この章では、***HiBase*** の短い紹介（***HiBase*** の製品に含まれている「インタフェースプログラム」の概要とサポートサービス）及び、「***HiBase*** のアプリケーションプログラムの開発環境」の構築方法について説明しています。

HiBase のマスターディスク構成についての詳細は、「スタートアップマニュアル」を参照してください。

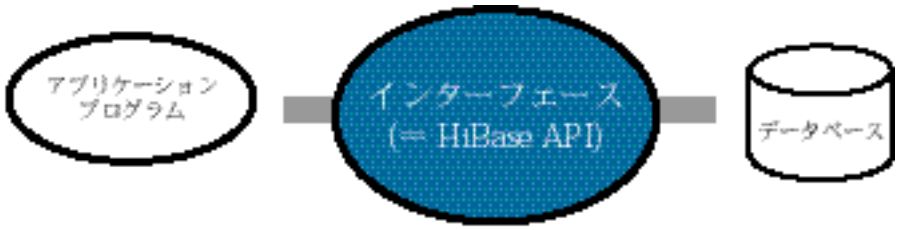
この章で解説する主要な項目は、次の通りです。

１－１．HiBase 開発環境の基礎知識

HiBase API

HiBase はデータベース管理システムです。

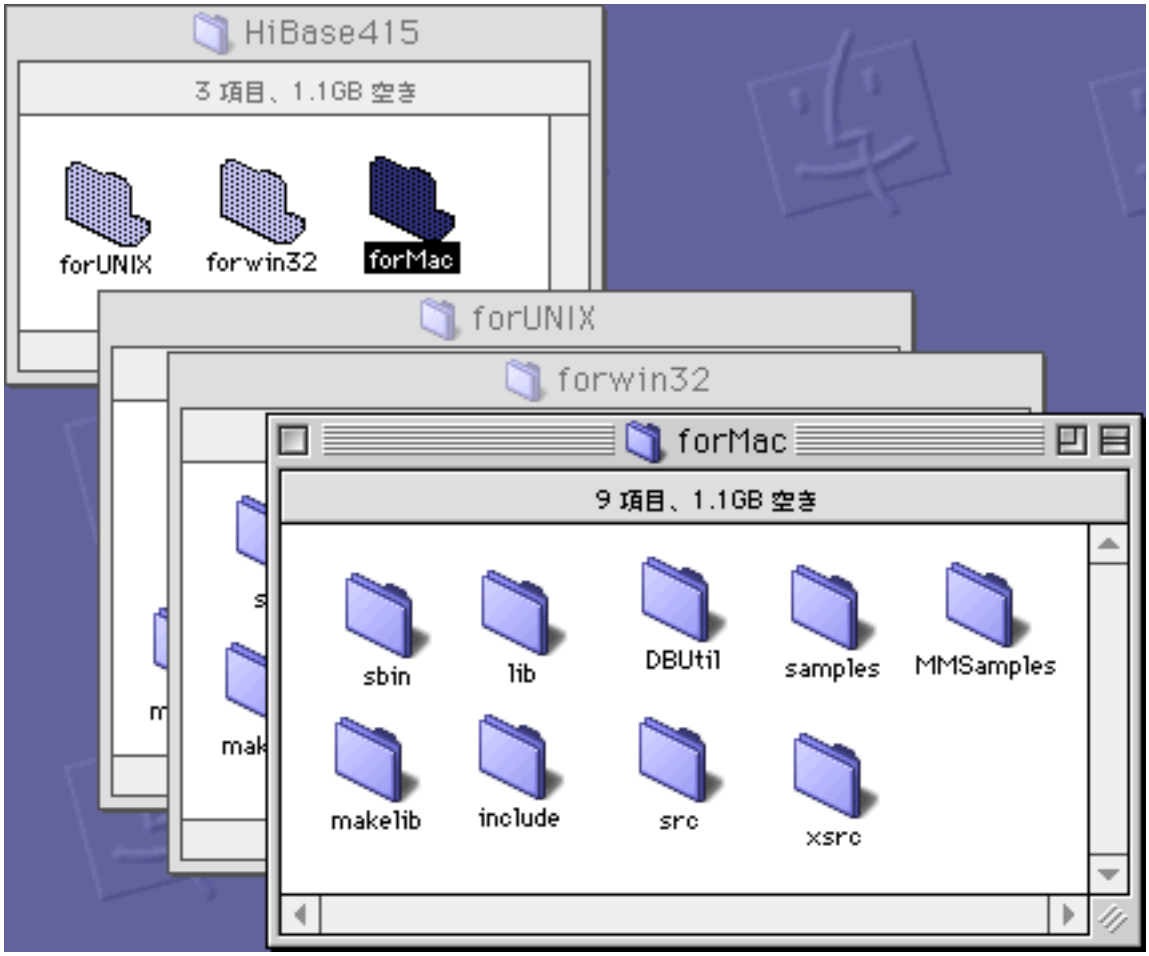
そしてHiBase開発環境は、アプリケーションプログラムへが、HiBaseの管理するデータベースにアクセスするためのインターフェースを提供するのが目的です。この「HiBaseの管理するデータベースにアクセスするためのインターフェース」を「HiBase API」といいます。



HiBaseAPI は、「マルチプラットフォーム」、「シングルクライアントからクラインアントサーバまで」、「シングルユーザ/シングルスレッドからマルチユーザ / マルチスレッドまで」、を単一の論理的なインターフェースで実現しています。

マルチプラットフォーム

HiBase は、現在、UNIX、Win32（Windows 95 / Windows 98 / Windows NT）、Macintosh のそれぞれの「OS およびハードウェア」に対応しています。これらの「OS およびハードウェア」に対応した開発環境は、ホロンの提供するメディア（弊社のホームページ、もしくは、CD-ROM）で下図のように提供されています。



さらに、HiBaseは移植性に優れているため、ホロン株式会社は、特殊な「OS およびハードウェア」への移植サービスをおこなっています。

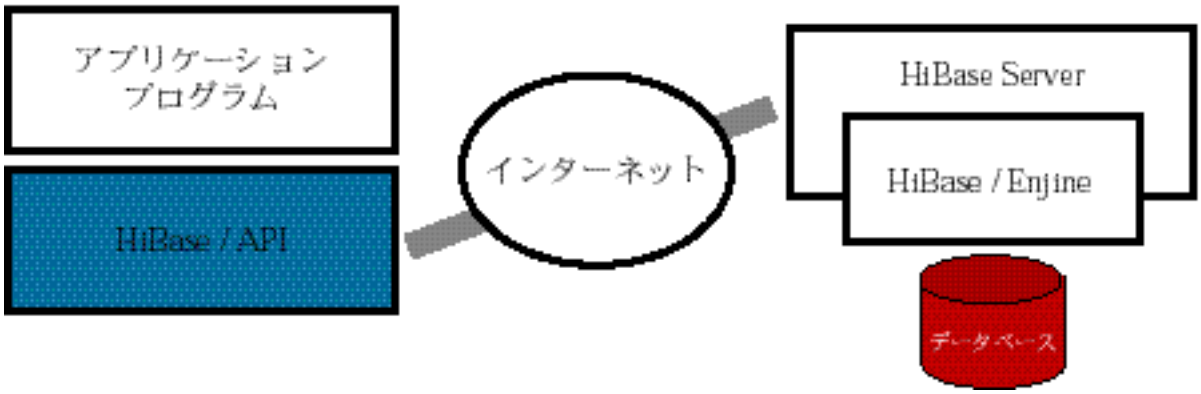
シングルクライアントから

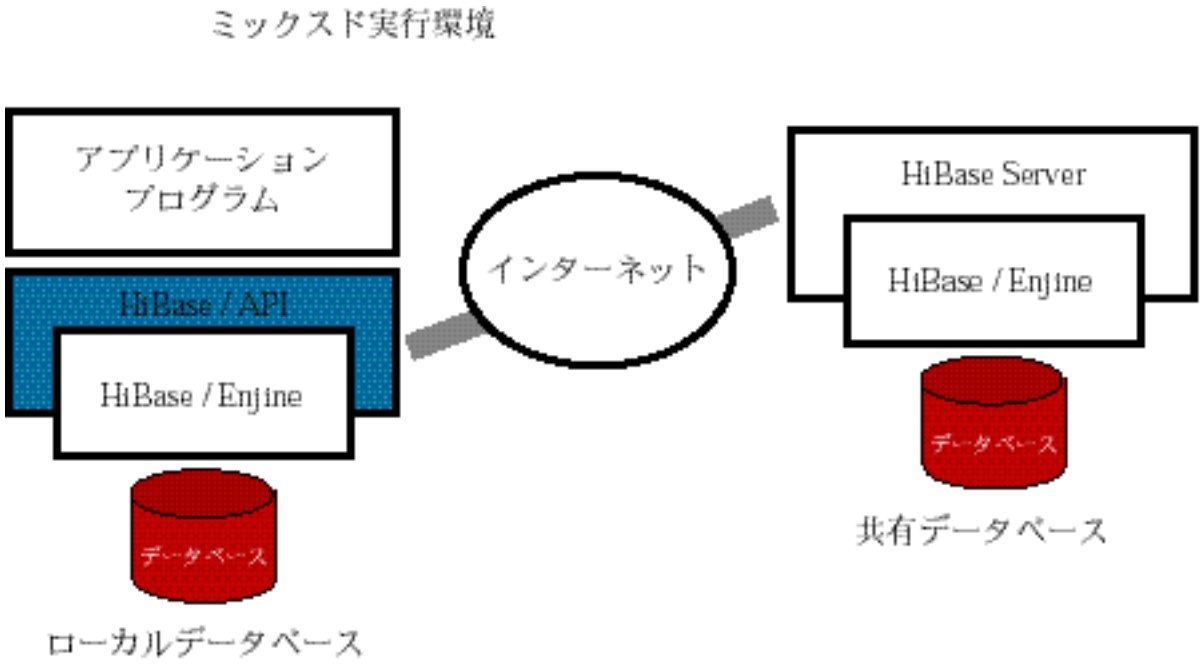
クラインアントサーバまで

HiBaseは、シングルクライアント実行環境、および、クライアントサーバー実行環境、さらに両者を混合した実行環境（ミックスド実行環境）のいずれも対応できます。

シングルクライアント実行環境

クライアントサーバー 実行環境





例えば、HiBase サーバーを linux 環境で運用して共有のデータベースを管理し、クライアントである MacintoshやWindowsからは、共有のデータベースをクライアントサーバ実行環境でアクセスし、さらにローカルなデータベースをシングルクライアント実行環境で運用するというような環境が構築できます。

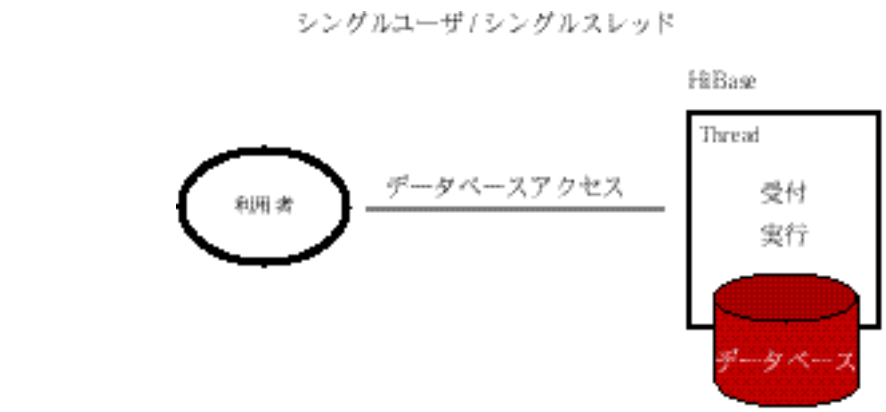
HiBase の「クライアントサーバー実行環境」は、「TCP / IP プロトコル」および「SOCKET（BSD SOCKET、WinSock）」を利用して実現しています。

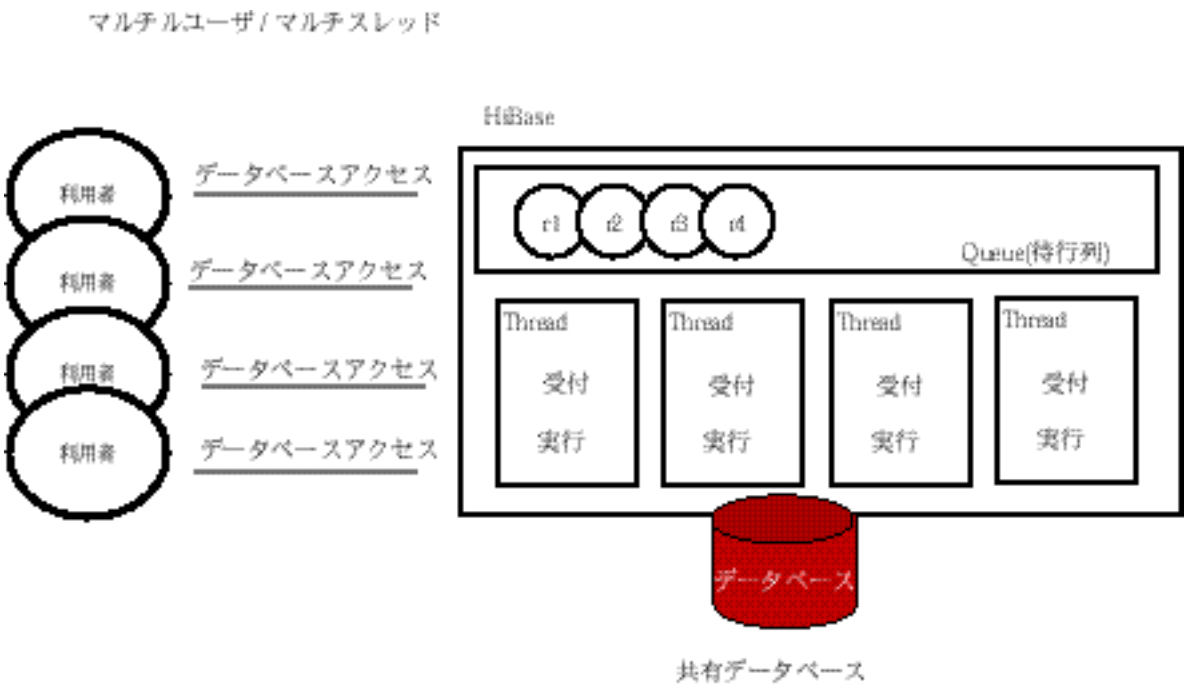
シングルユーザ / シングルスレッドから
マルチユーザ / マルチスレッドまで

HiBase の実行形態としては、シングルユーザ/ シングルスレッドからマルチユーザ / マルチスレッドの選択ができます。

シングルユーザ/シングルスレッドは、データベースをアクセスする利用者が一人だけの場合に有効な実行形態です。普通、シングルクライアント実行環境での実行形態は、シングルユーザ /シングルスレッドとなります。一方、マルチユーザ / マルチスレッドは、同時に複数の利用者が同じデータベースにアクセスする場合に有効です。そのためHiBase Serverはマルチユーザ / マルチスレッドの形態で運用されます。また、シングルクライアント実行環境で実行するアプリケーションでありながら、同時に複数の利用者からの利用を受け付ける特殊なサーバーを開発する場合は、HiBaseの実行形態として「マルチユーザ / マルチスレッド」を選択することができます。

HiBase の「マルチユーザ / マルチスレッド」は、「PSIX THREAD：Pthread」、「Windows Thread」を使って実現しています。

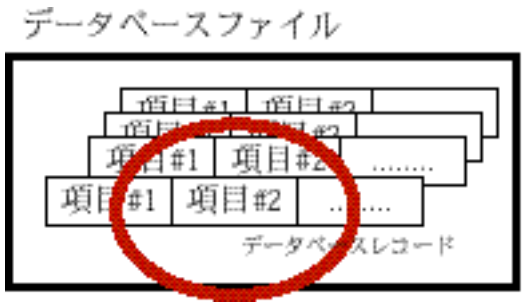




リレーショナルデータベースを維持しながら

マルチメディアデータベース

HiBaseデータベースを外から（アプリケーションプログラムから）眺めた場合、単純な表の集合と見えます。つまりユーザから、または、プログラム開発者からは、リレーショナルデータベースと見えます。しかし、その表を構成する項目のそれぞれに、単純な数字や文字列はもとより、絵（GIF、JPEG、PICT、BMP、...）、文書（Text、HTML、PDF、）、ファイル（通常ファイル、Macintoshのリソース付ファイル）、音、を格納し、利用することができます。さらに（最大127種類までの）ユーザ固有のデータ型を定義して追加する（例えば、「映画を格納する項目」を追加する）ことができます。このユーザ固有のデータ型の追加は、ユーザがVDD （バリュードライバー）のコードを開発することでおこないます



十進数字	**絵　（GIF、JPEG、PICT、BMP、PNG、....）**
バイナリ数字	**文章（Text、HTML、PDF、....）**
文字	**ファイル（DOS file、Mac file、...）**
文字列	**.....**
フラグ	
日付	**ユーザ定義#1 ~ #127**
時間	
.....	

その他

HiBase は、開発言語として「C++」を利用しています。そして、HiBase のデータベースを操作する全ての機能を、「クラス」とその「メンバ関数」で表現し、これらを「クラスライブラリ」の形にまとめています。

この「HiBase ＝クラスライブラリ」は、百を超える「クラス」と数千の「メンバ関数」が定義されている大規模なものですが、アプリケーションプログラムが直接利用する部分はある程度限定されていることから、HiBase は、アプリケーションプログラムからHiBase のデータベースを操作する時に利用する「クラス」とその「メンバ関数」を「HiBase API（HiBase インタフェースプログラム）」と定義しています。

１-１-１. ご利用前に

HiBase の権利は、すべてホロン株式会社に帰属します。

本製品パッケージに含まれている「使用許諾契約」に基づき、お使いください。また、ホロン株式会社の文書による承諾なしに、無断で使用することはできません。

【注意】

ソフトウェアをインストールする前に、「使用許諾契約書」を読み、その内容に同意する必要があります。

なお、HiBase で開発した独自のアプリケーションプログラムを第三者に配付／販売したり、HiBase に移植や改変を行うためには、別途「コピーライセンス」や「改変ライセンス」が必要です。これらの作業を行う場合は、「ライセンス許諾契約」を行ってください。

HiBase のライセンス契約についての詳細は、「オペレーション・マニュアル」巻末の「ライセンスのご案内」を参照してください。

サポートについて

HiBase は、ソフトウェア開発に於て、常にベストのサポートを提供するため、優秀なサポートチームを控えています。サポート窓口は、この章の最後にまとめてありますので、必要に応じ参照してください。

◇　既にソフトウェアを購入された方へ

「ユーザー登録カード」に必要事項を記入し、返送してください。

バージョンアップ、技術情報、各種セミナーなどのご案内を致します。

◇　質問をしたい方へ

操作中にトラブルが生じた場合、または、HiBase に関するテクニカルなサポートを受けたい方は、support@lists.hln.co.jp 宛てにe-mail をお願い致します。

またHiBaseのライセンス契約等、ノンテクニカルなご質問は、sales@lists.hln.co.jp 宛てにe-mail をお願い致します。

１-２．HBase 開発環境の構築

- HiBase アプリケーションプログラムの開発環境...

HiBase API を利用するためには、HiBase アプリケーションプログラムの開発環境を構築する必要があります。この章では、その必要条件について説明します。

◇　必要システム

アプリケーションプログラム開発に必要なシステム、及び、完成したHiBase のアプリケーションを実行するために必要なシステム

◇　HiBase APIの構成

HiBase の開発環境のセットアップ（プロジェクトの作成、サンプルプログラム実行環境の調整）

１-２-１．必要システム

HiBase のアプリケーションプログラム開発、及び、HiBase で開発したアプリケーションプログラムを実行するためには、以下のシステム条件が必要です。

【Mac OSのシステム条件】

項目	条件
ハードウェア	PowerPC（推奨）又は、 68030以上のチップを搭載した68K の Macintosh 50 MBのディスクスペース CD-ROMドライブ
ソフトウェア	System7.1以降、及び、 Metro Werks CodeWarrior

【 Windowsのシステム条件 】

項目	条件
ハードウェア	Intel Pentiumプロセッサを搭載したコンピュータ 40 MBのディスクスペース CD-ROMドライブ
ソフトウェア	Windows 95 もしくは Windows 98 もしくは Windows NT、及び、 MicroSoft Visual C++ もしくは Metro Werks CodeWarrior

【 UNIXのシステム条件 】

項目	条件
ハードウェア	Intel Pentiumプロセッサもしくは Motorola PowerPCを搭載したコンピュータ 40 MBのディスクスペース CD-ROMドライブ
ソフトウェア	GNU 開発環境

開発時の条件

HiBase のアプリケーションプログラムの開発は、Macintosh、Windows 95、Windows 98、Windows NT、UNIX系OS のいずれか（もしくは全て）が必要です。

実行時の条件

HiBase で開発したアプリケーションプログラムを実行するためは、Power Macintosh、または、68030以上のチップを搭載した68KのMacintosh、または、Intel Pentium プロセッサを搭載しWindows 95/98/NT で運用するコンピュータ、Intel Pentium プロセッサもしくは Motorola PowerPCを搭載しPC-UNIX(linux、FreeBSDなど)で運用するコンピュータのいずれか（もしくは全て）が必要です。

HiBaseは、最低 256KB のメモリを実行時に必要とします。

１－２－２．HiBase 開発環境の構成

HiBase 開発環境は、HiBase マスターディスクの「Develop」ディレクトリの下に、開発するプラットフォーム別に

「forMac」　：Macintosh用

「forWin32」：Windows 95/98/NT用

「forUNIX」： UNIX 用

として納められています。

これら「forMac」、「forWin32」、「forUNIX 」のそれぞれは、同じ目的を持つ同じ名前の以下のようなディレクトリで構成されます。

lib.....HiBase ライブラリ

include.....HiBase のヘダーファイル

src.....HiBase のソースコードファイル

xsrc.....HiBase の（ネットワーク関連）ソースコードファイル

makelib.....HiBase ライブラリを生成するプロジェクト

utils.....HiBase ユーティリティを生成するプロジェクト

samples.....サンプルプログラムのプロジェクト

mmsamples.....（マルチメディア）サンプルプログラムのプロジェクト

【 注意 】

「HiBase 標準環境」には「ソースコード」が含まれません。

独自の環境でのHiBase のアプリケーションプログラム開発を希望する場合は､「HiBase プロフェッショナル環境」へのアップグレードをお薦めします。

アップグレードについての詳細は、サポート窓口（support@lists.hln.co.jp）へお問い合わせください。

１-２-２-１．HiBase ライブラリ

HiBase ライブラリは、それぞれのプラットフォーム依存フォルダ（上記の、「forMac」、「forWin32」、「forUNIX」）内の「lib」フォルダに以下の名前で納められています。

Win32（Windows 95/98/NT）対応ライブラリ

HBSW32.lib.....シングルクライアント実行環境用

HBSW32(D).lib.....シングルクライアント実行環境用（デバッガ情報付）

HBMW32.lib.....クライアントサーバー実行環境用

HBMW32(D).lib.....クライアントサーバー実行環境用（デバッガ情報付）

Power Macintosh対応ライブラリ

HBSPPC.lib.....シングルクライアント実行環境用

HBSPPC(D).lib.....シングルクライアント実行環境用（デバッガ情報付）

HBMPPC.lib.....クライアントサーバー実行環境用

HBMPPC(D).lib.....クライアントサーバー実行環境用（デバッガ情報付）

68k Macintosh対応ライブラリ

HBS68k.lib.....シングルクライアント実行環境用

HBS68k(D).lib.....シングルクライアント実行環境用（デバッガ情報付）

HBM68k.lib.....クライアントサーバー実行環境用

HBM68k(D).lib.....クライアントサーバー実行環境用（デバッガ情報付）

UNIX対応ライブラリ

libhbs.a.....シングルクライアント実行環境用

libhbs_d.a.....シングルクライアント実行環境用（デバッガ情報付）

libhbm.a.....クライアントサーバー実行環境用

libhbm_d.a.....クライアントサーバー実行環境用（デバッガ情報付）

これらバイナリ形式のライブラリは、以下の条件で動作します。

□ Win32（Windows 95/98/NT）対応ライブラリは、MicroSoft Visual C++開発環境で、スタティックリンクライブラリとして利用できます。

□ Power Macintosh対応ライブラリおよび、68k Macintosh対応ライブラリは、Metrowerks CodeWarrior開発環境で、スタティックリンクライブラリとして利用できます。

□ UNIX対応ライブラリは、g++（GNU）を利用した開発環境の、スタティックリンクライブラリとして利用できます。

【注意】

上記以外のプラットフォームで利用する場合は、バイナリ形式のライブラリを生成し直す必要が有ります。この場合、HiBaseのソースコードが必要となり、ソースコードライセンスが必要です。

【ライブラリの命名規則】

HiBaseは、多様なプラットフォームと実行環境に対応しており、目的別に利用するライブラリを選択する必要が有ります。そのため、以下のような緩やかなライブラリの命名規則を持っています。「緩やかなライブラリの命名規則」というのは、UNIXプラットフォームに関して、UNIXでの命名規則が存在するため、以下の命名規則を適応していないという意味です。

HB<実行環境><プラットフォーム ><デバッグ >.lib

<実行環境>	クライアントサーバー対応（m）,シングルクライアント対応（s）
<プラットフォーム>	PowerMacintosh対応（PPC）、68k Macintosh対応（68k）、Win32対応（W32）
<デバッグ>	デバッガー対応版は(D)を付加

１-２-２-２．HiBase ライブラリの生成

HiBaseライブラリの生成は、それぞれのプラットフォーム依存フォルダ内の「makelib」フォルダにプロジェクトもしくはMakefileを利用しておこないます。

Win32（Windows 95/98/NT）対応ライブラリを生成するプロジェクト

HBSW32.mcp...シングルクライアント実行用ライブラリの生成

HBMW32.mcp...クライアントサーバー実行用クライアントライブラリの生成

Macintosh（Power Macintosh、68kMacintosh の両方）対応ライブラリを生成するプロジェクト

HBSMac.mcp...シングルクライアント実行用ライブラリの生成

HBMMac.mcp...クライアントサーバー実行用クライアントライブラリの生成

UNIX 対応ライブラリの Makefile

Makefile...シングルクライアント実行用ライブラリおよびクライアントサーバー実行用クライアントライブラリの生成

１-２-２-３．サンプルプログラムについて

HiBaseアプリケーションプログラム開発の参考に、サンプルプログラムを用意しています。

・まず一般的な理解のために、プラットフォーム依存フォルダに含まれる「samples」フォルダを利用してください。

「samples」フォルダには、以下のサンプルプログラムが含まれています。

◇　DBDefine....データベースの生成とスキーマの定義

◇　DBLoad.....データベースへのデータの投入（10,000レコード）

◇　DBSearch....データ検索

◇　DBDelete....データの検索と消去

上記のサンプルプログラムは、1.DBDefine、2.DBLoad、3.DBSearch、4.DBDeleteの順番で実行するようになっています。また、データベース回りの説明を強調する目的で、GUI を省略しコンソール形式で実行するものです。

・次に、「MMSamples」フォルダを見てください。これは、HiBaseの高度な利用の参考に、マルチメディアデータの投入と読み出しをおこなうテクニックを示しています。

「MMSamples」フォルダには、以下のサンプルプログラムが含まれています。

◇ MMDefine....データベースの生成と（マルチメディア対応）スキーマの定義

◇ MMText.....テキストデータの投入と読み出し

◇ MMHTML.....HTMLテキストデータの投入と読み出し

◇ MMPDF.....PDFテキストデータの投入と読み出し

◇ MMGIF.....GIFデータの投入と読み出し

◇ MMPict.....Pictデータの投入と読み出し

上記のサンプルプログラムは、MMDefine をまず最初に実行するようになっています。また、データベース回りの説明を強調する目的で、GUIを省略しコンソール形式で実行するものです。

【サポート窓口】

操作中にトラブルが生じた場合、または、HiBase に関してテクニカルな質問がある場合は、下記までお問い合わせください。また、以下に示すインターネットアドレスにアクセスすると、最新のHiBaseのアップデータやニュースを入手することができます。

◇　ホームページ

http: //www.hln.co.jp

◇　問い合わせ

テクニカル：support@lists.hln.co.jp

一般　　：sales@lists.hln.co.jp

この章では、***HiBase*** の特長、及び、***HiBase*** インタフェースプログラムの背景（***HiBase*** の基本的概念、アプリケーションプログラムインタフェースの設計概要）について説明します。

この章を一読することで、***HiBase*** とは何か、また ***HiBase*** インタフェースプログラムの目的とその使用方法を理解することができるでしょう。

この章で解説する主要な項目は、次の通りです。

2 - 1 . *HiBase* の特長

HiBase は、これまでのデータベース管理システムに見られない幾つかの独創的な特長を兼ね備えた「リレーショナルデータベース管理システム（RDBMS）」です。この章では、そのユニークな特長を紹介します。

インターネット／イントラネット対応の アプリケーションプログラム開発を支援

HiBase は、急速に成長するコンピュータ業界や「RDBMS」の世界を見つめながら、5年の歳月をかけて開発され、1998年1 月に販売をはじめました。***HiBase*** は、次世代のコンピュータの新しい可能性を実現するために、あなたが手掛けるアプリケーションプログラムの心臓部になりたいと考えています。

HiBase でのアプリケーションプログラム開発は、データベース管理システム特有の難しい考え方を必要としません。***HiBase*** のインタフェースプログラム（クラスライブラリ）を「CodeWarrior（又は、Visual C++）プロジェクト」に追加するだけで、すぐにプログラミングを開始することができます。

HiBase は、Client / Server モデル、Multi User / Multi Thread 環境を標準としています。そのため、***HiBase*** のインタフェースプログラムを利用したアプリケーションプログラムは、自動的にインターネット／イントラネット対応の「グループウェア」として機能し、開発者が独自にTCP / IP に関するプログラミングを行なう必要がありません。

Macintosh、UNIX、Windows の マルチプラットフォーム上で稼働／高移植性

HiBase のエンジンは、Macintosh、Windows （Windows31, 95, 98, NT)、UNIX など、現在考えられる全ての汎用プラットフォーム上で稼働し、特殊なOSへの移植も短期間で行えるよう「移植性・柔軟性」に重点を置いて設計されています。

また、***HiBase*** は、汎用のWeb Browser（Netscape Navigator、Microsoft Internet Explorer ect..)」からのデータベースアクセスを可能 にしました。

各環境で作り上げたデータベースファイルには、ハードウェアやOS に左右されないバイナリレベルでのコンパチビリティーがあります。そのため、非常にユーザコネクティビティの高いトランザクション処理の実現が可能となります。（***HiBase*** を移植することで、例えば、Macintoshで作り出したデータベースをWindowsにコピーして使う、Web Browserでブラウズする... 、或いは、メーカー独自の業務アプリケーションを開発するといったことが容易に可能となります。）

APIを重視した 高度なレイヤー構造

HiBase 開発の当初の目的は、“ アプリケーションから利用できるデータベースエンジンの提供 ” にありました。そのため、開発当初からC 及び、Pascal 等の汎用言語をクライアントとするAPI（アプリケーションプログラムインタフェース）を備えています。

現在、***HiBase*** には、従来の「C / C++ 言語インタフェース（HiBase API)」に加え、「JAVA 言語インタフェース（HiBase JAVA API)」が用意され、「データベースエンジン」を中核として、「HiBase API」、「JAVA API」、「SQL サーバ」、「SQL プロセッサ」、「ODBC ドライバ」... を組み合わせた高度なレイヤー構造となっています。

HiBase のクライアントプログラムをJAVA言語で開発し、「アプレット」として実行することで、汎用ブラウザからデータベースのトランザクション処理を行なうことができます。また、面倒なCGI 開発を伴わずに、ダイナミックな「WEB Site」を構築することも可能です。

更に、現在、複数の利用者から「SQL 言語」でのデータベースアクセス要求を実行する「SQL Server」を開発中です。（このプロセッサは、「ODBC/ JDBC」との組み合わせが可能な設計です。）

これらの利用により、***HiBase*** を他のRDBMSと共存させ、将来に渡って安定したデータプロセシング環境を運用することが保証されます。開発者は、用途に合わせて「汎用言語」、「JAVA 言語」及び「SQL」を選択することができ、***HiBase*** のどの階層からでも、データベースへのアクセスが可能となります。

他に類を見ない柔軟性
高速で快適なデータベース環境

HiBase は、どんな分野のアプリケーションプログラムにも適応する、汎用化の目的 を持って設計され、「RDBMS」であると同時に、「ソリューションベース」という側面をも兼ね備えています。また、***HiBase*** のデータ操作は、検索条件に一致するレコード群を「集合」として生成／管理する「集合演算指向」のトランザクション処理を基本としているため、大量データでも高速に処理することが可能です。この機能はGUI環境下でデータベースを利用するアプリケーションプログラムにとって、特に便利な機能と言えます。

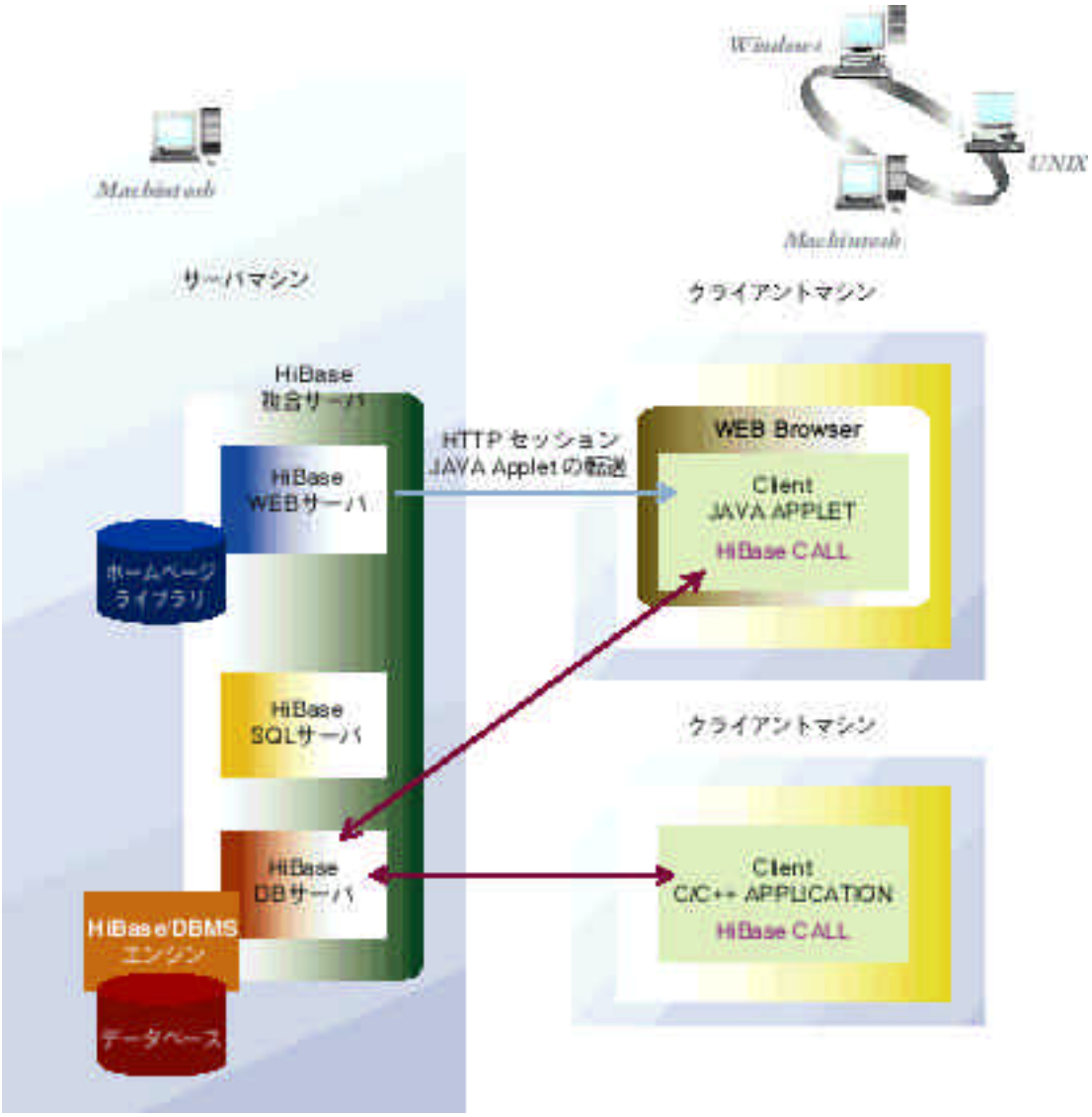
通常「RDBMS」は、１項目に１つの「値」しか格納できませんが、***HiBase*** は、レコード内の全ての項目に、複数個の値（「マルチバリュー」）が格納可能な設計です。また、データスキーマとして「グラフィックス」、「サウンド」、各種「ドキュメント」... などもサポートしています。

HiBase は、「項目」のデータタイプ、サイズ、値の数、項目定義有無まで含め、全てが実行時に結合するという、他に例を見ない柔軟性を備えています。ダイナミック生成によりデータを格納する際、「スキーマ」に定義されていない「項目」に「値」が出現した場合でも、ダイナミックに「項目」を生成して「値」を格納します。
（例えば、「人事管理スキーマ」に「住所」が定義されていなくても、「住所データ」を格納することができます。）

また、***HiBase*** に定義されている「項目」と異なる長さ、異なるデータ型でも、データベースアクセスが可能です。
（例えば、「人事管理スキーマ」に「20バイト」と定義した「名前アイテム」に、「50バイト」の「名前データ」を格納するといったことも可能です。）

更に、「項目最大長」の制限を取り去りるなど、***HiBase*** には、「データベースの応用分野」を大きく拡げるための改善が随所に凝らされています。

【 *HiBase* を利用した データベース管理システム 】



2-2. *HiBase* アプリケーションプログラムインタフェースの概要

HiBase の実体は、C++で開発されたプログラム集団の、巨大なクラスライブラリとなっています。そして、このクラスライブラリは、「HiBase エンジン」、「HiBaseAPI」、「サポートユーティリティ」に分類することが出来ます。

「HiBase エンジン」は、ハードディスク上に展開されるデータベース空間を直接操作するもので、データベース管理システム実体といえます。「HiBase API」は、アプリケーションプログラムと「HiBase エンジン」の中間に存在するインターフェースです。両者を分離することで、「HiBase エンジン」の機能拡張などの変更が行いやすくなっています。「サポートユーティリティ」は、アプリケーションプログラムが「HiBaseAPI」を使う際の面倒な作業を代行するいろんな機能を提供します。

このドキュメントは、「HiBaseAPI」および「サポートユーティリティ」の概要説明を行うのが目的です。

【 HiBase API の構造 】

HiBase APIには、次のような機能が用意されています。

この章では、以下に示す「***HiBase*** API」の各機能概略、及び、その理解に必要な基礎的概念を、項目順に説明していきます。
これらの解説を一読することで、「***HiBase*** API」の大筋の概要（どのような目的のために、何クラスが用意されているのか）を理解することができます。

HiBase API

「***HiBase*** API」は、***HiBase*** の「データベース」を操作するための機能を提供するもので、***HiBase*** の全てを覆っています。
具体的には、以下のような基本機能を提供します。

- ◇ ***HiBase*** の開始／終了 （***HiBase*** への接続 ）
- ◇ データベース管理
- ◇ トランザクション管理

HiBase を利用するアプリケーションプログラムは、まず最初に ***HiBase*** との接続に必要な各「データベースソケット」を準備した後、「***HiBase*** API（HDBHandle クラス）」のインスタンス（オブジェクト）を生成します。

「データベースソケット」とは、HiBase の多彩な実行環境に応じて、アプロケーションプログラムとHiBase の結合形態を決定するもので、以下のものが準備されています。
HBSBoss.....シングルスレッドで動作するHiBaseエンジンを起動します。
HBMBoss....マルチスレッドで動作するHiBase エンジンを起動します。
HXBoss.....HiBaseサーバーに接続するクライアントスレッドを起動します。

【注意】
UNIX およびWin32では上記の全ての「データベースソケット」を利用することができます。しかしMacintoshでは、HBMBossを利用することができません。これは、Macintoshの「スレッド」に技術的な制約が存在するためです。

HiBase の開始／終了、データベース構造のメンテナンス（DD/D 関連）、レコードの追加／削除／変更／読み出し／検索 ... 等のトランザクション処理（DML 関連）は、「HDBHandle クラス」を利用して行ないます。

サポートユーティリティ I

「***HiBase*** API」が「***HiBase*** 全ての機能の提供」であるのに対し、「サポートユーティリティ」は、「アプリケーションプログラムからの使い易さ」と「プラットフォーム独立」を目的に、「***HiBase*** API」を補佐するものです。
「サポートユーティリティ」は、具体的に以下のような機能を提供します。

スキーマ変換

「スキーマ変換」とは、***HiBase*** の「内部形式レコード」をアプリケーションプログラムの要求する「外部データ構造」に変換する機能です。
一般的に、アプリケーションプログラムが ***HiBase*** のデータ編集を行なう場合は、「スキーマ変換クラス（HRecord)」を利用し、項目を「レコード」単位で編集します。

I / O メディアの抽象化

「HDir（ディレクトリを表現するクラス)」、「HFile（ファイルを表現するクラス)」は、I / O メディアを抽象化し、物理的なアクセスを補佐するクラスです。
これらのクラスには、「ディレクトリ（フォルダ)」や「ファイル」を操作する「メンバ関数」、及び、***HiBase*** の利用環境の構築に利用される「スタティックメンバ関数」等が定義されています。

ファイル／メモリをストリームに抽象化

「HStream （ファイル／メモリを ストリームに 抽象化するクラス)」は、***HiBase*** が利用する「メモリ」を「ストリーム」に抽象化して表現する「基本クラス」です。
ストリーム操作を理解することで、プログラム中での、ファイル、動的メモリ（ハンドル、ポインタメモリ)、静的メモリ（スタック内メモリ）等の効率的な管理が可能となります。

「サポートユーティリティ」の用途は ***HiBase*** の多岐の機能に渡ります。
そのため、本ドキュメントでは「サービスクラス」に関する説明を上記解説に留め、省略しています。

「サービスクラス」に関する詳細は、別冊の「プログラムリファレンス」を参照してください。

【参照】

◇　プログラムリファレンス；　第３章 - その他のサービスクラス

2-2-1. *HiBase* API　- ***HiBase*** にアクセス...

「***HiBase*** API」は、***HiBase*** を操作するための機能を提供するもので、***HiBase*** データベースの全ての機能を覆う、最も重要なAPIです。

この章では、「***HiBase*** API」の生成と各機能の概略、及び、その理解に必要な基礎的概念について説明します。説明する主要な項目は、次の通りです。

◇ ***HiBase*** への接続概要
- ソケットの準備と「基本 ***HiBase*** API」の生成
- ***HiBase*** の開始／終了

◇ データベースの管理と ***HiBase*** のデータベース構成

◇ ***HiBase*** でのトランザクション処理
- 基本的なレコード操作
- ***HiBase*** の「集合」（データ検索）
- シーケンシャルアクセス機能

2-2-1-1. *HiBase* への接続

HiBase を利用するアプリケーションプログラムは、まず最初に「データベースソケット」を生成し、次に「***HiBase*** APIであるDBHandle」を生成します。この「DBHandle」の生成のとき、コンストラクタに「データベースソケット」を渡し、***HiBase*** との結合を行ないます。

「データベースソケット」は、***HiBase*** の実行形態によって、3種類に分れています。

◇ シングルモード（ノンクライアントサーバー）利用時

→HBSBoss クラス（シングルスレッドHiBaseを起動）

→HBMBoss クラス（マルチスレッドHiBaseを起動）

◇ マルチモード（クライアントサーバー）利用時

→HBXBossクラス（HiBase サーバーへの接続ソケット）

HiBase への接続

以下に、各利用モードにおける ***HiBase*** への接続概要を説明します。

シングル結合ソケット

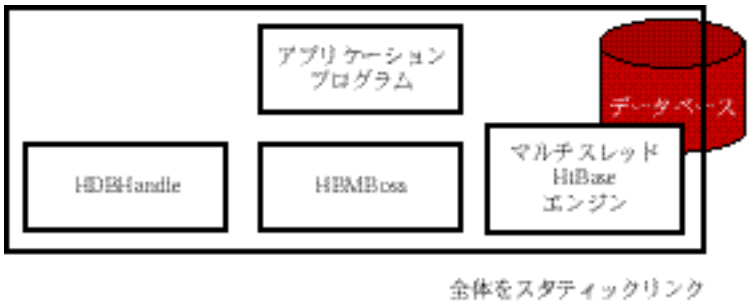
HiBase を「シングルモード」でアクセスする場合は、アプリケーションプログラムに「***HiBase*** のデータベースエンジン」を直接リンクして実行することになります。

アプリケーションプログラムは、「***HiBase*** API（HDBHandle クラス）」の生成に先立って、「シングル結合ソケット」を生成し、これをパラメータとして「HDBHandle クラス」のコンストラクタに渡します。

その後、***HiBase*** の開始宣言（Open）を行ない、データベースアクセスを開始します。

【***HiBase*** との結合（シングル結合ソケット： HBSBoss クラス）】

【***HiBase*** との結合（シングル結合ソケット： HBMBossクラス）】



【 主要なメンバ関数 】

以下に、***HiBase*** への接続に関連する主要なメンバ関数を紹介します。

HBSBoss / HBMBoss / HBXBossクラス

ソケットの準備

クラス名	用途
HBSBoss	シングル（シングルスレッド）モード
HBMBoss	シングル（マルチスレッド）モード
HXBoss	マルチモード

HDBHandle クラス

HiBase の開始／終了

クラス/関数名	機能
HDBHandle	HiBaseAPIの生成
HiBase の開始／終了	
Open	***HiBase*** の開始
Close	***HiBase*** の終了

マルチ結合ソケット

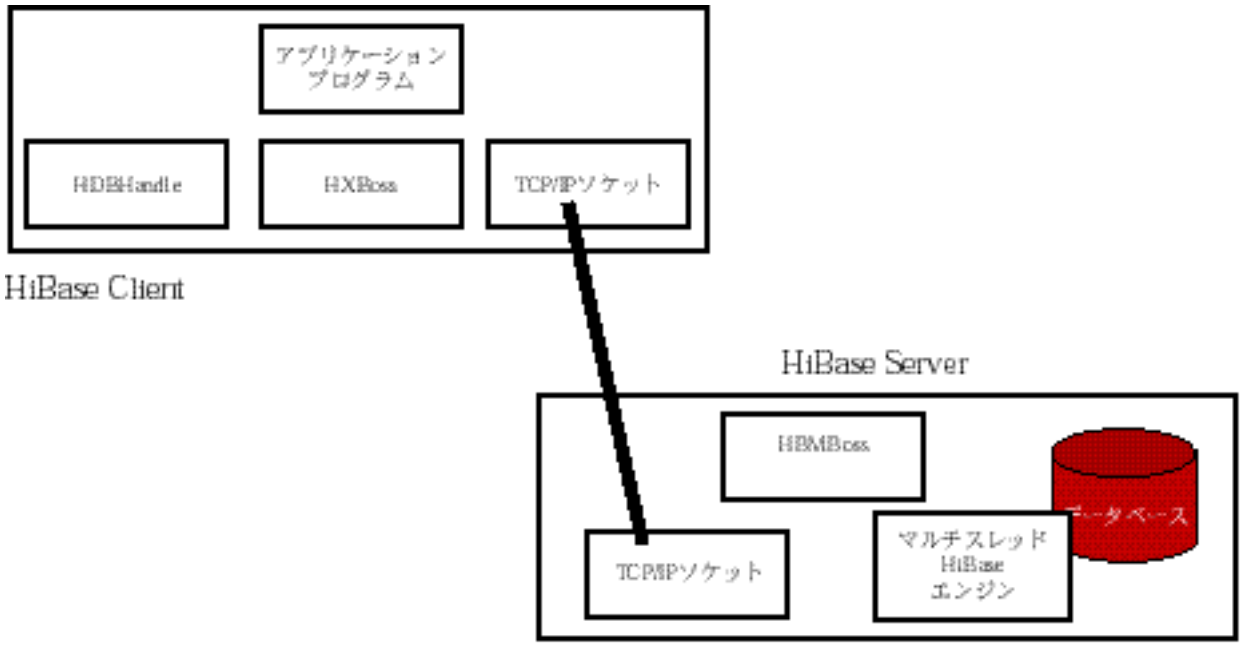
HiBase を「マルチモード」でアクセスする場合は、アプリケーションプログラムとは別のタスクで「***HiBase*** のデータベースサーバ（サーバプログラム）」を実行しておかなければなりません。

アプリケーションプログラムは、***HiBase*** API（HDBHandle クラス）」の生成に先立って、「マルチ結合ソケット（HXBossクラス）」を生成し、これをパラメータとして「HDBHandle クラス」のコンストラクタに渡します。

その後、***HiBase*** の開始宣言（Open）を行ない、データベースアクセスを開始します。

クライアントであるアプリケーションプログラムと ***HiBase*** のサーバプログラム間は、「 TCP/IP 」で結合されます。また、複数のユーザ（アプリケーションプログラム）が、同時にデータベースをアクセスすることが可能です。

【 ***HiBase*** との結合（マルチ結合ソケット： HNetClient クラス）】



なお、各利用モード用のライブラリ、及び、各利用モードの概要に関しては、前の「第１章 - インタフェースプログラムの構成 ; ***HiBase*** の利用モード」を参照してください。

【参照】

◇　第１章 - インタフェースプログラムの構成 ; ***HiBase*** の利用モード

【参照】

以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するプログラミングの概要】 ⇒

◇ 第３章 - アプリケーションプログラミングの概要; ***HiBase*** の開始／終了

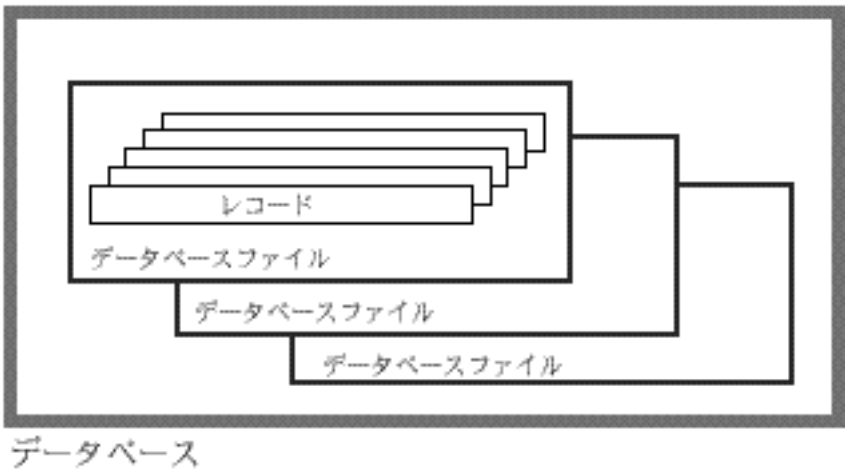
【以上の内容に関連するAPIの詳細】 ⇒ プログラムリファレンス（別冊）

◇ 第１章 - API ; Boss クラス_ソケットの準備、
HDBHandle クラス_ ***HiBase*** の開始／終了

2-2-1-2. *HiBase* のデータベース

アプリケーションプログラムから見た場合、「***HiBase*** のデータベース」は次のように見えます。

【 アプリケーションプログラムから見た ***HiBase*** データベース 】



HiBase の「データベース」の中には、最大64,000個までの「データベースファイル」があり、それぞれ「レコード」を含んでいます。

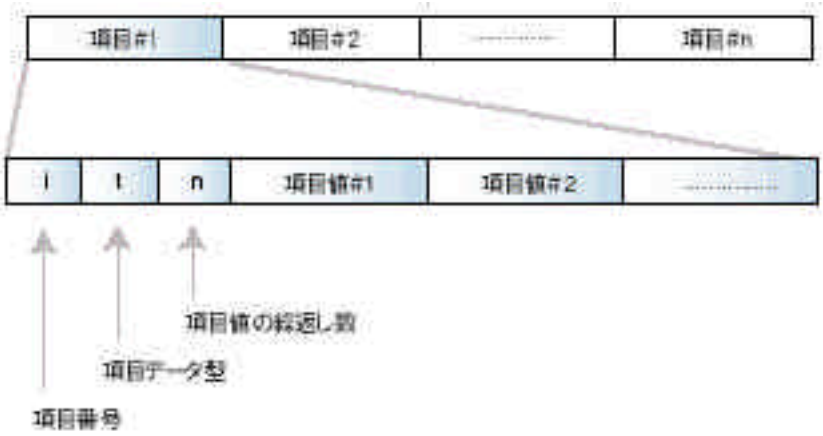
データベースファイル

「データベースファイル」は、「レコード」の集団です。

レコード

「レコード」は「項目（アイテム）」の並びで、それぞれの「項目」は「値」を保持しています。

【 *HiBase* データベースファイル内のレコード構造 】



「項目」に保持する「値」は、一つでもいいし、複数でも構いません。最大255個までの複数の値（「マルチバリュー」）を一つの項目に格納することが出来ます。このマルチバリュー機能は、***HiBase*** データベースの特長である柔軟なスキーマ構造の基本です。

また、「項目データ型」には、「文字列」、「10進数字」、「バイナリ数字」、「日付」、「時間」、「日付＆時間」、「フラグ」、「1バイト文字」、「バイナリ」の他、「絵（GIF、JPEG、PNG、PICT、BMP）」、「文章（Text、HTML、PDF）」、「ファイル（DOSファイル、Macintoshファイル）」が用意されています。さらに、127個までの「ユーザ定義データ型」を追加することができます。

アプリケーションプログラムは、この「レコード」に対し、追加／削除／変更／読み出し／検索... 等のトランザクション処理を行います。

◇　例えば、「住所録」を定義し、

一人一人の「名前」、「電話番号」、「住所」、「郵便番号」といった項目を「レコード」として作り出し、これを「ファイル」に格納し、名前で検索したり、住所を変更したり... といったトランザクション処理を行います。

なお、「データベースファイル」、「項目」、「キー」の各定義情報は、「***HiBase*** API（HDBHandle クラス）」を利用して、読み出しや更新が可能です。これらのデータ構造に関しては、後の「第３章 - アプリケーションプログラミングの概要; データベースの管理」で説明します。

2-2-1-3. *HiBase* でのトランザクション処理

アプリケーションプログラムが「トランザクション処理」を行なう場合は、「スキーマ変換クラス（HRecord)」を使って「レコード編集」をし、「***HiBase*** API（HDBHandle クラス)」の DML 関連のメンバ関数を実行します。

これらのメンバ関数は、アプリケーションプログラムが管理すべきデータを、「レコード」の形で ***HiBase*** のデータベースに出し入れする機能を実行します。
具体的には、以下のような機能を実行します。

- ◇ レコードの直接読み出し／追加／削除／更新
- ◇ データ検索 （***HiBase*** の「集合」操作機能）
- ◇ シーケンシャルアクセス

一般的に、データベースを利用するアプリケーションプログラムは、「検索」で目的の「レコード群」を探し、これを表示／削除／更新するという処理を繰り返します。***HiBase で***は、「検索」で探しだしたレコード群を「集合」として HiBase 内部に保持し、この集合のエレメントである「メンバレコード」に対し、読み出し／削除／更新を実行するスタイルになります。

基本的なレコード操作

「レコード」操作を行なう場合は、トランザクションの確立（「HDBHandle::TRCommit」）後、以下のメンバ関数を利用します。

A） レコードの読出	→	**GetRecord**
B） レコードの追加	→	**InsRecord**
C） レコードの削除	→	**DelRecord**
D） レコードの更新	→	**UpdRecord**

HiBase では、アプリケーションプログラムがデータベースファイル内にレコードを追加する場合、「スキーマ変換クラス（HRecord)」を使って「レコード編集」をおこない、InsRecord 関数を呼び出します。
この「スキーマ変換クラス（HRecord)」の機能に関しては、後の「スキーマ変換クラスの機能」で説明します。

また、レコードの読み出し／更新／削除などを行なう場合は、「Setメンバ関数群」を使って「集合」を作成し、集合内のレコードを、GetRecord／UpdRecord／DelRecordの各関数で処理します。
HiBase の「集合」機能に関しては、次の「HiBase の集合」で説明します。

【主要なメンバ関数】

以下に、***HiBase*** のレコード操作に関連する主要なメンバ関数を紹介します。

HDBHandle クラス

トランザクション処理（DML 関連）

関数名	機能
トランザクションの確立／キャンセル	
TRCommit	トランザクションの確立
TRCancel	トランザクションのキャンセル
レコード操作	
GetRecord	レコードの直接読み出し
InsRecord	レコードの追加
DelRecord	レコードの削除
UpdRecord	レコードの更新
データベースファイルの情報取得	
MaxRNbr	最大レコード番号の取得
FileSize	レコード総数の取得

【参照】

◇　第２章 - 拡張HiBase API ; スキーマ変換クラスの機能

なお、以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するプログラミングの概要】 ⇒

◇　第３章 - アプリケーションプログラミングの概要;
_トランザクション処理 > 一般的なレコード操作)

【以上の内容に関連するAPI の詳細】 ⇒ プログラムリファレンス（別冊）

◇　第１章 - 基本API ; HDBHandle クラス
_トランザクション処理（DML 関連）
> トランザクションの確立／キャンセル, レコード操作)

HiBase の「集合」

HiBase のデータ操作は、レコードを構成している「項目」を「キー」に指定して検索を行い、見つかったレコード群を、***HiBase*** 内部に「集合」として作り出し、これ対し、読み出し／削除／更新を実行するといった集合演算指向のトランザクション処理を基本としています。

集合のエレメントである「メンバレコード」は、「インデックス」で管理されます。そのため、大量データでも高速に処理することが可能です。

この ***HiBase*** の「集合」機能 は、アプリケーションプログラムの大量データ処理を著しく簡素化し、開発工数を下げることができます。
特に、GUI環境下でデータベースを利用するアプリケーションプログラムにとって便利な機能と言えます。

「集合」には、**キー／項目／絞込**などの検索タイプがありますが、これに「母集合」や「集合演算」を組み合わせることで、更に複雑な条件での検索が可能となります。

集合操作には、以下のような多彩な関数群が用意されています。

◇ 直に対して、次のような検索の指定ができます。

- 完全一致
- 前方一致
- 後方一致
- 中間一致

◇ 集合同士の「論理演算（and / or / not）」による新たな「集合」の作成

◇ 集合内のレコード項目値の降順／昇順による並び変え（**集合ソート**）

HiBase は、これら集合操作により作成された「レコード集団」をビットマップに変換、保存することでディスク容量をコンパクト化し、集合演算を高速化しています。またそれぞれの集合を保持するのに消費するメモリもごくわずかですみます。

更に、データベースI / O バッファとして、独自のディスクキャッシュをメモリ上に確保し、時間のかかるディスクI / O を最小限に抑え、処理の高速化／データベース更新の効率化を計っています。

（これらバッファ全体は、最も頻繁に要求される場所がメモリ上に残るよう自 己学習するアルゴリズム「LRU」によって管理される設計です。）

「データ検索」を行なう場合は、以下のような手順となります。

1. 「集合」の生成＝検索の実行

キー検索集合の作成　→　SetMake

実行の結果、***HiBase***の内部にレコード群としての「集合」が作られ、その集合の識別子がアプリケーションプログラムに返されます。
SetMake の実行の時、検索条件式をパラメータに指定します。この検索条件式は、「キー項目条件式（例えば、キー項目=XXXX）」、「キーでない項目条件式（例えば、項目=XXXX）」、「既に作られた集合」を AND / OR / NOT の演算子で連結するスタイルになります。

2. 集合内の「レコード」の読み出し　（「HDBHandle:: SetRGet」）

◇　例えば、集合内の全レコードを読み出す場合...

(a) 「HDBHandle:: SetSize」で「集合」内のレコード数を取得し、
(b) 「HDBHandle:: SetRGet」で個々の「レコード」をインデックスを指定して読み出します。

インデックスゼロから、(a)で得たレコード数まで、読み出し（SetRGet）を繰り返し呼び出すことで、「集合」内の全てのレコードを読み出すことができます。

◇　集合内にレコードを追加／削除する場合...

(a) 「HDBHandle:: SetRAdd」で集合にレコードを追加、
(b)「HDBHandle:: SetDel 」で集合からレコードを削除することができます。

また、「HDBHandle::SetSort」を呼び出して、「集合」内のレコードのソートを行なうことができます。

【 主要なメンバ関数 】

以下に、***HiBase*** の集合操作（データ検索）に関連する主要なメンバ関数を紹介します。

HDBHandle クラス

DML 関連

関数名	機能
集合化の操作	
SetMake	集合の生成
SetSort	集合内のソート
集合内のレコード操作	
SetSize	集合内レコード数の取得
SetRGet	レコードの読み出し
SetRAdd	レコードの追加
SetRDel	レコードの削除
SetCancel	集合の削除
キーの追加／削除	
CreateKey	キーの追加
DeleteKey	キーの削除

【 参 照 】

◇　第２章 - 拡張HiBase API ; スキーマ変換クラスの機能

なお、以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するプログラミングの概要】 ⇒

◇　第３章 - アプリケーションプログラミングの概要;
_トランザクション処理 > 一般的なレコード操作 > 集合の作成

【以上の内容に関連するAPI の詳細】 ⇒　プログラムリファレンス（別冊）

シーケンシャルアクセス機能

「基本 ***HiBase*** API（HDBHandle クラス）」には、全データを対象とする「トランザクション処理」を想定し、次の２種の「シーケンシャルアクセス」機能が用意されています。

これらの機能は、データベース内の「全レコード」を対象に、以下のような処理を行なう場合に有効な機能です。

◇　１つの「キー」の値の順番に「レコード」を順次読み出す場合

→　**キーシーケンシャルアクセス**

◇　「全レコード」を高速に読み出す場合

→　**高速シーケンシャルアクセス**

シーケンシャルアクセス機能を利用する場合は、以下のような手順となります。

キーシーケンシャルアクセス

「キーシーケンシャルアクセス」を実行する場合は、（「HDBHandle:: KSメンバ関数群」）を利用します。

アプリケーションプログラムは、まずKSLocate 関数を呼び出し、読み出しの開始場所を指定後、KSReadを繰り返し呼びだします。
KSRead関数を繰り返し呼び出すことで、キーの値、その値を持つレコードの数、とレコード番号のリストが、キーの値の昇順で返されます。。

【注意】

キーシーケンシャルアクセス機能は、「レコード番号」のみを読み出します。
そのため、「レコード本体」が必要な場合は、続けて GetRecord 関数を呼び出し、「レコード」の読み出しを実行する必要があります。

高速シーケンシャルアクセス

「高速シーケンシャルアクセス」を実行する場合は、（「HDBHandle:: PS メンバ関数群」）を利用します。

アプリケーションプログラムは、まずPSLocate 関数を呼び出し、読み出しの開始を宣言を行ない、その後PSReadを繰り返し呼びだします。
PSRead関数を繰り返し呼び出すことで、データベース内の「レコード」が、格納順に、順次読み出されます。

なお、読み出されたレコードは、いずれの場合も、「スキーマ変換クラス（HRecord）」を利用して編集します。（「スキーマ変換クラス（HRecord）」の機能に関しては、後の「スキーマ変換クラスの機能」で説明します。）

【主要なメンバ関数】

以下に、シーケンシャルアクセス機能に関連する主要なメンバ関数を紹介します。

HDBHandle クラス

トランザクション処理（DML 関連）

関数名	機能
キーシーケンシャルアクセス	
KSLocate	読み出し開始位置の設定
KSRead	キー順次読み出しの実行
KSCancel	読み出しの終了
高速シーケンシャルアクセス	
PSLocate	読み出し開始の宣言
PSRead	順不同高速読み出しの実行
PSCancel	読み出しの終了

【参照】

◇　第２章 - 拡張HiBase API ; スキーマ変換クラスの機能

なお、以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するプログラミングの概要】 ⇒

◇　第３章 - アプリケーションプログラミングの概要;
_トランザクション処理 > シーケンシャルアクセス

【以上の内容に関連するAPI の詳細】 ⇒ プログラムリファレンス（別冊）

◇　第１章 - 基本API ; HDBHandle クラス
_トランザクション処理（DML 関連）> シーケンシャル処理

2-2-2. サポートユーティリティ

「***HiBase*** API」が「***HiBase*** 全ての機能の提供」であるのに対し、「サポートユーティリティ」は、「アプリケーションプログラムからの使い易さ」を目的に、便利な機能を提供します。

この章では、「拡張 ***HiBase*** API」の各機能が提供する内容の概略、及び、その理解に必要な基礎的概念について説明します。

説明する主要な項目は、次の通りです。

◇ 「スキーマ変換」クラスの機能

- データ構造の変換
  （***HiBase*** の「内部形式レコード」と「外部データ構造」）
- スキーマ変換フォーマット

2 - 2 - 2 - 1 . スキーマ変換クラスの機能

「スキーマ変換」は、***HiBase*** の「内部形式レコード」をアプリケーションプログラムの要求する「外部データ形式」に変換します。

「スキーマ変換クラス（HRecord）」を呼び出す場合は、「書き込みレコード編集用」或いは、「読み出しレコード編集用」のいずれかの「コンストラクタ」を利用して、「スキーマ変換クラス（HRecord）」のインスタンス（オブジェクト）を生成します。

その後、Append／Fetch／Update／Delete等のメンバ関数を実行し、「項目データ」の追加／読み出し／更新／削除を行ないます。

◇ ***HiBase*** のデータベースファイルへデータを追加する場合

→ 書き込みレコード編集用コンストラクタ

◇ ***HiBase*** のデータベースファイルからデータを読み出す場合

→ 読み出しレコード編集用コンストラクタ

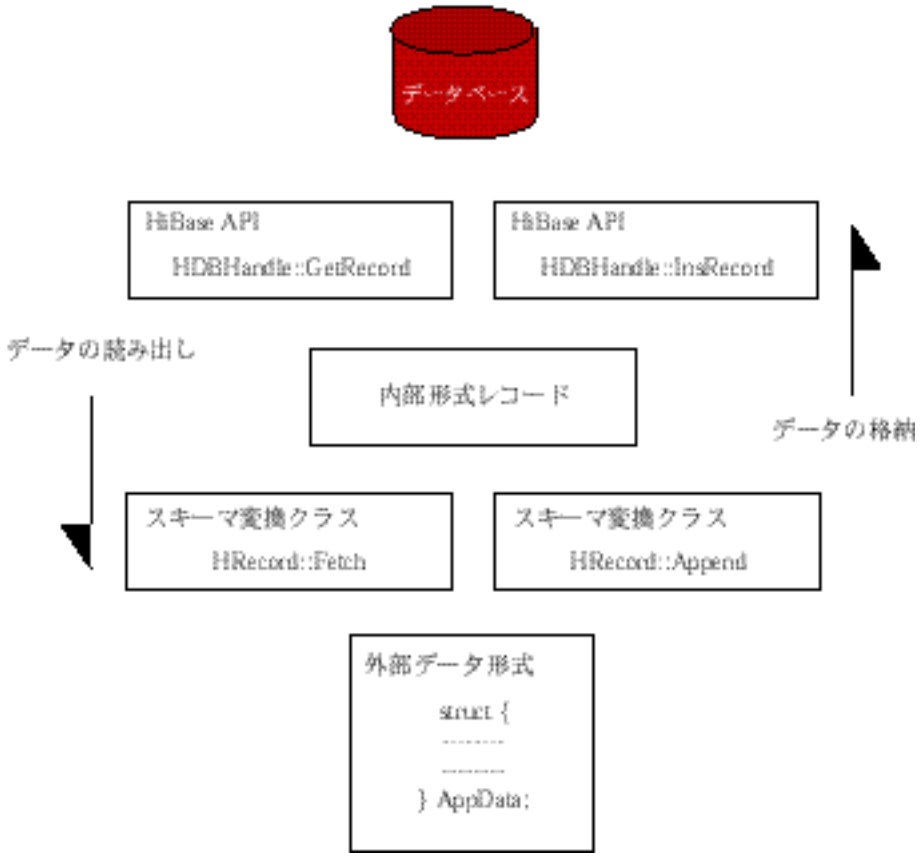
データ構造の変換

アプリケーションプログラムが、***HiBase*** の「データベースファイル」へ「データ」を追加する場合、アプリケーションプログラムの要求する「外部データ構造」で表現された「項目データ」を、***HiBase*** の「内部形式レコード」に変換して InsRecord 関数を呼び出します。

また、アプリケーションプログラムが、***HiBase*** の「データベースファイル」から「データ」を読み出す場合、GetRecord 関数で読み出した ***HiBase*** の「内部形式レコード」を、アプリケーションプログラムの要求する「外部データ形式」に変換します。

そのため、「スキーマ変換クラス（HRecord）」は、***HiBase*** の「内部形式レコード」を格納するための、ダイナミックに伸縮するメモリー（「メモリープール」）を用意し、***HiBase*** の「内部形式レコード」と、アプリケーションプログラムの要求する「外部データ形式」を、双方向に変換する機能を提供します。

以下に、「スキーマ変換」機能の概要図を示します。



レコードの書き込み （項目データの追加）

アプリケーションプログラムが、***HiBase*** の「データベースファイル」へ「レコード」を追加する場合は、アプリケーションプログラムの要求する「外部データ構造」で表現された「項目データ」へのポインタをAppend関数に渡し、***HiBase*** の「内部形式レコード」に変換しながら「メモリープール」に積んでいきます。「項目データ」を追加後、***HiBase*** API（HDBHandleクラス）」のInsRecord関数を呼び出し、この「メモリープール」へのポインタをパラメータとして渡して、レコードの追加を実行します。

項目データを追加する場合、具体的には、以下のような手順となります。

1. 「レコード編集クラス」を準備 （「HRecord:: コンストラクタ（書き込み）」）

   「書き込みレコード編集用」のコンストラクタを利用して、「スキーマ変換クラス（HRecord）」のインスタンス（オブジェクト）を生成します。

2. 「内部形式レコード」に変換 （「HRecord:: Append 関数ファミリー」）

   「HRecord:: Append」で、アプリケーションプログラムのデータ構造を ***HiBase*** の「内部形式レコード」に変換します。

   （この際、「スキーマ変換フォーマット」をパラメータに指定します。）

3. 「レコード」の追加 （「HDBHandle:: InsRecord」）

   「HDBHandle:: InsRecord」で、***HiBase*** に「レコード」を追加します。

「***HiBase*** API（HDBHandleクラス）」の生成、及び、「InsRecord 関数」に関しては、「第 2 章 - 基本HiBase API 」を参照してください

「スキーマ変換フォーマット」に関しては、後の「第 2 章 - 拡張HiBase API ; スキーマ変換クラスの機能_スキーマ変換フォーマット」で説明します。

【参照】

- ◇ 第 2 章 - 基本HiBase API ; ***HiBase*** でのトランザクション処理
- ◇ 第 2 章 - 拡張API ; スキーマ変換（HXRecord クラス）
  _スキーマ変換のパラメータフォーマット

レコードの読み出し （項目データの読み出し）

アプリケーションプログラムが、***HiBase*** の「データベースファイル」から「項目データ」を読み出す場合は、 「***HiBase*** API（HDBHandleクラス）」の GetRecord 関数で ***HiBase*** の「内部形式レコード」を読み出し、これを「スキーマ変換クラス（HRecord）」の「メモリープール」に展開します。その後、Fetch 関数を利用して、アプリケーションプログラムの要求する「外部データ構造」に変換します。

項目データを読み出す場合、具体的には、以下のような手順となります。

1. 「レコード編集クラス」を準備 （「HRecord:: コンストラクタ（読み出し）」）

   「読み出しレコード編集用」のコンストラクタを利用して、「スキーマ変換クラス（HRecord）」のインスタンス（オブジェクト）を生成します。

2. 「レコード」の読み出し （「HDBHandle:: GetRecord」）

   「HDBHandle:: GetRecord」で、***HiBase*** から「レコード」を読み出します。

3. 「内部形式レコード」に変換 （「HRecord:: Fetch 関数ファミリー」）

   「HRecord:: Fetch」で、アプリケーションプログラムの望む「外部データ構造」に変換します。

   （この際、「スキーマ変換フォーマット」をパラメータに指定します。）

「基本 ***HiBase*** API（HDBHandleクラス）」の生成、及び、「GetRecord 関数」に関しては、「第２章 - 基本HiBase API 」を参照してください

「スキーマ変換フォーマット」に関しては、後の「第２章 - 拡張HiBase API ; スキーマ変換クラスの機能_スキーマ変換フォーマット」で説明します。

【参照】

- ◇ 第２章 - 基本HiBase API ; ***HiBase*** でのトランザクション処理
- ◇ 第２章 - 拡張API ; スキーマ変換（HXRecord クラス）
  _スキーマ変換のパラメータフォーマット

【主要なメンバ関数】

以下に、スキーマ変換に関連する主要なメンバ関数を紹介します。

HRecord クラス

関数名	機能
項目データの操作	
コンストラクタ（「スキーマ変換クラス」の生成）	
HRecord	
項目データの追加　（Append 関数ファミリー）	
Append	データの追加（複数）
AppendItem	データの追加（１項目）
項目データの読み出し　（Fetch 関数ファミリー）	
Fetch	データの読み出し（複数）
FetchItem	データの読み出し（１項目）
項目データの更新　（Update 関数ファミリー）	
Update	データの更新（複数）
UpdateItem	データの更新（１項目）
項目データの削除　（Delete 関数ファミリー）	
Delete	データの削除（複数）
DeleteItem	データの削除（１項目）
項目データの情報取得	
ValueCount	値数の取得
ValueSize	データ長の取得
ValueType	データ型の取得
その他のメンバ関数	
DBRec	メモリプール内のレコード取得
Length	メモリプールの使用レングス取得
Clear	メモリの初期化

【参照】

◇　第１章 - 基本HiBase API ; ***HiBase*** でのトランザクション処理

なお、以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するプログラミングの概要】 ⇒

◇　第３章 - アプリケーションプログラミングの概要;
_トランザクション処理 > 一般的なレコード操作

【以上の内容に関連するAPIの詳細】 ⇒　プログラムリファレンス（別冊）

◇　第２章 - 拡張API ; スキーマ変換（HRecord クラス）

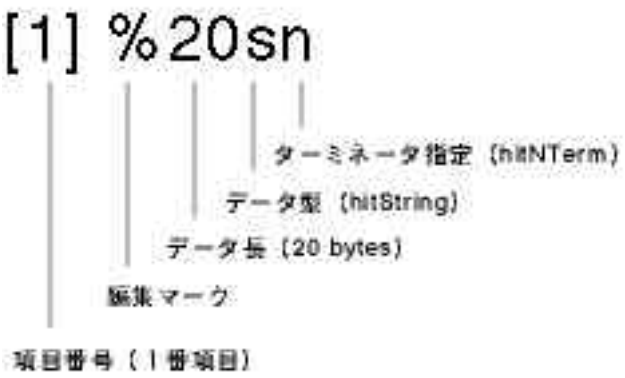
◇　第１章 - 基本API ; HDBHandle クラス
_トランザクション処理（DML 関連）
> レコード操作、データ検索、シーケンシャル処理

スキーマ変換フォーマット

「スキーマ変換クラス（HRecord）」のAppend／Fetch／Update／Delete メンバ関数ファミリーを利用する場合は、各関数の「第１パラメータ」に「スキーマ変換フォーマット（***HiBase*** を利用するアプリケーションプログラムの要求する外部データ構造）」を指定します。

以下に、スキーマ変換のパラメータ指定のサンプルを示します。

【 スキーマ変換フォーマットの指定例 】



上記サンプルは、項目番号１のデータを、20 バイトの C 言語仕様（null terminate）の文字列として扱うことを表現しています。

項目の「データ型」には、「文字列」や「数字」、「日付」、「時間」、「日付＆時間」などの「スタンダードデータ」の他に、ピクチャーや音声、テキストなど、16 M Bytes までの非常に大きなデータを扱うことのできる「バイナリタイプ」が定義されています。

「データ型」が「文字列」、「数字」、「日付」、「時間」、「日付＆時間」の場合は、データの最後を示す「ターミネータ：ヌル（Null）／タブ（HorizontalTab）／改行（CarriageReturn）」、及び、レコードの最後を示す「拡張ターミネータ」を指定することができます。

また、連続する「項目」の編集指定が同じ場合は、連続する「項目番号」をマイナス記号（- ）で繋ぐことで、指定を省略することができます。

【 省略指定例 】

◇　**[1] %st, [2] %st, [3] %st, [4] %nr;**　を省略

→　**[1] - [3] %st, [4] %nr;**

マルチバリュー

HiBase は、すべての「項目」に、複数（最大255個）の「値（**「マルチバリュー」**）」を格納することができます。
同一項目に対して、複数の値を格納する場合は、項目番号に「バリューインデックス」を付加します。

【マルチバリュー指定例】

◇　１番項目の４番目のバリューのみを指定
→　**[1 (4)]**

◇　１番項目の２番目から５番目までのバリューを指定
→　**[1 (2 - 5)]**

【例】

◇　例えば、以下のようなデータ構造に編集する場合は、

```
struct {
        char    name[32];
        char    family[32];
        char    address[50];
        int     age;
} Jinji;
```

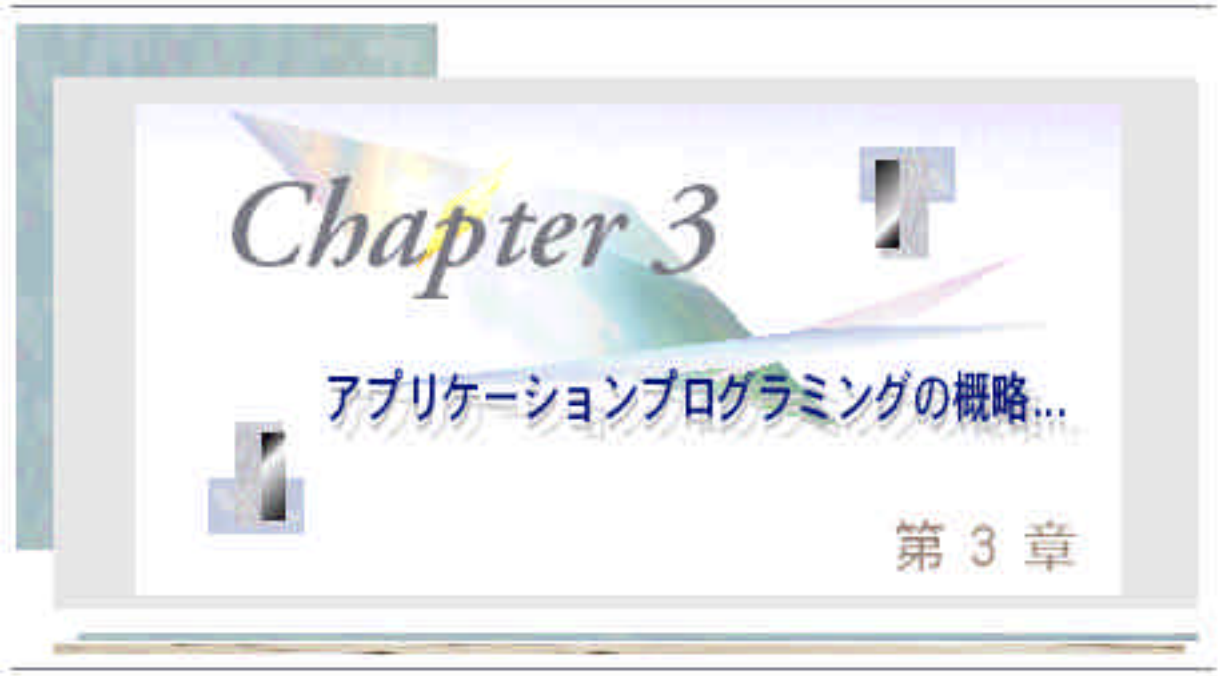
[1] %32sn, [2] %32sn, [3] %50sn, [4] %2i;　のように指定します。

【参照】

なお、以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するAPIの詳細】　⇒　プログラムリファレンス（別冊）

◇　第２章 - API；スキーマ変換（HRecord クラス）
_スキーマ変換のパラメータフォーマット



この章では、***HiBase*** を利用したアプリケーションプログラミングの概略を、一般的な流れ（***HiBase*** との接続 → データベースの生成／管理 → トランザクション処理 → 拡張機能）に添って説明します。

この章を一読することで、***HiBase*** を利用した、具体的なアプリケーションプログラミングの概要を理解することができます。

この章で解説する主要な項目は、次の通りです。

3-1. *HiBase* アプリケーションプログラミングの概要

この章では、***HiBase*** を利用した一般的なアプリケーションプログラミングの流れと、その概要について説明します。

3-1-1. 開始と終了

HiBase を利用するアプリケーションプログラムは、大体において次の形になります。

```
// ソケットを作る
//  パラメータ...環境ファイルの存在場所
HBSBoss* theBoss = new HBSBoss("../HiBase.conf");
// HiBase APIを作る
HDBHandle* theHandle = new HDBHandle(theBoss);
// HiBase APIの開始を宣言
//  パラメータ#1...ユーザID
//  パラメータ#2...パスワード
ec = theHandle->Open("HOLON", "12345678");
if (ec == ecNormal) { // エラーをチェック

 ..............
// トランザクション処理
 ..............


 theHandle->Close();

} else { // HiBase APIの開始でエラー

 ..............
// エラー処理
 ..............

}

// HiBase APIを削除
delete theHandle;
// ソケットを削除
delete theBoss;
```

3-1-1-1. データベースアクセスの開始

HiBase を利用するアプリケーションプログラムはまず最初にソ「データベースソケット」をつくります（ HBSBoss* theBoss = new HBSBoss)。
HiBase は、様々な実行形態（クライアントサーバー、マルチスレッド）に対応していますが、この実行形態を選択するのが「データベースソケット」です。言い換えますと、この「データベースソケット」の選択だけが、環境依存の部分です。
このプログラムの「データベースソケット」は、シングルモード、シングルスレッドで実行するHiBase（HBSBoss）を選択しています。
HiBaseではこのほかに

HBMBoss.....シングルモード、マルチスレッドのHiBase を起動
HBXBoss.....マルチモード（クライアントサーバ）でHiBase サーバを利用

の「データベースソケット」を利用することができます。

「データベースソケット」を生成するとき、環境ファイルが必要になります。プログラムの中では「../HiBase.conf」となっています。
この環境ファイルに関しては二つの話題があります。

まず一つは、「ファイルの存在場所の指定」です。

一般的にHiBase の世界では、「UNIX のパス文法」でファイルの存在場所を指定します。
つまり、ディレクトリを指定する際、Windowsでは「\」、Macintoshでは「:」、UNIX では「/」をデリミッタとして利用します。HiBase のような「プラットフォームに依存しない」システムはこれが大問題です。つまり、「HiBase を利用するアプリケーションプログラムがプラットフォーム依存」になってしまいます。そのため、HiBase では、インターネットやUNIX で使われている「/」をディレクトリのデリミッタとして統一しました。

もう一つは、環境ファイルの内容です。

環境ファイルの内容は、「シングルモードの時は、HiBase 本体の実行に必要な情報（例えばデータベースの存在場所)」、「マルチモードの時は、HiBase サーバーと結合するための情報（HiBase サーバの動いているホストアドレスとポート番号)」となります。

しかし、アプリケーションプログラムの開発段階で、「シングルモード」で使われるのか、「マルチモード」で使われるのか、わからない場合はあります。また、両方の使い方に対応する場合も有ります。
このため、環境ファイルは以下のように両方の情報を定義しておくことをお薦めします。

```
DBPATH   = "DBSample"
WORKPATH = "DBTemp"
CASHSIZE = 300
DDD      = on
SECURITY = on
VALUE    = on
TLOG     = off
CLOG     = on
READONLY = off

SERVER   = "www.hln.co.jp"
PORT     = 3330
```

○シングルモード情報（HiBaseの起動に関する情報）

DBPATH	**データベースの存在場所**
WORKPATH	**ワークファイルの場所**
CASHSIZE	**キャッシュサイズ**
DDD	**DD/D 利用の有無**
SECURITY	**セキュリティの有無**
VALUE	**セキュリティの有無**
TLOG	**トランザクションログの有無**
CLOG	**コマンドログの有無**
READONLY	**データベース書き込み禁止**

○マルチモード情報（HiBaseサーバーに関する情報）

SERVER	**HiBaseサーバーのホストアドレス**
PORT	**HiBaseサーバーのポート番号**

【参照】

以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するプログラミングの基礎知識】 ⇒

◇ 第１章 - インタフェースプログラムの構成 ; ***HiBase*** の利用モード

◇ 第２章 - プログラミングの基礎知識; ***HiBase*** への接続

【以上の内容に関連するAPI の詳細】 ⇒ プログラムリファレンス（別冊）

◇ 第１章 - 基本API ; HBoss クラス_ソケットの準備、
HDBHandle クラス_ ***HiBase*** の開始／終了

3-1-1-2. データベースアクセスの終了

HiBase の終了は、終了宣言（HDBHandle::Close）を実行し、生成した「***HiBase*** API（HDBHandle クラス)」と、「データベースソケット」を消去します。

具体的には、以下のような手順となります。

1. 終了宣言 （「HDBHandle::Close」）

「HiBaseAPI（HDBHandle） クラス」のClose関数を呼び出し、HiBaseにアクセスの終了を宣言します。このときHiBase は、トランザクションの終了、キャッシュに残っている情報のディスクへの反映、etc さまざまな重要な処理を実行します。例えば、この終了宣言を忘れると、HiBase は「アプリケーションプログラムの異常終了」と見なし、トランザクションのキャンセルを実施し、このアプリケーションプログラムの実行した更新をデータベースから取り去ってしまいます。

２．「基本 **HiBase API**」と「ソケット」の消去 （「HDBHandle :: Delete」）

「HDBHandle クラス」のDelete 関数を呼び出し、「基本 ***HiBase*** API（HDBHandle クラス)」と、「データベースソケット」を消去します。

まとめ

1. データベースソケットの生成（H?Boss）

1.1シングルモード（HB?Boss）

1.1.1シングルスレッド HiBase を起動（HBSBoss）

データベースソケットとして、HBSBossを起動します。

環境ファイルはシングルモード情報が指定されている必要があります。

1.1.2マルチスレッド HiBase を起動（HBMBoss）

データベースソケットとして、HBMBossを起動します。

環境ファイルはシングルモード情報が指定されている必要があります。

1.2マルチモード（HXBoss）

データベースソケットとして、HXBossを起動します。

環境ファイルはマルチモード情報が指定されている必要があります。

2. HiBaseAPI の生成（HDBHandle）

HiBaseAPI（HDBHandle）を生成します。

3. データベースアクセスの開始（HDBHandle::Open）

「HDBHandle クラス:: Open」を呼び出し、***HiBase*** の開始宣言を行ないます。この時、ユーザのアカウント（ユーザID とパスワード）をパラメータで指定します。

【参照】

以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するAPI の詳細】 ⇒ プログラムリファレンス（別冊）

◇ 第１章 - 基本API ; HBoss クラス_ソケットの準備、
HDBHandle クラス_ ***HiBase*** の開始／終了

3-1-2. データベースの管理

データベースを管理するアプリケーションプログラム（普通このようなプログラムはデータベースアドミニストレータの道具という性格を持つものです）は、「***HiBase*** API（HDBHandle クラス）」と「***サポートユーティリティ***（HSchema クラス）」を利用します。

データベースの管理とは、データベース自身の定義をメンテナンスする行為で、データベースの生成と削除、データベースファイルの生成と削除、データベースファイルのスキーマの変更を示します。本章では、データベースの生成と削除、データベースファイルの生成と削除を解説します。

3-1-2-1. データベースの生成

HiBase の「データベース」を生成する場合は、以下のような手順となります。

「データベース」の生成　（「HDBHandle :: Create」）

「***HiBase*** API（HDBHandle クラス）」のインスタンス（オブジェクト）を生成後、「Create関数」を実行します。

```
//      データベースの生成
HBSBoss* theBoss = new HBSBoss("../HiBase.conf");
HDBHandle* theHandle = new HDBHandle(theBoss);
ECode ec = theHandle->Create("TEST-Database", DB for Examples");
delete theHandle;
delete theBoss;
```

「データベースソケット」を生成し、「HiBaseAPI」を生成し、その後HDBHandle::Create を呼びだします。この時、データベースに付ける名前、メモをパラメータで渡します。また、Createの実行に関して、HiBaseAPIに開始宣言をする必要はありません。

【重要】

データベースの生成は、非常に特殊な機能であるため、また、危険な機能であるためマルチモードでの実行、および、マルチスレッドでの実行はできません。つまり、利用できる「データベースソケット」は、HBSBossにかぎります。

3-1-2-2. データベースの削除

HiBase の「データベース」を生成する場合は、以下のような手順となります。

データベースの削除（「HDBHandle :: Erase」）

「***HiBase*** API（HDBHandle クラス）」のインスタンス（オブジェクト）を生成後、「Erase 関数」を実行します。

```
//      データベースの削除

HBSBoss* theBoss = new HBSBoss("../HiBase.conf");
HDBHandle* theHandle = new HDBHandle(theBoss);
ECode ec = theHandle->Erase();
delete theHandle;
delete theBoss;
```

「データベースソケット」を生成し、「HiBaseAPI」を生成し、その後 HDBHandle::Erase を呼びだします。Erase の実行に関して、HiBaseAPIに開始宣言をする必要はありません。

【重要】

データベースの削除は、非常に特殊な機能であるため、また、危険な機能であるためマルチモードでの実行、および、マルチスレッドでの実行はできません。つまり、利用できる「データベースソケット」は、HBSBoss にかぎります。

3-1-2-3 データベースファイルの生成

HiBase の「データベース」を生成する場合は、以下のような手順となります。

スキーマを作る（HSchema::MakeSchema）
データベースファイルを生成する（HDBHandle::CreateDBFile）

```
ECode  ec = ecNormal;
char*  argv[] = {
 "[FILE:1], NAME = \"TEST-FILE-1\", LABEL = \"file001\", BLOCKSIZE = (2048, 4096)",
 "[ITEM:1], NAME = \"ID\", TYPE = String, LENGTH = 32",
 "[ITEM:2], NAME = \"DateTime\", TYPE = DateTime, LENGTH = 32",
 "[ITEM:3], NAME = \"File-Name\", TYPE = String, LENGTH = 64",
 "[ITEM:11], NAME = \"File\",     TYPE = File,     LENGTH = 4000",
 "[ITEM:12], NAME = \"MacFile\", TYPE = MacFile, LENGTH = 4000",
 "[ITEM:13], NAME = \"TEXT\",     TYPE = Text,     LENGTH = 4000",
 "[ITEM:14], NAME = \"HTML\",     TYPE = HTML,     LENGTH = 4000",
 "[ITEM:15], NAME = \"PDF\",      TYPE = PDF,      LENGTH = 4000",
 "[ITEM:16], NAME = \"JPEG\",     TYPE = JPEG,     LENGTH = 4000",
 "[ITEM:17], NAME = \"GIF\",      TYPE = GIF,      LENGTH = 4000",
 "[ITEM:18], NAME = \"PNG\",      TYPE = PNG,      LENGTH = 4000",
 "[ITEM:19], NAME = \"PICT\",     TYPE = PICT,     LENGTH = 4000",
 "[ITEM:20], NAME = \"BMP\",      TYPE = BMP,      LENGTH = 4000",

 "[KEY:1], NAME = \"ID\", TYPE = String, BIND = [1]",
 "[KEY:2], NAME = \"DateTime\", TYPE = DateTime, BIND = [2]",
 "[KEY:3], NAME = \"File-Name\", TYPE = String, BIND = [3]",
 NULL
};

HSch_DB* theSchema = (HSch_DB*) HSchema::MakeSchema(argv);

if (theSchema != NULL) {
 HBSBoss* theBoss = new HBSBoss("../HiBase.conf");
 HDBHandle* theHandle = new HDBHandle(theBoss);

 ec = theHandle->Open();
 if (ec == ecNormal) {
  ec = theHandle->CreateDBFile(theSchema);
  theHandle->Close();
 }
 delete theHandle;
 delete theBoss;
}
```

■スキーマをつくる

スキーマは、データベースファイルの構造の定義情報です。

HiBaseでは、このスキーマを簡単に作り出す機能としてHSchem::MakeSchemaを用意しています。

HSchema::MakeSchemaは、メモリもしくはファイルに、データベース記述文法に従って書かれているテキストから、スキーマ定義オブジェクト（HSchema：スキーマ情報を内部的に表現した構造体）を作り出します。

■データベースファイルを生成する

データベースファイルの生成は、HDBHandle::CreateDBFileを実行します。この時、上記のスキーマ定義オブジェクトをパラメータで渡します。

HDBHandle::CreateDBFileに渡すスキーマ定義オブジェクトは、普通は一つのデータベースファイルに関するスキーマ定義オブジェクトです。しかし、一つに限る必要はなく、複数のデータベースファイルを表現するものでも構いません。この場合、データベースファイルは複数が生成されます。

3-1-2-2. データベースファイルの削除

HiBase の「データベース」を生成する場合は、以下のような手順となります。

データベースファイルの削除（「HDBHandle :: EraseDBFile」）

```
HBSBoss* theBoss = new HBSBoss("../HiBase.conf");
HDBHandle* theHandle = new HDBHandle(theBoss);

ECode ec = theHandle->Open();
if (ec == ecNormal) {
 ec = theHandle->EraseDBFile(1);
 theHandle->Close();
}
delete theHandle;
delete theBoss;
```

【参照】

以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するプログラミングの基礎知識】 ⇒

◇ 第２章 - プログラミングの基礎知識; ***HiBase*** のデータベース構成

【以上の内容に関連するAPI の詳細】 ⇒ プログラムリファレンス（別冊）

◇ 第１章 - 基本API ; HDBHandle クラス_データベースの管理、
_データベースファイルの管理（DD/D 関連）

3-1-3. トランザクション処理

トランザクション処理（データの入出力）を行なう場合は、DML 関連 のメンバ関数を利用します。DML 関連のメンバ関数は、アプリケーションプログラムが管理すべきデータを、レコードの形で ***HiBase*** のデータベースに出し入れする機能を提供します。

アプリケーションプログラムが「トランザクション処理」を行なう場合、通常は、本APIと「スキーマ変換クラス（HRecord）」を組み合わせて利用します。

3-1-3-1. 一般的なレコード操作

アプリケーションプログラムは、一般的に以下のような手順で「トランザクション処理」を行ないます。

レコードの追加

アプリケーションプログラムがデータベースファイル内にレコードを追加する場合は、「スキーマ変換クラス（HRecord）」を使って「レコード編集クラス」を準備した後、「InsRecord 関数」を呼び出します。

具体的には、以下のような手順となります。

1. 「レコード編集クラス」の準備 （「HRecordクラス」）

   「スキーマ変換クラス（HRecord）」を利用し、追加レコードを編集します。

2. 「レコード」の追加 （「HDBHandle :: InsRecord」）

   「HDBHandle クラス」の「InsRecord 関数」を呼び出し、レコードを追加します。

   （この際、「スキーマ変換クラス（HRecord）」を利用して編集された「レコード」を指定します。）

【 レコードの追加例 】

```
HXRecord    theRec(hdl, file);                          // レコード編集クラスを準備
long        nbr = 1;
char        altem[32], bltem[32], cltem[32];
RNbr        rNbr;                                       // レコード番号
ECode       ec;                                         // エラーコード

theRec.Clear();                                         // レコードを初期化

theRec.Append("[1] %4i", &nbr);                         // 項目 1 を long integerで設定

sprintf(altem, "A%05ld", nbr);
theRec.Append("[2] %sn", altem );                       // 項目 2 を文字列（c 文字列）で設定

sprintf(bltem, "B%05ld", nbr);
theRec.Append("[3] %sn", bltem);                        // 項目 3 を文字列（c 文字列）で設定

sprintf(cltem, "C%05ld", nbr);
theRec.Append("[4] %sn", cltem);                        // 項目 3 を文字列（c 文字列）で設定

ec = hdl->InsRecord(file, rNbr, (DBRec*) theRec);       // 追加
```

データ検索（集合の作成）

HiBase は、レコードを構成している「項目」を「キー」に指定して検索を行い、検索の結果見つかった「レコード」群を「集合」として扱います。

「集合」は、***HiBase*** 内部に作られ、以降の読み出しや集合演算に利用されます。

「集合」を作成する場合は、以下のような手順となります。

1. 「検索条件式」の準備

「検索条件」は、文字列による簡潔な式で表現されます。

「条件式」は、検索に利用する 「項目 または キー名」もしくは、「項目 または キー番号」を 中括弧 （ [ ] ）で鋏み、その後ろに「イコール （＝)」を置き、その後ろに検索対象とする「値指定」を並べ、これをダブルクォーテーション（**" "**）で括って指定します。

「条件式」は、以下のような柔軟な記述が許されています。

◇　項目／キー名の指定

「条件式」の中に、データベースファイルの生成に定義した 「項目名 または キー名」を指定することができます。

【 項目名の指定例 】

◇　項目名「姓」の値が「鈴木」のレコードを検索する

→　**[姓] = "鈴木";**　と表現することができます。

◇　項目／キーの混在検索

「条件式」の中に、検索に利用する 「項目 または キー」の「識別子」を埋め込み、同一条件式内に「項目／キー」の検索を混在させることができます。

「キー」の「識別子」は「k:」、「項目」の「識別子」は「i:」 です。
（識別子が省略された場合は「k:（キー検索）」をデフォルトとします。）

なお、「基本 ***HiBase*** API（HDBHandle クラス）」では、「項目検索」、「キー検索」を「条件式」ではなく「関数」の使い分け（「HDBHandle:: SetMakeItm ／SetMakeKey」）で区別します。

【 項目／キー混在検索の指定例 】

◇　キー番号１（趣味）の値が「スポーツ」か「読書」で、項目番号1（性別）の値が「女性」の場合、

→　**[K:1] = "スポーツ ", "読書 " and [i:1] = " 女性";**

→　「項目／キー」の論理積（and）が検索されます。

◇　条件式に集合を利用する

「条件式」の中に、他の「集合」を加えることができます。

【 条件式に集合を利用した指定例 】

◇　キー番号１（趣味）の値が「スポーツ」で、これを既に作られている集合（仮に集合番号:1 とします）との AND をとる。

→　**[K:1] = "スポーツ " and #1 ;**

→　キー番号１の値が「スポーツ」のレコードを捜し、その結果と「集合番号1番= "#1" 」との論理積（and）が実行されます。

◇　前方／後方／中間／完全 一致 の指定

「条件式」の中に、「前方一致」、「後方一致」、「中間一致」、「完全一致」の「一致条件」や「範囲」などを指定することができます。

完全一致 ： =　値
前方一致 ： =　値 ?
後方一致 ： =　? 値
中間一致 ： =　? 値 ?
範囲　　： =　値 - 値
列挙　　： =　値 <, 値 <...>>

【一致条件の指定例】

◇　項目名「名前」の値が「鈴木」の場合、

>>「前方一致」の例：

「名前」が「鈴木」で始るレコードを検索する

→　**[名前] = "鈴木 "? ;**

>>「後方一致」の例：

「名前」が「鈴木」で終わるレコードを検索する

→　**[名前] = ? "鈴木 ";**

>>「中間一致」の例：

「名前」が「鈴木」を含むレコードを検索する

→　**[名前] = ? "鈴木 "?;**

>>「完全一致」の例：

「名前」が「鈴木」と一致するレコードを検索する

→　**[名前] = "鈴木 ";**

2．「集合」の生成、検索の実行　（「HDBHandle:: SetMake」）

HDBHandle::SetMakeを呼びだして、条件式を実行します。

```
HBSBoss* theBoss = new HBSBoss("../HiBase.conf");
HDBHandle* theDBHandle = new HDBHandle(theBoss);
ECode  ec = theHandle->Open();

long sid, rQty;

if (ec == ecNormal) {
 ec = theHandle->SetMake(fileNumber, sid, rQty, "[趣味]=\"スポーツ\";");
 if ((ec == ecNormal) && (rQty > 0)) {
   ..............
 // 前後の操作（読み出し、レコードの更新、etc）
   ..............
 }
 theHandle->Close();
}
delete theDBHandle;
delete theBoss;
```

レコードの読み出し

一般的に、アプリケーションプログラムがデータベースファイル内のレコードを読み出す場合は、「SetMake関数」を使って「集合」を作成し、「SetRGet 関数」で検索された集合内のレコードを読み出します。

読み出されたレコードは、通常「スキーマ変換クラス（HRecord)」を利用して編集します。

具体的には、以下のような手順となります。

1. データ検索（集合の作成）（「HDBHandle:: SetMake 関数」）

   「HDBHandle クラス」の「Setメンバ関数群(SetMakeKey 或いは、SetMakeItem 関数等)」を使って、「集合」を作成します。

2. 「集合」内の「レコード」の読み出し（「HDBHandle :: SetRGet」）

   「HDBHandle クラス」の「SetRGet 関数」を呼び出し、「集合」から、idx で示すインデックスの「レコード」を読み出します。

   見つかったレコードの件数分の処理（i = 1 ~ n ）を繰り返します。

3. 「レコード」の編集（「HRecordクラス」）

   「HRecord クラス」の「Fetch 関数」を呼び出し、項目データを得ます。

【 レコードの読み出し例 】

```
void ListRecord (HDBHandle* theHandle, FNbr theFNbr, SetID sid)
{
    JLong rQty;
    RNbr rNbr;
    HRecord theRec;

    ECode ec = theHandle->SetSize(theFNbr, sid, rQty);
    if ((ec == ecNormal) && (rQty > 0)){
//      {
//              ec = theHandle->SetSort(theFNbr, sid, "[1] %d;");
//      }
    for (JLong i = 0; (i < rQty) && (ec == ecNormal); i ++) {
        ec = theHandle->SetRGet(theFNbr, sid, i, rNbr, &theRec);
        if (ec == ecNormal) {
                JByte work[256];

                printf("[R:%5ld] ", rNbr);
                theRec.Fetch("[1] %nn;", work);
                printf("%s  ", work);
                theRec.Fetch("[2] %sn;", work);
                printf("%s  ", work);
                theRec.Fetch("[3] %sn;", work);
                printf("%s  ", work);
                theRec.Fetch("[4] %sn;", work);
                printf("%s\n", work);
        }
    }
}
```

レコードの削除

一般的に、アプリケーションプログラムがレコードの削除を行なう場合は、「Set メンバ関数群」を使って「集合」を作成後、「DelRecord 関数」を実行します。

具体的には、以下のような手順となります。

1. データ検索（集合の作成）（「HDBHandle:: SetMake 関数」）

   「HDBHandle クラス」の「SetMake」を使って、「集合」を作成します。

2. 「集合」内の「レコード」の読み出し （「HDBHandle :: SetRGet」）

   「HDBHandle クラス」の「SetRGet 関数」を呼び出し、「集合」から、idx で示すインデックスの「レコード」を読み出します。

   見つかったレコードの件数分の処理（ i = 1 ~ n ）を繰り返します。

3. 「レコード」の削除（「HDBHandle :: DelRecord」）

   「HDBHandle クラス」の「DelRecord 関数」を呼び出し、「レコード」を削除します。

   （この際、削除する「レコード番号」を指定します。

レコードの更新

アプリケーションプログラムがレコードの更新を行なう場合は、「Setメンバ関数群」を使ってデータ検索を行ない、「スキーマ変換クラス（HRecord)」でレコードに編集を加えた後、「UpdRecord 関数」を実行します。

具体的には、以下のような手順となります。

1. データ検索（集合の作成）（「HDBHandle:: SetMake」）

   「HDBHandle クラス」の「SetMake」を使って、「集合」を作成します。

2. 「集合」内の「レコード」の読み出し（「HDBHandle :: SetRGet」）

   「HDBHandle クラス」の「SetRGet 関数」を呼び出し、「集合」から、idx で示すインデックスの「レコード」を読み出します。

   見つかったレコードの件数分の処理（i = 1 ~ n ）を繰り返します。

3. 「レコード」の編集（「HRecordクラス」）

   「スキーマ変換クラス（HRecord)」を利用し、更新する「レコード」を編集します。

4. 「レコード」の更新（「HDBHandle :: UpdRecord」）

   「HDBHandle クラス」の「UpdRecord 関数」を呼び出し、「レコード」を更新します。

   （この際、「スキーマ変換クラス（HRecord)」を利用して編集された「レコード」を指定します。）

【レコードの更新例】

```
long            rQty, i;
RNbr            rNbr;

//
//      集合は既に作られて、その識別子が sid に入っているものとする
//      また、データベースハンドルが hdl に、
//      ファイル番号が file に入っているものとする
//

//      集合のサイズを知る
ec = hdl->SetSize(file, sid, &rQty);
if ((ec == ecNormal) && (rQty > 0)) {
        HXRecord        theRec(hdl, file);
        for (i = 0; (i < rQty) && (ec == ecNormal); i ++) {

                //  i 番目のレコード番号を得る
                ec = hdl->SetRGet(file, sid, i, &rNbr, (DBRec*) theRec);
                if (ec == ecNormal) {

                        //  項目#2をHiBase Databaseに変更
                        theRec.Update("[2] %sn;", "HiBase Database");
                        //      delete
                        ec = hdl->UpdRecord(file, rNbr, (DBRec*) theRec);
                }
        }
        hdl->SetCancel(sid);
}
```

【参照】

以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するプログラミングの基礎知識】 ⇒

◇ 第２章 - プログラミングの基礎知識 ; ***HiBase*** でのトランザクション処理

【以上の内容に関連するAPIの詳細】 ⇒ プログラムリファレンス（別冊）

◇ 第１章 - 基本API ; HDBHandle クラス
_トランザクション処理（DML 関連）
> レコード操作、データ検索

3-1-3-2. シーケンシャルアクセス

HiBase は、全データを対象とする「トランザクション処理」を想定し、「キーシーケンシャルアクセス」と「高速シーケンシャルアクセス」機能を用意しています。

アプリケーションプログラムが、データベース内の「全レコード」を対象に、「シーケンシャル処理」を行なう場合は、「HDBHandle:: KSメンバ関数群／PSメンバ関数群」を利用します。

キーシーケンシャルアクセス

１つの「キー」の値の順番に「レコード」を順次読み出す場合は、「HDBHandle:: KSメンバ関数群」を呼び出し、「キーシーケンシャルアクセス」を実行します。

具体的には、以下のような手順となります。

１．「読み出し条件」の設定 （「HDBHandle :: KSLocate」）

「HDBHandle クラス」の「KSLocate 関数」を呼び出し、読み出しの条件（読み出すキー番号／読み出し開始値）を指定します。

２．「キーシーケンシャルアクセス」の実行 （「HDBHandle :: KSRecord」）

「HDBHandle クラス」の「KSRead関数」を呼び出し、「キーシーケンシャルアクセス」を実行します。

終了条件を満たすまで「KSRead関数」を繰り返し呼び出すことで、該当レコードから成る「集合」が作られ、「キーの値」、その値を持つ「キーの個数」及び、「レコード番号リスト」が読み出されます。

３． アクセスの終了 （「HDBHandle :: KSCancel」）

終了条件を満たしたら、「HDBHandle クラス」の「KSCancel 関数」を呼び出し、「キーシーケンシャルアクセス」を終了します。

４．「レコード」の読み出し （「HDBHandle :: GetRecord」）

「HDBHandle クラス」の「GetRecord 関数」を呼び出し、「レコード」を読み出します。

見つかったレコードの件数分の処理（ｉ＝１~ ｎ）を繰り返します。

５．「レコード」の編集 （「HRecordクラス」）

「HRecord クラス」を利用し、「レコード」を編集します。

【キーシーケンシャルアクセス例】

```
struct {
        long     nbr;              // item #1
        char     altem[32];        // item #2
        char     bItem[32];        // item #3
        char     cItem[32];        // item #4
} theData;

SetID   sid;
short   rQty, i;
RNbr    rList[500];

// 読み出し開始位置をキーの値（[2]="A00001"）で設定
ec = hdl->KSLocate(file, &sid, 2, "\pA00001");
while (ec == ecNormal) {
        HXRecord        theRec(hdl, file);

        // 次の値とその値を持つレコード件数およびレコード番号リストを読み出す
        ec = hdl->KSRead(file, sid, pVal, &rQty, rList);
        if (ec == ecNormal) {
                for (JWord i = 0; i < rQty; i ++) {
                        ec = hdl->GetRecord(file, rList[i], (DBRec*) theRec);
                        if (ec == ecNormal) {
                                theRec.Fetch("[1] %4i, [2]-[4] %32sn;", &theRec);
                                ..............................
                        }
                }
        }
        hdl->KSCancel(sid);
}
```

高速シーケンシャルアクセス

データベース内の「全レコード」を、格納されている順番に高速に読み出す場合は、「HDBHandle:: PSメンバ関数群」を呼び出し、「高速シーケンシャルアクセス」を実行します。

具体的には、以下のような手順となります。

１．「読み出し開始」の宣言 （「HDBHandle :: PSLocate」）

「HDBHandle クラス」の「PSLocate 関数」を呼び出し、読み出しの開始宣言を行ないます。

２．「高速シーケンシャルアクセス」の実行 （「HDBHandle :: PSRecord」）

「HDBHandle クラス」の「PSRead関数」を呼び出し、「高速シーケンシャルアクセス」を実行します。

「PSRead関数」を繰り返し呼び出すことで、データベース内の「レコード」が、格納順に、順次読み出されます。

４． アクセスの終了 （「HDBHandle :: PSCancel」）

終了条件を満たしたら、「HDBHandle クラス」の「PSCancel 関数」を呼び出し、「高速シーケンシャルアクセス」を終了します。

５．「レコード」の編集 （「HRecordクラス」）

「HRecord クラス」を利用し、「レコード」を編集します。

【 高速シーケンシャルアクセス例 】

```
struct {
        long    nbr;            // item #1
        char    altem[32];      // item #2
        char    bltem[32];      // item #3
        char    cltem[32];      // item #4
} theData;

SetID   sid;
RNbr    rNbr;

// 読み出しの宣言
ec = hdl->PSLocate(file, &sid);
while (ec == ecNormal) {
        HXRecord        theRec(hdl, file);
        // 次の値とその値を持つレコード件数およびレコード番号リストを読み出す
        ec = hdl->PSRead( file, sid, &rNbr, (DBRec*) theRec);
        if (ec == ecNormal) {
                theRec.Fetch( "[1] %4i, [2]-[4] %32sn;", &theRec);
                ..........................
        }
        hdl->PSCancel(sid);
}
```

【参照】

以上の内容に関連するドキュメントを参照する場合は、以下を参考にしてください。

【以上の内容に関連するプログラミングの基礎知識】 ⇒

◇ 第２章 - プログラミングの基礎知識; ***HiBase*** でのトランザクション処理
> シーケンシャルアクセス機能

【以上の内容に関連するAPIの詳細】 ⇒ プログラムリファレンス（別冊）

◇ 第１章 - 基本API ; HDBHandle クラス
_トランザクション処理（DML 関連）> シーケンシャル処理

HiBase 関連ドキュメント & サンプルプログラム一覧

HiBase のアプリケーションプログラム開発 に関連するドキュメント、及び、
サンプルプログラムを参照する場合は、以下を参考にしてください。

◇ ***HiBase*** アプリケーションプログラミングの基礎知識

プログラミング・ガイド　（PrGuide.pdf）

Documentsフォルダ内

◇ ***HiBase*** APIの詳細（設計とその使用法）

プログラム・リファレンス　（PrRefs.pdf）

Documentsフォルダ内

◇ ***HiBase*** APIを利用したサンプルプログラム

PDBDefine.Prj, DBLoad.Prj, DBSearch.Prj, DBDelete.Prj,
DBUnload.Prj, DBTest.Prj　Develop:: Samplesフォルダ内

◇ ***HiBase*** API for Java （設計とその使用法）

プログラム・リファレンス for Java》》》（PrRefs(Java).pdf）

Documentsフォルダ内

◇ ***HiBase*** 基本アプリケーション＆インタフェースプログラムインストール

スタートアップ・マニュアル（StartUp.pdf）

Documentsフォルダ内

◇ ***HiBase*** の運用（基本アプリケーション操作）

オペレーション・マニュアル（Operation.pdf）

Documentsフォルダ内

◇ CGI 開発の支援

CGI 開発支援解説 for HCGI & HACGI》》》（HCGI&HACGI.pdf）

Documentsフォルダ内

Credit（Programming Guide）

Writing :　Yumiko Itoh

HiBase プログラミング・ガイド

発行日 ：　１９９９年10月 １日

発　行 ：　ホロン株式会社
