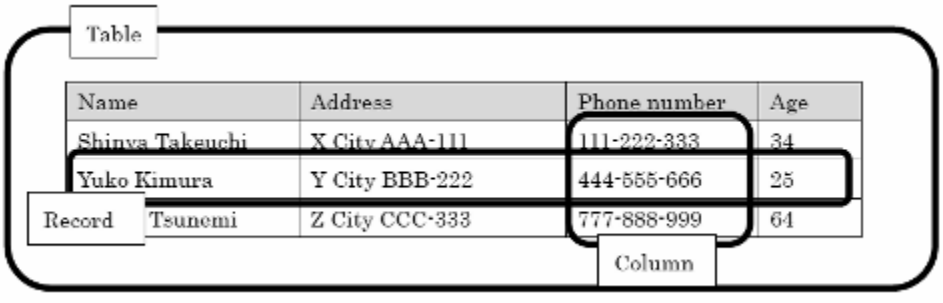
以下にStkDataの概念図を示します。



StkData利用プログラムは，StkDataが提供するインターフェース(API)を介して データにアクセスします。
StkDataとStkData利用プログラム間のデータの受け渡しには RecordDataインスタンスが使用されます。
StkData利用プログラムからStkDataへのテーブル定義要求には，TableDefインスタンスが 使用されます。
StkDataのデータは常にメモリ上に展開されているため，通常StkDataが動作するプロセス が終了したときに当該データは失われます。ただし，SktDataはファイル入出力用の APIを提供しており，必要に応じてメモリからファイルへの書込み，ファイルからメモリ への読込みを行うことができます。

3.2 StkDataが扱うデータ

StkDataはデータを下図のようなテーブル(Table)と呼ばれる入れ物で管理します。テーブルは複数のカラム(Column)から構成され，カラムには整数型，文字列型のように"型"を定義する ことができます。StkData利用プログラムはテーブルにレコード(Record)を登録することにより データを保持します。レコードはStkData利用プログラムが使用するデータと同意と考えることができます。



3.3 使用上の制限事項

StkDataには下記に示す使用上の制限事項があります。

登録可能なテーブル数	16

登録可能なカラム数／1テーブル	32
整数型カラムのバイト数	4
単精度浮動小数点数型カラムのバイト数	4
文字列型カラムに使用可能な文字コード	ASCII (0x20-0x7e)
ワイド文字列型カラムに使用可能な文字コード	UTF-16
文字列型カラムの最小文字数	2 (最終文字NULL文字を含む)
文字列型カラムの最大文字数	256 (最終文字NULL文字を含む)
ワイド文字列型カラムの最小文字数	2 (最終文字NULL文字を含む)
ワイド文字列型カラムの最大文字数	256 (最終文字NULL文字を含む)
バイナリ型カラムの最小バイト数	1
バイナリ型カラムの最大バイト文字数	10000000
最小レコード数／1テーブル	1
最大レコード数／1テーブル	16383
テーブル名バイト数	16 (最終文字NULLを含む)
テーブル名に使用可能な文字コード	UTF-16
カラム名バイト数	16 (最終文字NULLを含む)
カラム名に使用可能な文字コード	UTF-16

3.4 StkDataで使用する定数

StkDataでは以下の定数を使用します。

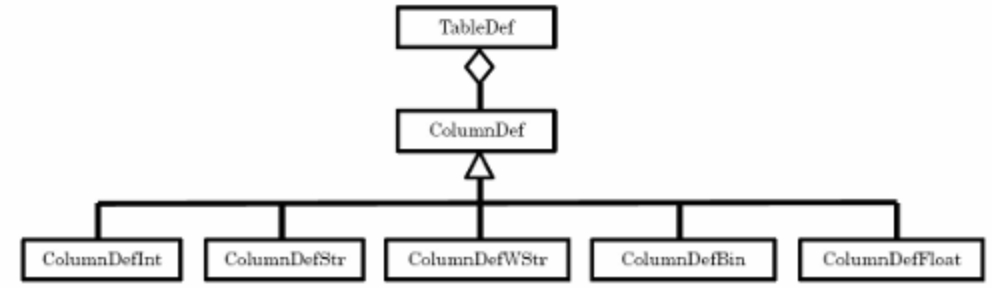
MAX_TABLE_NUMBER	登録可能なテーブル数	16
MAX_COLUMN_NUMBER	登録可能なカラム数	32
MAX_RECORD	最大レコード数／1テーブル	16383
LOCK_FREE	テーブルロック開放状態	0
LOCK_SHARE	テーブル共有ロック状態	1
LOCK_EXCLUSIVE	テーブル排他ロック状態	2
COLUMN_TYPE_INT	整数型カラムを示す定数	0
COLUMN_TYPE_STR	文字列型カラムを示す定数	1
COLUMN_TYPE_WSTR	ワイド文字列型カラムを示す定数	2
COLUMN_TYPE_BIN	バイナリ型カラムを示す定数	3
COLUMN_TYPE_FLOAT	単精度浮動小数点数型	4
TABLE_NAME_SIZE	テーブル名の最大長	16
COLUMN_NAME_SIZE	カラム名の最大長	16

4. データモデル

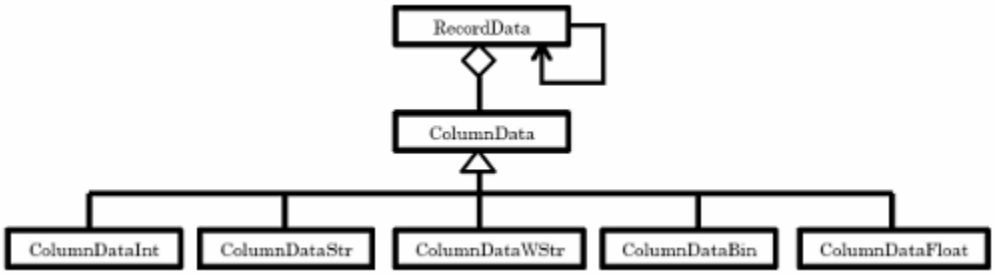
StkDataが扱うデータモデルには，大きく分けて下記の2種類があります。

テーブル定義用データモデル	TableDef ColumnDef ColumnDefInt ColumnDefStr ColumnDefWStr ColumnDefBin ColumnDefFloat
データアクセス用データモデル	RecordData ColumnData ColumnDataInt ColumnDataStr ColumnDataWStr ColumnDataBin ColumnDataFloat

テーブル定義用データモデルのクラス関連図（データモデル関連図）は下図のようになります。
TableDefはテーブルを定義するデータモデルで，ColumnDefを集約します。ColumnDefはカラムを定義するデータモデルで，整数型カラム(ColumnDefInt)，文字列型カラム(ColumnDefStr)，ワイド文字列型カラム(ColumnDefWStr)，バイナリ型カラム(ColumnDefBin)，単精度浮動小数点数型カラム(ColumnDefFloat) に派生します。



また，データアクセス用データモデルのクラス関連図は下図のようになります。
RecordDataはレコードを示すデータモデルで，ColumnDataを集約します。ColumnDataはカラムを示すデータモデルで，整数型カラム(ColumnDataInt)，文字列型カラム(ColumnDataStr)，ワイド文字列型カラム(ColumnDataWStr)，バイナリ型カラム(ColumnDataBin)，単精度浮動小数点数型カラム(ColumnDataFloat) に派生します。
StkDataが公開するAPIは，1つの関数呼び出しで複数のレコード操作を行うことから，RecordDataは線形リスト型のデータ構造となります。複数のRecordDataを扱うときは，RecordDataインスタンスが別の RecordDataインスタンスへのポインタを保持するようにします。



4.1 TableDef

テーブルを定義するためのクラス（データモデル）です。

4.1.1 TableDefのメンバ変数

private

TCHAR	m_TableName[TABLE_NAME_SIZE]	テーブル名
ColumnDef*	m_Column[MAX_COLUMN_NUMBER]	カラムの定義を格納する配列
int	m_NumberOfColumn	インスタンスを構成する実際のカラム数
int	m_MaxRecord	登録可能最大レコード数

4.1.2 TableDefの関数

(1) TableDef()

デフォルトコンストラクタ。

(2) TableDef(TCHAR* TableName, int MaxRecord)

コンストラクタ。

TableName	テーブル名
MaxRecord	生成するテーブルに登録可能な最大レコード数

(3) virtual ~TableDef()

デストラクタ。

(4) void SetTableName(TCHAR* TableName)

テーブル名を指定します。TableNameが16文字(NULL含む)を超えた場合，17文字目以降の文字は切り取られます。

TableName	テーブル名

(5) TCHAR* GetTableName()

テーブル名を取得します。

戻り値	テーブル名

(6) void SetMaxRecord(int MaxRec)

登録可能最大レコード数を設定します。

MaxRec	最大レコード数

(7) int GetMaxRecord()

登録可能最大レコード数を取得します。

戻り値	最大レコード数

(8) int AddColumnDef(ColumnDef* ColDef)

カラムの定義を追加します。

ColDef	ColumnDefインスタンス

戻り値	0:正常終了 -1:登録可能なColumnDefインスタンスの最大値を超えました

(9) int DeleteLastColumnDef()

最後のカラムの定義を削除します。

戻り値	0:正常終了 -1:カラムの定義は存在しません

(10) ColumnDef* GetColumnDef(int Num)

Num番目のカラムの定義を取得します。

Num	登録されているカラムの定義のインデックス値
戻り値	ColumnDefインスタンス NULL:カラムの定義は存在しません

(12) int GetNumOfColumn()

登録されているカラムの定義の数を取得します。

戻り値	登録されているカラムの定義の数

4.2 ColumnDef

カラムを定義するためのクラス（データモデル）です。

4.2.1 ColumnDefのメンバ変数

protected

TCHAR	m_ColumnName[COLUMN_NAME_SIZE]	カラム名
int	m_ColumnType	カラム種別

4.2.2 ColumnDefの関数

(1) ColumnDef()
　　デフォルトコンストラクタ。

(2) virtual ~ColumnDef()

　　デストラクタ。

(3) void SetColumnName(TCHAR* ColumnName)

　　カラム名を指定します。ColumnNameが16文字(NULL含む)を超えた場合，17文字目以降の文字は切り取られます。

ColumnName	カラム名

(4) TCHAR* GetColumnName()

　　カラム名を取得します。

戻り値	カラム名

(5) void SetColumnType(int ColType)

　　カラムの種別を指定します。

ColType	COLUMN_TYPE_INT:整数型 COLUMN_TYPE_STR:文字列型 COLUMN_TYPE_WSTR:ワイド文字列型 COLUMN_TYPE_BIN:バイナリ型 COLUMN_TYPE_FLOAT:単精度浮動小数点数型

(6) int GetColumnType()

　　カラムの種別を取得します。

戻り値	COLUMN_TYPE_INT:整数型 COLUMN_TYPE_STR:文字列型 COLUMN_TYPE_WSTR:ワイド文字列型 COLUMN_TYPE_BIN:バイナリ型 COLUMN_TYPE_FLOAT:単精度浮動小数点数型

4.3 ColumnDefInt

整数型カラムを定義するためのクラス（データモデル）です。ColumnDefクラスの派生クラスとして定義されます。

4.3.1 ColumnDefIntの関数

(1) ColumnDefInt()

デフォルトコンストラクタ。

(2) ColumnDefInt(TCHAR* ColumnName)

コンストラクタ。

ColumnName	カラム名

(3) virtual ~ColumnDefInt()

デストラクタ。

4.4 ColumnDefStr

文字列型カラムを定義するためのクラス（データモデル）です。ColumnDefクラスの派生クラスとして定義されます。

4.4.1 ColumnDefStrのメンバ変数

private

int	m_MaxLength	最大文字列長（文字数）

4.4.2 ColumnDefStrの関数

(1) ColumnDefStr()

デフォルトコンストラクタ。

(2) ColumnDefStr(TCHAR* ColumnName, int Length)

コンストラクタ。

ColumnName	カラム名
Length	最大文字列長（文字数）

(3) virtual ~ColumnDefStr()

デストラクタ。

(4) void SetMaxLength(int MaxLength)

最大文字列長を設定します。

MaxLength	最大文字列長（文字数）

(4) int GetMaxLength()

最大文字列長を取得します。

戻り値	最大文字列長（文字数）

4.5 ColumnDefWStr

ワイド文字列型カラムを定義するためのクラス（データモデル）です。ColumnDefクラスの派生クラスとして定義されます。

4.5.1 ColumnDefWStrのメンバ変数

private

int	m_MaxLength	最大文字列長（文字数）

4.5.2 ColumnDefWStrの関数

(1) ColumnDefWStr()

デフォルトコンストラクタ。

(2) ColumnDefWStr(TCHAR* ColumnName, int Length)

コンストラクタ。

ColumnName	カラム名
Length	最大文字列長（文字数）

(3) virtual ~ColumnDefWStr()

デストラクタ。

(4) void SetMaxLength(int MaxLength)

最大文字列長を設定します。

MaxLength	最大文字列長（文字数）

(4) int GetMaxLength()

最大文字列長を取得します。

戻り値	最大文字列長（文字数）

4.6 ColumnDefBin

バイナリ型カラムを定義するためのクラス（データモデル）です。ColumnDefクラスの派生クラスとして定義されます。

4.6.1 ColumnDefBinのメンバ変数

private

int	m_MaxLength	最大文字列長（文字数）

4.6.2 ColumnDefBinの関数

(1) ColumnDefBin()

デフォルトコンストラクタ。

(2) ColumnDefBin(TCHAR* ColumnName, int Length)

コンストラクタ。

ColumnName	カラム名
Length	最大文字列長（文字数）

(3) virtual ~ColumnDefBin()

デストラクタ。

(4) void SetMaxLength(int MaxLength)

最大バイト数を設定します。

MaxLength	最大バイト数

(4) int GetMaxLength()

最大バイト数を取得します。

戻り値	最大バイト数

4.7 ColumnDefFloat

単精度浮動小数点数型カラムを定義するためのクラス（データモデル）です。ColumnDefクラスの派生クラスとして定義されます。

4.7.1 ColumnDefFloatの関数

(1) ColumnDefFloat()
　　デフォルトコンストラクタ。

(2) ColumnDefFloat(TCHAR* ColumnName)

　　コンストラクタ。

ColumnName	カラム名

(3) virtual ~ColumnDefFloat()

　　デストラクタ。

4.8 RecordData

StkDataのレコードにアクセスするためのクラス（データモデル）です。

4.8.1 RecordDataのメンバ変数

private

int	m_CurrentColumnNum	インスタンスを構成する実際のカラム数
TCHAR	m_TableName[TABLE_NAME_SIZE]	テーブル名
ColumnData*	m_ColumnData[MAX_COLUMN_NUMBER]	カラムを格納する配列
RecordData*	m_NextRecord	次のRecordDataへのポインタ NULL:次のRecordDataは存在しません

4.8.2 RecordDataの関数

(1) RecordData()

デフォルトコンストラクタ。

(2) RecordData(TCHAR TableName[TABLE_NAME_SIZE], ColumnData** ColDat, int NumOfColDat)

コンストラクタ。指定したカラムを含むRecordDataインスタンスを生成します。

TableName	テーブル名
ColDat	追加する1つ以上のカラムを含むColumnData[]配列
NumOfColDat	ColDatの配列の数

(3) virtual ~RecordData()

デストラクタ。

(4) void AddColumn(ColumnData* ColDat)

カラムを追加します。

ColDat	追加するカラム

(5) void DeleteColumn()

最後に追加したカラムを削除します。

(6) ColumnData* GetColumn(int ColIndex)

ColIndex番目に追加したカラムを取得します。

ColIndex	登録されているカラムのインデックス値
戻り値	指定したインデックス値で特定したカラムを返す。カラムが特定できない場合，NULLを返す。

(7) ColumnData* GetColumn(TCHAR* ColName)

指定した名称のカラムを取得します。

ColName)	登録されているカラムの名称
戻り値	指定した名称で特定したカラムを返す。カラムが特定できない場合，NULLを返す。

(8) int GetColumnCount()

RecordDataに設定されているカラムの数を返します。

(9) void SetTableName(TCHAR* TableName)

テーブル名を指定します。TableNameが16文字(NULL含む)を超えた場合，17文字目以降の文字は切り取られます。

TableName	テーブル名

(10) TCHAR* GetTableName()

テーブル名を取得します。

戻り値	テーブル名

(11) void SetNextRecord(RecordData* RecDat)

次のレコードを設定します。

RecDat	次のレコード

(12) RecordData* GetNextRecord()

次のレコードを取得します。

戻り値	次のレコード NULL:レコードは存在しません

4.9 ColumnData

StkDataのデータのレコード内カラムにアクセスするためのクラス（データモデル）です。

4.9.1 ColumnDataのメンバ変数

protected

TCHAR	m_ColumnName[COLUMN_NAME_SIZE]	カラム名
int	m_ColumnType	カラム種別

4.9.2 ColumnDataの関数

(1) ColumnData()
デフォルトコンストラクタ。

(2) virtual ~ColumnData()

デストラクタ。

(3) void SetColumnName(TCHAR* ColumnName)

カラム名を指定します。ColumnNameが16文字(NULL含む)を超えた場合，17文字目以降の文字は切り取られます。

ColumnName	カラム名

(4) TCHAR* GetColumnName()

カラム名を取得します。

戻り値	カラム名

(5) int GetColumnType()

カラムの種別を取得します。

戻り値	COLUMN_TYPE_INT:整数型 COLUMN_TYPE_STR:文字列型 COLUMN_TYPE_WSTR:ワイド文字列型 COLUMN_TYPE_BIN:バイナリ型 COLUMN_TYPE_FLOAT:単精度浮動小数点数型

4.10 ColumnDataInt

StkDataのレコード内整数型カラムにアクセスするためのクラス（データモデル）です。ColumnDataクラスの派生クラスとして定義されます。

4.10.1 ColumnDataIntのメンバ変数

private

int	m_Value	整数型カラムの値

4.10.2 ColumnDataIntの関数

(1) ColumnDataInt(TCHAR* ColumnName, int Val)

コンストラクタ。

ColumnName	カラム名
Val	整数型カラムの値

(2) virtual ~ColumnDataInt()

デストラクタ。

(3) int GetValue()

整数型カラムのデータを取得する。

戻り値	整数型カラムの値

4.11 ColumnDataStr

StkDataのレコード内文字列型カラムにアクセスするためのクラス（データモデル）です。
ColumnDataクラスの派生クラスとして定義されます。

4.11.1 ColumnDataStrのメンバ変数

private

char	m_Value[256]	文字列型カラムのデータ

4.11.2 ColumnDataStrの関数

(1) ColumnDataStr(TCHAR* ColumnName, char* Val)

コンストラクタ。Valの値256文字分をm_Valueにコピーする。

ColumnName	カラム名
Val	文字列型カラムデータへのポインタ

(2) virtual ~ColumnDataStr()

デストラクタ。

(3) char* GetValue()

文字列型カラムのデータを取得する。

戻り値	文字列型カラムデータへのポインタ

4.12 ColumnDataWStr

StkDataのレコード内ワイド文字列型カラムにアクセスするためのクラス（データモデル）です。
ColumnDataクラスの派生クラスとして定義されます。

4.12.1 ColumnDataWStrのメンバ変数

private

TCHAR	m_Value[256]	ワイド文字列型カラムのデータ

4.12.2 ColumnDataWStrの関数

(1) ColumnDataWStr(TCHAR* ColumnName, TCHAR* Val)

コンストラクタ。Valの値256文字分をm_Valueにコピーする。

ColumnName	カラム名
Val	ワイド文字列型カラムデータへのポインタ

(2) virtual ~ColumnDataWStr()

デストラクタ。

(3) TCHAR* GetValue()

ワイド文字列型カラムのデータを取得する。

戻り値	ワイド文字列型カラムデータへのポインタ

4.13 ColumnDataBin

StkDataのレコード内バイナリ型カラムにアクセスするためのクラス（データモデル）です。
ColumnDataクラスの派生クラスとして定義されます。

4.13.1 ColumnDataBinのメンバ変数

private

BYTE*	m_Value	バイナリ型カラムのデータ

4.13.2 ColumnDataBinの関数

(1) ColumnDataBin(TCHAR* ColumnName, BYTE* Val, int Length)

コンストラクタ。Lengthバイトのサイズをもつバイナリ型データm_Valueを生成し，Valの値をコピーする。

ColumnName	カラム名
Val	バイナリ型カラムデータへのポインタ
Length	コピーするValのデータのサイズ（バイト数）。実存するカラムのサイズと一致させる必要がある。

(2) virtual ~ColumnDataBin()

デストラクタ。m_Valueの領域を開放する。

(3) BYTE* GetValue()

バイナリ型カラムのデータを取得する。

戻り値	バイナリ型カラムデータへのポインタ

4.10 ColumnDataFloat

StkDataのレコード内単精度浮動小数点数型カラムにアクセスするためのクラス（データモデル）です。ColumnDataクラスの派生クラスとして定義されます。

4.10.1 ColumnDataFloatのメンバ変数

private

float	m_Value	単精度浮動小数点数型カラムの値

4.10.2 ColumnDataFloatの関数

(1) ColumnDataFloat(TCHAR* ColumnName, float Val)

コンストラクタ。

ColumnName	カラム名
Val	単精度浮動小数点数型カラムの値

(2) virtual ~ColumnDataFloat()

デストラクタ。

(3) float GetValue()

単精度浮動小数点数型カラムのデータを取得する。

戻り値	単精度浮動小数点数型カラムの値

5. インターフェース

5.1 インターフェース一覧

■リポジトリ情報取得用

(1) GetTableCount
(2) GetTableName
(3) GetTableSize
(4) GetTableVersion
(5) GetColumnCount
(6) GetColumnName
(7) GetColumnSize
(8) GetColumnType
(9) GetNumOfRecords
(10) GetMaxNumOfRecords

■テーブル定義用

(11) CreateTable
(12) DeleteTable

■テーブルロック用

(13) LockTable
(14) LockAllTable
(15) UnlockTable
(16) UnlockAllTable

■データアクセス用

(17) InsertRecord
(18) UpdateRecord
(19) DeleteRecord(**全レコード削除**)
(20) DeleteRecord(**検索条件付削除**)
(21) GetRecord(**全レコード取得**)
(22) GetRecord(**検索条件付取得**)
(23) ClearRecordData

■レコードソート用

(24) AzSortRecord
(25) ZaSortRecord

■ファイルアクセス用

(26) SaveData
(27) LoadData
(28) AutoSave

5.2 インターフェース詳細

(1) int GetTableCount()

この関数は存在するテーブルの数を返却します。

戻り値	存在するテーブルの数

(2) int GetTableName(TCHAR TableNames[MAX_TABLE_NUMBER][TABLE_NAME_SIZE])

この関数は存在するの全てのテーブルの名称を返却します。

TableNames	**テーブルの名称(最大16個)を返却する配列**
戻り値	**テーブルの数**

(3) int GetTableSize(TCHAR* TableName)

この関数は指定したテーブルのサイズ（バイト数）を返却します。

TableName	**サイズを取得する対象のテーブルの名称**
戻り値	**テーブルのサイズ（バイト数）**

(4) int GetTableVersion(TCHAR* TableName)

この関数は指定したテーブルのバージョンを返却します。
テーブルのバージョンとは，対象のテーブルが何回更新されたかを示す数値で，InsertRecord, UpdateRecord, DeleteRecord **でテーブル内のレコードを変更する度に1加算されます。**
StkData**利用プログラムで，対象のテーブルの変更有無を調べる場合，この関数を呼び出します。バージョンの初期値は**0**で，**CreateTable**およびLoadData**を呼び出した後，バージョンには初期値が設定されます。バージョンの取り得る範囲は**0**から**2147483647**の間で，バージョンが**2147483647**時点でテーブルが変更された場合，バージョンは**0**に戻り，その後変更の度に**1**加算されます。これは，テーブルの変更有無を調べたい場合，テーブルが前回から変更されていないのか**2147483648**回変更 されたのかを判別することができないことを意味します。**

TableName	**バージョン取得先のテーブルの名称**
戻り値	-1：**指定したテーブルは存在しない** 0～2147483647：**指定したテーブルのバージョン**

(5) int GetColumnCount(TCHAR* TableName)

この関数は指定したテーブルに存在するカラムの数を返却します。

TableName	**対象となるテーブルの名称**
戻り値	-1：**失敗** -1**以外**：**カラムの数**

(6) int GetColumnName(TCHAR* TableName, TCHAR ColumnNames[MAX_COLUMN_NUMBER][COLUMN_NAME_SIZE])

この関数は指定したカラムの名称を返却します。

TableName	**テーブルの名称**
ColumnNames	TableName**で指定したテーブルに存在するカラムの名称(最大**32**個)を返却する配列**
戻り値	-1：**失敗** -1**以外**：**返却されるカラム名の数**

(7) int GetColumnSize(TCHAR* TableName, TCHAR* ColumnName)

この関数は指定したカラムのサイズ（バイト数）を返却します。

TableName	**テーブルの名称**
ColumnName	TableName**で指定したテーブルに存在するカラムの名称**
戻り値	-1：**失敗** -1**以外**：**指定したカラムのサイズ（バイト数）**

(8) int GetColumnType(TCHAR* TableName, TCHAR* ColumnName)
この関数は指定したカラムの種別を返却します。

TableName	**テーブルの名称**
ColumnName	TableName**で指定したテーブルに存在するカラムの名称**
戻り値	-1：**失敗** COLUMN_TYPE_INT:**整数型** COLUMN_TYPE_FLOAT:**単精度浮動小数点数型** COLUMN_TYPE_STR:**文字列型** COLUMN_TYPE_WSTR:**ワイド文字列型** COLUMN_TYPE_BIN:**バイナリ型**

(9) int GetNumOfRecords(TCHAR* TableName)

この関数は，指定されたテーブルに登録されているレコードの件数を返却します。

TableName	**レコード件数取得対象テーブルの名称**
戻り値	-1：**失敗** -1**以外**：**指定されたテーブルに登録されているレコードの件数**

(10) int GetMaxNumOfRecords(TCHAR* TableName)

この関数は，指定されたテーブルに登録できる最大のレコード件数を返却します。

TableName	**レコード件数取得対象テーブルの名称**
戻り値	-1：**失敗** -1**以外**：**指定されたテーブルに登録可能な最大のレコード件数**

(11) int CreateTable(TableDef* TabDef)

この関数はテーブル生成します。内部処理として，仮想メモリを割当てそれを初期化します。

TabDef	TableDef**インスタンス**
戻り値	0：**成功** -1：**メモリ割当エラー** -2：**パラメータエラー** (**カラムが存在しない**; **最大レコード数が不正**; **テーブル名が不正**) -3：**コミットエラー** -4：**登録可能最大テーブル数を超えた**

(12) int DeleteTable(TCHAR* TableName)

この関数は既存のテーブルを削除します。内部処理として，仮想メモリの開放を行います。

TableName	**削除するテーブルの名称**
戻り値	0：**成功** -1：**メモリ開放エラー** -2：**パラメータエラー** -3：**デコミットエラー**

(13) int LockTable(TCHAR* TableName, int LockType)

データアクセス用関数(InsertRecord, DeleteRecord, UpdateRecord, AzSortRecord, ZaSortRecord **または** GetRecord)**は マルチスレッド呼び出しをサポートしています。異なるスレッドからの同一テーブルへの同時アクセスを防ぐため，** StkData**はロッキングメカニズムを採用しています。**
2**つのスレッド（スレッド**A**とスレッド**B**）が存在しており，スレッド**A**がロック中の場合，スレッド**B**はスレッド**A**がアンロック するまでロックを掛けることができません。この場合，スレッド**A**がアンロッ**

クするまでスレッドBはロック待ちとなります。StkData利用プログラムは，データアクセス用関数（InsertRecord, DeleteRecord, UpdateRecord, GetRecord, AzSortRecord, ZaSortRecord, LoadData, SaveData）を 呼び出す前にLockTable()を呼び出す必要があります。また，StkData利用プログラムはデータアクセス用関数を使用し 終わったあと，UnlockTable()を呼び出す必要があります。
StkData利用プログラムは2つのタイプのロックを掛けることができます。1つは共有ロック，もう1つは排他ロックです。
共有ロックはGetRecord()などの情報取得系関数のみを使用する場合に掛けられます。共有ロックでは異なる2つのスレッドが 同時に同じテーブルに情報取得用アクセスを行うことを認めます。
排他ロックはInsertRecord()，UpdateRecord()，DeleteRecord()などの情報更新系関数を使用するときに掛けられます。排他ロックでは異なる2つのスレッドが同時に同じテーブルにアクセスを行うことを認めません。
StkData利用プログラムは下記の共有ロックと排他ロックの振る舞いについて考慮する必要があります。

1-共有ロック, 2-共有ロック
→ 成功
1-共有ロック, 2-排他ロック
→ #2 は #1のアンロックを待ち続ける
1-排他ロック, 2-共有ロック
→ #2 は #1のアンロックを待ち続ける
1-排他ロック, 2-排他ロック
→ #2 は #1のアンロックを待ち続ける

デッドロックを防ぐために，StkData利用プログラムはロックを掛ける順番を考慮する必要がある。
例: 下記のスレッドを同時に実行するとデッドロックが発生するおそれがあります。

Thread (A) : [Table-1 排他ロック] ---> [Table-2 排他ロック]
Thread (B) : [Table-2 排他ロック] ---> [Table-1 排他ロック]

TableName	ロックを掛けるテーブルの名称
LockType	ロック種別 LOCK_SHARE : 共有ロック LOCK_EXCLUSIVE : 排他ロック
戻り値	0 : 成功 -1 : ロックエラー（指定されたテーブルは存在しない, 指定されたロック種別が不正）

(14) int LockAllTable(int LockType)

この関数は全てのテーブルにロックを掛けます。

LockType	ロック種別 LOCK_SHARE : 共有ロック LOCK_EXCLUSIVE : 排他ロック
戻り値	常に0

(15) int UnlockTable(TCHAR* TableName)

この関数は指定されたテーブルをアンロックします。

TableName	アンロックするテーブルの名称
戻り値	0 : 成功, -1 : アンロックエラー

(16) int UnlockAllTable()

この関数は全てのテーブルをアンロックします。

戻り値	常に0

(17) int InsertRecord(RecordData* RecDat)

この関数は指定したテーブルにレコードを追加します。複数のレコードを追加するには 連結されたRecordDataインスタンスを引数に指定します。

RecDat	1. 連結されたRecordDataについて 複数のレコードを指定するときは，RecordDataのSetNextRecord関数で連結したRecordData インスタンスの先頭インスタンスを指定します。InsertRecord関数はそれぞれ別のテーブル を操作対象とする複数の連結されたRecordDataインスタンスが指定されることを許します。 2. RecordData内ColumnDataStrインスタンスについて - ColumnDataStrのデータの取り得る範囲は0x00および0x20-0x7e - 上記範囲外の文字が与えられた場合，空白文字' 'に置き換えられる - 空文字を指定することができる - NULLは指定することができない - ColumnDataStrで指定する文字列の長さはカラムの最大文字列長-1以下でなければならない。カラムの最大文字列長以上の長さの文字列が与えられた場合，最大文字列長-1を超える文字は切り取られる。 3. RecordData内ColumnDataWStrインスタンスについて - 空文字を指定することができる - NULLは指定することができない - ColumnDataWStrで指定する文字列の長さはカラムの最大文字列長-1以下でなければならない。カラムの最大文字列長以上の長さの文字列が与えられた場合，最大文字列長-1を超える文字は切り取られる。 4. RecordData内ColumnDataBinインスタンスについて - NULLは指定することができない - ColumnDataBinで指定するデータのサイズは操作対象のカラムのサイズと一致させなければならない。サイズが不一致のデータを指定した場合，データは不正に処理される。 5. RecordData内ColumnDataIntインスタンスについて - NULLは指定することができない - ColumnDataIntのデータの取り得る範囲は -2147483648 < 2147483647 6. RecordData内ColumnDataFloatインスタンスについて - NULLは指定することができない - ColumnDataFloatのデータの取り得る範囲は32bitの単精度浮動小数点数の範囲 7. ColumnDataについて - ColumnDataIntインスタンス，ColumnDataFloatインスタンス，ColumnDataStrインスタンス，ColumnDataWStr，ColumnDataBin インスタンスとしてNULLを指定することはできない 8. 指定するカラムについて - RecordDataインスタンスには，操作対象であるテーブルに含まれる全てのカラムの データを設定しなければならない。一部でも不足した場合，処理は失敗する。
戻り値	0：成功 -1：失敗

(18) int UpdateRecord(RecordData* SchRec, RecordData* UpdRec)

この関数は1つ目のRecordDataで検索された既存のレコードを2つ目のRecordDataに 置き換えます。

SchRec	検索用レコードを指定します。 複数のレコードを検索するために，SetNextRecord()関数を用いて連結したRecordDataインスタンス を指定することができます。UpdateRecord関数は，連結された複数のRecordDataインスタンスの 操作対象テーブルが異なることを許していません。
UpdRec	更新用レコードを指定します。 1. レコードの更新について SchRecで検索されたレコードをUpdRecで置き換えます。UpdRecは連結されていない単独のRecordDataインスタンスである必要があります。UpdRecで指定したRecordDataインスタンスが 複数の連結されたRecordDataインスタンスの場合，1番最初のインスタンスのデータで置き換え ます。2番目 以降のインスタンスは無視されます。 UpdRecで指定するRecordDataインスタンスには更新するカラムのみ指定します。指定されない カラムについては更新されません。 2. RecordData内ColumnDataStrインスタンスについて - ColumnDataStrのデータの取り得る範囲は0x00および0x20-0x7e - 上記範囲外の文字が与えられた場合，空白文字' 'に置き換えられる - 空文字を指定することができる - NULLは指定することができない - ColumnDataStrで指定する文字列の長さはカラムの最大文字列長-1以下でなければならない。カラムの最大文字列長以上の長さの文字列が与えられた場合，最大文字列長-1を超える文字は切り取られる。 3. RecordData内ColumnDataWStrインスタンスについて - 空文字を指定することができる - NULLは指定することができない - ColumnDataWStrで指定する文字列の長さはカラムの最大文字列長-1以下でなければならない。カラムの最大文字列長以上の長さの文字列が与えられた場合，最大文字列長-1を超える文字は切り取られる。 4. RecordData内ColumnDataBinインスタンスについて - NULLは指定することができない - ColumnDataBinで指定するデータのサイズは操作対象のカラムのサイズと一致させなければならない。サイズが不一致のデータを指定した場合，データは不正に処理される。 5. RecordData内ColumnDataIntインスタンスについて - NULLは指定することができない - ColumnDataIntのデータの取り得る範囲は -2147483648 < 2147483647 6. RecordData内ColumnDataFloatインスタンスについて - NULLは指定することができない - ColumnDataFloatのデータの取り得る範囲は32bitの単精度浮動小数点数の範囲 7. ColumnDataについて - ColumnDataIntインスタンス，ColumnDataFloatインスタンス，ColumnDataStrインスタンス，ColumnDataWStrインスタンス，ColumnDataBinインスタンスとしてNULLを指定することはできない
戻り値	0：成功 -1：失敗

(19) int DeleteRecord(TCHAR* TableName)

この関数は，指定されたテーブルに登録されている全てのレコードを削除します。

TableName	全レコードを削除する対象となるテーブルの名称
戻り値	0：成功 -1：失敗

(20) int DeleteRecord(RecordData* DelRec)

この関数は指定したRecordDataインスタンスで検索されたレコードを削除します。
レコードを削除した場所には，対象テーブルの最後のレコードが移動します。この結果DeleteRecord実行前と実行後でテーブル内のレコードの並び順序が異なる場合があります。（レコード削除性能を向上するためにこのような仕様となっています。）

DelRec	検索用レコードを指定します。 DelRecで検索されたレコードは削除されます。DelRecのRecordDataインスタンスに レコードの全てのカラムを指定する必要はありません。何もカラムを指定しない場合， 指定したテーブルのすべてのレコードが削除されます。DeleteRecord関数は，DelRecに連結された複数のRecordDataインスタンスが指定されることを許します。また，上記複数のインスタンスがそれぞれ異なるテーブルを操作対象とすることを許します。
戻り値	0：成功 -1：失敗

(21) RecordData* GetRecord(TCHAR* TableName)

この関数は，指定されたテーブルに登録されている全てのレコードを返却します。

TableName	レコード取得対象テーブルの名 称
戻り値	指定されたテーブルが複数のレコードを登録している場合，連結された複数のRecordData インスタンスが返却されます。指定したテーブルが存在しない場合，または 指定したテーブルに何もレコードが存在しない場合，NULLが返却されます。

(22) RecordData* GetRecord(RecordData* SchRec)

この関数はSchRecで検索されたレコードを返却します。

SchRec	取得するレコードの検索条件を指定します。連結された複数のRecordDataインスタンス を指定することができます。連結された複数のRecordDataインスタンスが指定する場合， GetRecord関数は，各々のインスタンスが異なるテーブルを対象とすることを許します。
戻り値	連結された複数のRecordDataインスタンスが返却されます。指定したテーブルが存在しない場合，または 指定したテーブルに何もレコードが存在しない場合，NULLが返却されます。 複数のRecordDataインスタンスが連結してSchRecに指定される場合で，その複数のRecordDataの検索条件に 重複して該当するレコードが存在する場合，同じデータを持つ複数のレコードが重複して返却されます。

(23) void ClearRecordData(RecordData* RecDat)

GetRecord関数はRecordDataインスタンスを生成し，それを呼び出し元に返します。
RecordDataインスタンスはColumnDataインスタンスを含んでおり，また，RecordDataインスタンス 自身も他のRecordDataインスタンスと連結している場合があります。GetRecord関数で取得したRecordDataインスタンスの数が膨大になる場合，使い終わった RecordDataインスタンスは削除してメモリを開放する必要があります。
ClearResponseData関数は，指定されたRecordDataインスタンスおよびそれに連結するインスタンス について，中に梱包されているColumnDataインスタンスを含めて削除します。
ClearRecordDataはGetRecord関数を用いて取得したインスタンスを削除するだけでなく，InsertRecord関数やUpdateRecord関数用にユーザーが作成したRecordDataインスタンスを削除 することができます（InsertRecord関数等でユーザーが作成／指定したRecordDataインスタンスは，通常関数呼び出し後，別の用途で使用される機会は少ない）。

RecDat	削除したいRecordDataインスタンス。RecordDataインスタンスが連結して複数のデータを構成している場合は，先頭の RecordDataインスタンスを指定します。

(24) int AzSortRecord(TCHAR* TableName, TCHAR* ColumnName)

指定したテーブル内の全レコードを，指定したカラムで昇順となるように並び替えます。
カラム種別がCOLUMN_TYPE_INT : 数値の小さい順に並び替えます。
カラム種別がCOLUMN_TYPE_FLOAT : 数値の小さい順に並び替えます。
カラム種別がCOLUMN_TYPE_STR : 文字コードの小さい順に並び替えます。
カラム種別がCOLUMN_TYPE_WSTR : 文字コードの小さい順に並び替えます。
カラム種別がCOLUMN_TYPE_BIN : コードの小さい順に並び替えます。
AzSortRecord関数を呼び出す前に対象のテーブルに排他ロックを掛ける必要があります。

TableName	ソート対象のテーブルの名称
ColumnName	ソート対象のカラムの名称
戻り値	0：成功 -1：失敗

(25) int ZaSortRecord(TCHAR* TableName, TCHAR* ColumnName)

指定したテーブル内の全レコードを，指定したカラムで降順となるように並び替えます。
カラム種別がCOLUMN_TYPE_INT : 数値の大きい順に並び替えます。
カラム種別がCOLUMN_TYPE_FLOAT : 数値の大きい順に並び替えます。
カラム種別がCOLUMN_TYPE_STR : 文字コードの大きい順に並び替えます。
カラム種別がCOLUMN_TYPE_WSTR : 文字コードの大きい順に並び替えます。
カラム種別がCOLUMN_TYPE_BIN : コードの大きい順に並び替えます。
ZaSortRecord関数を呼び出す前に対象のテーブルに排他ロックを掛ける必要があります。

TableName	ソート対象のテーブルの名称
ColumnName	ソート対象のカラムの名称
戻り値	0：成功 -1：失敗

(26) int SaveData(TCHAR* FileName)

StkDataにより操作された情報は全て仮想メモリ中に格納されます。したがって，仮想メモリ中のデータをファイルにセーブすることなくStkData利用プログラムを 終了した場合，全てのデータが失われることになります。
SaveData関数は仮想メモリ中に存在する全てのテーブル，レコードを 指定したファイルに書き出します。SaveData関数を呼び出す前に全てのテーブルに共有ロックを掛ける必要があります。

FileName	ファイル名（ファイルパス）
戻り値	0：成功 -1：失敗

(27) int LoadData(TCHAR* FileName)

LoadData関数は指定したファイルからテーブルおよびレコードをロードし，仮想メモリに格納します。LoadData関数は既存の全テーブルを削除後，読込んだファイル内に定義されている情報をもとに新たにテーブルを作成します。このため，LoadData関数は処理のはじめに既存の全テーブルに排他ロックが掛かって いるかチェックを行い，1つでも排他ロックが掛かっていないテーブルがあった場合， 処理を中断し，-1を返却します。ただし，LoadData関数の呼び出し時点でテーブルが1つも 存在しない場合，ロックの有無に関係なしにLoadData関数を実行することができます。

FileName	ファイル名（ファイルパス）
戻り値	0：成功 -1：失敗

(28) int AutoSave(TCHAR* FileName, int Sec, BOOL Flag)

この関数は仮想メモリ中に存在する全てのテーブル，レコードを指定したファイルに自動的に

書き出すかどうか を制御します。AutoSaveは指定された間隔で無条件にファイルに書き出すのではなく，内部的にGetTableVersionを呼び 出し，前回ファイル書き出し時から少なくとも1つのテーブルに変更が加えられていればファイルに書き出しします。AutoSaveは，CreateTable, DeleteTableによりテーブル数が変化したときも，それを検知し自動的にファイルに書き出します。前回からどのテーブルにも変更が無い場合，指定された間隔が過ぎてもファイルに書き出ししません。

FileName	ファイル名（ファイルパス）
Sec	自動的にファイルに書き出す間隔（秒数）を指定します。ここで指定した間隔（秒数）で はあくまでも目安であり，正確にこの間隔でファイルに書き出すことを保証するものではありません。 30～86400秒（30秒～24時間）の間で値を指定できます。
Flag	TRUE：自動的なファイル書き出しを許可する FALSE：自動的なファイル書き出しを禁止する 既にTRUEを指定して自動的なファイル書き出しを実行しているときにさらにTRUEを指定した場合，または既にFALSE を指定して自動的なファイル書き出しを停止しているときにさらにFALSEを指定した場合，AutoSaveは-1を返します。
戻り値	0：成功 -1：失敗

6. データおよびファイル

6.1 ファイル仕様

SaveData関数およびAutoSave関数により，StkDataはメモリ内のデータをファイルに出力します。
出力されるファイルは以下のような構造になっています。

情報のサイズ	保存される情報
16バイト	文字列"StkData_0100"。そのあとに続く4バイトは不定値
4バイト	テーブルの数(N)
32バイト	n(n=0, 1, 2, ... , 15)番目のテーブルのテーブル名
4バイト	n(n=0, 1, 2, ... , 15)番目のテーブルの最大レコード数
4バイト	n(n=0, 1, 2, ... , 15)番目のテーブルのカラム数(M)
32バイト	n(n=0, 1, 2, ... , 15)番目のテーブルのm番目(m=0, 1, 2, ... , M)のカラムの名称
4バイト	n(n=0, 1, 2, ... , 15)番目のテーブルのm番目(m=0, 1, 2, ... , M)のカラムの種別
4バイト	n(n=0, 1, 2, ... , 15)番目のテーブルのm番目(m=0, 1, 2, ... , M)のカラムがCOLUMN_TYPE_STR, COLUMN_TYPE_WSTR, COLUMN_TYPE_BINの場合，カラムのサイズ（バイト数）。 COLUMN_TYPE_INT，COLUMN_TYPE_FLOATの場合この領域は存在しません。
4バイト	n(n=0, 1, 2, ... , 15)番目のテーブルのサイズ（バイト数）
4バイト	n(n=0, 1, 2, ... , 15)番目のテーブルの1レコード分のサイズ（バイト数）
4バイト	n(n=0, 1, 2, ... , 15)番目のテーブルのレコード数
レコード数×1レコード分のサイズ	n(n=0, 1, 2, ... , 15)番目のテーブルの全レコードのデータ