○　Hamlog50.DLL の仕様書（Ver5.03 以上に付属のものが対象）

　Turbo HAMLOG/Winのデータベースエンジン Hamlog50.DLL の仕様書です。
このＤＬＬを使って、Turbo HAMLOG用の Windows版アワード用ソフト等を作ってくださる方がいれば嬉しい限りです。

＊　Hamlog50.DLL の使用条件

１．Turbo HAMLOGのデータ類をアクセスするソフトでのみ使用すること。他の用途に使うことを拒否します。
２．フリーソフトで使用してください。金品を要求するソフトで使用することを拒否します。
３．Hamlog50.DLLを使っている旨を明記すること。

＊ Hamlog50.DLL内の関数について

　Delphiの宣言部分については、Turbo HAMLOG/Win本体が Delphi 3.1Jで書いたため問題ないのですが、Visual BASIC 4.0の宣言部分については、すべてにおいてテストしたわけではないので、あまり自信がありません。(^_^;)
　Visual BASICで使う場合は、Hamlog50.DLL と THDLL2VB.DLL を ?:¥Windows¥System フォルダにコピーして下さい。ＶＢで作成したアプリと同じフォルダでも動作すると思います。

　Hamlog50.DLL は Borland C++ 5.0Jで書いてあります。インポートライブラリについて、Borland と Microsoftでは仕様が違うようです。もし、Visual C++等からの利用でインポートライブラリがうまくリンクできない場合は、Win32-APIのLoadLibrary関数とGetProcAddress関数を使ってみて下さい。
　奥の手としては、Hamlog50.DLLの APIと同じプロトタイプで、何もしない同名の関数を持ったDLLを Visual C++で作成し、そのとき生成された LIBファイルをリンクするのがいいかもしれません。必要とする関数のプロトタイプのみで大丈夫だと思います。

　関数の戻り値で「成功した場合 SUCCESSを返す」と書いてあるものは、失敗した場合は、それ以外の値を返します。これらは DelphiとVB4では const節で宣言しているエラー番号を、C/C++ではヘッダファイル内で #define定義しているエラー番号を返します。
　まぁ、戻り値が SUCCESSか、そうでないかで判断すれば問題ないでしょう。

　関数のプロトタイプは、それぞれ Delphi 2.0J以上・Visual BASIC 4.0・Ｃ言語の順で記述しました。Delphi の Integer、及びC/C++の int は３２ビットですが、Visual BASIC のIntegerは１６ビットです。そのため、VB4用はLongで宣言してあります。

　ＱＳＯデータをアクセスするのであれば、プログラム起動時にHamlogOpen()でファイルを一括して開き、THW_????()で読み・書き・検索をして、プログラム終了時にHamlogClose()でファイルを一括して閉じるのがいいでしょう。

```
function GetThdllVersion: Integer;
Function GetThdllVersion() As Long
int WINAPI GetThdllVersion(void);
```

概　要：ＤＬＬのバージョンを返す。古いＤＬＬを使っている場合に備え、起動時にバージョンチェックを行うべき。
引　数：なし
戻り値：バージョン番号を１６進数で返す。Ver5.03では &H50300

```
[例]    if (GetThdllVersion() < 0x50300) // ﾊﾞｰｼﾞｮﾝが古い
            exit();
```

```
function dbf_write2(var db: TDBFh; rno: LongInt; fbuf: PChar; fst, cnt:
Integer): Integer;
```

```
Function dbf_write2(d As TDBFh, ByVal n As Long, ByVal s As String, ByVal l As
Long, ByVal i As Long) As Long
int WINAPI dbf_write2(TDBFh *, long, const void *, const int, const int);
概　要：dBASE互換データファイルへの書き込み
        （指定レコードの fstﾊﾞｲﾄ目から cntﾊﾞｲﾄ書き込む）
        連続したレコードにまとめて書き込むことも可能。
引　数：db      dBASE互換データアクセス構造体
        rno     書き込むレコード番号
        fbuf    書き込むバッファ
        fst     指定レコードの書き込む領域の先頭。０バイト目は削除マークを書き込
む領域。従って通常は１から。
        cnt     書き込むバイト数
戻り値：成功の場合は SUCCESS を返す
[例]    // １０レコードまとめて空白を書き込む　Ｃ言語
        char    buff[1025];
        FillMemory(buff, 1024, (Byte)' ');
        dbf_write2(&dbf, rno, buff, 0, dbf_rsize(&dbf) * 10);

--------------------------------------------------------------------------------
function dbf_write(var db: TDBFh; rno: LongInt; fbuf: Pointer): Integer;
Function dbf_write(d As TDBFh, ByVal n As Long, ByVal s As String) As Long
int  WINAPI dbf_write(TDBFh *, long, void *);
概　要：dBASE互換データへ１レコード書き込む
引　数：db      dBASE互換データアクセス構造体
        rno     書き込むレコード番号
        fbuf    書き込むバッファ
戻り値：成功の場合は SUCCESS を返す

--------------------------------------------------------------------------------
function dbf_create(fname: PChar; n: Pointer): Integer;
Function dbf_create(ByVal s As String, ByVal n As Long) As Long
int  WINAPI dbf_create(const char *, void *);
概　要：HAMLOG.HDBを新規作成する。既に存在すれば上書きし、できあがったデータの件
数はゼロである。
引　数：fname   ファイル名をフルパスで指定する
        n       Dephiではnil、VBでは0、CではNULLを指定
戻り値：成功の場合は SUCCESS を返す

--------------------------------------------------------------------------------
function dbf_open(fname: PChar; var db: TDBFh): Integer;
Function dbf_open(ByVal s As String, d As TDBFh) As Long
int  WINAPI dbf_open(const char *, TDBFh *);
概　要：HAMLOG.HDBや、HAMLOG.MSTをオープンする。
        dBASE互換データファイルがオープン可能。
引　数：fname   ファイル名をフルパスで指定する
        db      dBASE互換データアクセス構造体
戻り値：成功の場合は SUCCESS を返す

--------------------------------------------------------------------------------
procedure dbf_close(var db: TDBFh);
Sub dbf_close(d As TDBFh)
void WINAPI dbf_close(TDBFh *);
概　要：HAMLOG.HDBや、HAMLOG.MSTをクローズする。
引　数：db      dBASE互換データアクセス構造体

--------------------------------------------------------------------------------
function dbf_read(var db: TDBFh; rno: LongInt; var fbuf): Integer;
Function dbf_read(d As TDBFh, ByVal n As Long, ByVal s As String) As Long
```

```
int  WINAPI dbf_read(TDBFh *, long, void *);
概　要：dBASE互換データファイルからバッファに１レコード読み込む。
引　数：db      dBASE互換データアクセス構造体
        rno     読み込むレコード番号
        fbuf    読み込むバッファ
戻り値：成功の場合は SUCCESS を返す

--------------------------------------------------------------------------------
function dbf_rsize(var db: TDBFh): Integer;
Function dbf_rsize(d As TDBFh) As Long
int  WINAPI dbf_rsize(TDBFh *);
概　要：dBASE互換データのレコードサイズ（バイト数）を返す。
引　数：db      dBASE互換データアクセス構造体
戻り値：データレコードサイズを返す

--------------------------------------------------------------------------------
function DB_append(var Th: TThLog; const rno: Integer): Integer;
Function DB_append(Th As TThLog, ByVal n As Long) As Long
int  WINAPI DB_append(TThLog *, const int);
概　要：ＱＳＯデータを追加／修正する。但し、インデックスファイルは更新されない。
        あらかじめ Th.Qso の全項目にデータがコピーされていること。
        また、Th.Qsoの各メンバーは、NULLで終わる文字列で満たしておくこと。
引　数：Th      Turbo HAMLOGデータアクセス構造体
        rno     レコード番号。１～dbf_rcount()の範囲を指定した場合は、修正。
                0や-1を指定した場合は、追加します。
戻り値：成功の場合は SUCCESS を返す

--------------------------------------------------------------------------------
function THW_read(var Th: TThLog; rno: Longint; sepa: Integer): Integer;
Function THW_read(h As TThLog, ByVal n As Long, ByVal i As Long) As Long
int  WINAPI THW_read(TThLog *, const long, const int);
概　要：レコード番号を指定して、ＱＳＯデータを読み込む。
        読み込まれた内容は、たとえばコールサインは Th.Qso.Callsに、ＱＴＨは、
Th.Qso.Qth に入っている
引　数：Th      Turbo HAMLOGデータアクセス構造体
        rno     レコード番号
        sepa    読み出し形式のフラグ
        (sepa & 4) カレントレコードナンバーをセットしない
        (sepa & 8) calls, date, time, code, glid, qslのみ読み込む
戻り値：成功の場合は SUCCESS を返す

--------------------------------------------------------------------------------
function THW_readv(var Th: TThLog; rno: Longint; var Qso: TQsoBuff): Integer;
Function THW_readv(h As TThLog, ByVal n As Long, ByVal q As TQsoBuff) As Long
int  WINAPI THW_readv(TThLog *, const long, TQsoBuff *);
概　要：レコード番号を指定して、別に用意したバッファにＱＳＯデータを読み込む。
        読み込まれた内容は、たとえばコールサインは Qso.Callsに、ＱＴＨは、
Qso.Qth に入っている
引　数：Th      Turbo HAMLOGデータアクセス構造体
        rno     レコード番号
        Qso     データを読み込むバッファ
戻り値：成功の場合は SUCCESS を返す

--------------------------------------------------------------------------------
function THW_skip(var Th: TThLog; const num, flg: Integer): Integer;
Function THW_skip(h As TThLog, ByVal n As Long, ByVal i As Long) As Long
int  WINAPI THW_skip(TThLog *, const int, const int);
概　要：現在のキーの位置から、num個データ位置をスキップした位置のデータを読み
```

込む。キーポインタ、またはレコードポインタはスキップした位置に移動。
引　数：Th　　Turbo HAMLOGデータアクセス構造体
　　num　　スキップする数（100～1 または -1～-100）
　　flg　　DbsNoNDX でレコード番号順（入力順）
　　DbsCallDX でコールサイン順
　　DbsCodeDX でＪＣＣ／Ｇコード順
戻り値：成功の場合は SUCCESS を返す

```
function THW_seek(var Th: TThLog; Key: PChar; flags: Integer): Integer;
Function THW_seek(h As TThLog, ByVal s As String, ByVal i As Long) As Long
int  WINAPI THW_seek(TThLog *, char *, const int);
```

概　要：データを高速に検索し、該当データを読み込む。
　　キーポインタは検索位置に移動する。
引　数：Th　　Turbo HAMLOGデータアクセス構造体
　　key　　検索するキー
　　"J" -------- Ｊから始まるｺｰﾙｻｲﾝを検索
　　"JG" ------- ＪＧから始まるｺｰﾙｻｲﾝを検索
　　"JG1" ------ ＪＧ１から始まるｺｰﾙｻｲﾝを検索
　　"JG1M" ----- ＪＧ１Ｍから始まるｺｰﾙｻｲﾝを検索
　　"JG1MO" ---- 以下同様
　　"JG1MOU" ---
　　flags　DbsCallDX でコールサインによる検索
　　DbsCodeDX でＪＣＣ／Ｇコードによる検索
　　※DUP_CHECK を足して指定すれば重複チェックも行う。この場合、
　　検索するキーが二つ以上あれば、key[0] = Chr(0) となって戻る。
　　従ってこの場合は、key は変更されても構わない変数を使うこと。
　　これにより、「該当データが複数ある」と判断できる。
戻り値：該当データがあった場合、SUCCESS を返す。該当データがない場合は、NDXEOF
　　を返す。

```
function THW_top(var Th: TThLog; flags: Integer): Integer;
Function THW_top(h As TThLog, ByVal i As Long) As Long
int  WINAPI THW_top(TThLog *, const int);
```

概　要：先頭のデータを読み込む。
引　数：Th　　Turbo HAMLOGデータアクセス構造体
　　flags　DbsNoNDX でレコード番号順（入力順）の先頭
　　DbsCallDX でコールサイン順の先頭
　　DbsCodeDX でＪＣＣ／Ｇコード順の先頭
戻り値：成功の場合は SUCCESS を返す

```
function THW_btm(var Th: TThLog; flags: Integer): Integer;
Function THW_btm(h As TThLog, ByVal i As Long) As Long
int  WINAPI THW_btm(TThLog *, const int);
```

概　要：最後のデータを読み込む
引　数：Th　　Turbo HAMLOGデータアクセス構造体
　　flags　DbsNoNDX でレコード番号順（入力順）の最後
　　DbsCallDX でコールサイン順
　　DbsCodeDX でＪＣＣ／Ｇコード順
戻り値：成功の場合は SUCCESS を返す

```
function dbf_delete(var dbf: TDBFh; const rno: LongInt): Integer;
Function dbf_delete(d As TDBFh, ByVal r As Long) As Long
int  WINAPI dbf_delete(TDBFh *, const long);
```

概　要：rno で指定したレコードに削除マークをつける。

引　数：dbf　　dBASE互換データアクセスの構造体
　　rno　　削除マークを付けるレコード番号
戻り値：成功の場合は SUCCESS を返す

```
function dbf_packwk(var dbf: TDBFh; buff: PChar; size: Integer): Integer;
Function dbf_packwk(d As TDBFh, ByVal b As String, ByVal s As Long) As Long
int  WINAPI dbf_packwk(TDBFh *, char *, int);
```

概　要：def_delete()で削除マークを付けたレコードをまとめて物理的に削除する。
引　数：dbf　　dBASE互換データアクセス構造体
　　buff　　ワークエリア（バッファ）へのポインタ。
　　size　　ワークエリアのサイズ（バイト数）。バッファは、HDDのクラスタサイズの２倍以上用意してください。
戻り値：成功の場合は SUCCESS を返す

```
function THW_update(var Th, Th2: TThLog; const rno: Longint;
                                const flg: Integer; var mes: Integer): Integer;
Function THW_update(Th As TThLog, Th2 As TThLog, ByVal r As Long, ByVal f As
Long, ByVal m As Long) As Long
int  WINAPI THW_update(TThLog *, TThLog *, const long, const int, int *);
```

概　要：rno で指定したレコード番号のデータの内容を書き換える。
　　データを書き換えると自動的にインデックスファイルも書き換えられますが、マスターデータの場合は書き変わりません。
　　この場合、後でインデックスの再構築をして下さい。
引　数：Th　　Turbo HAMLOGデータアクセス構造体。あらかじめ Th.Qsoの全メンバーに変更した内容をコピーしておくこと。
　　DLL内部で Th2.Qsoと比較し、変更があれば「データを登録してよろしいですか？」と確認のMessageBox が表示される。
　　Th2　　比較用のTurbo HAMLOGデータアクセス構造体。あらかじめ変更前のThの内容をコピーしておく。
　　Move(Th, Th2, Sizeof(Th2));
　　rno　　書き換えるデータのレコード番号
　　flg　　IsQSOdata でＱＳＯデータ（HAMLOG.HDB）
　　IsMASdata でマスターデータ（HAMLOG.MST）が対象となる。
　　mes　　変数の参照渡しで、この関数からリターンした後「データを登録してよろしいですか？」の確認 MessageBox の戻り値が入っている。
　　つまり、IDYES, IDNO 等の値が入っている。
　　モーダルウインドウから呼び出す場合は、予め変数に１を代入しておく。通常は０を代入しておく。
　　関数呼び出し前に KAKUNIN_NO を足して代入しておくと、確認メッセージ無しで実行します。（mes = KAKUNIN_NO + 1; ）

戻り値：登録しなかった場合、または登録成功の場合は SUCCESS を返す。

```
function THW_append(var Th: TThLog; const flg: Integer; var mes: Integer):
Integer;
Function THW_append(Th As TThLog, ByVal f As Long, m As Long) As Long
int  WINAPI THW_append(TThLog *, const int, int *);
```

概　要：データを１件追加する。。
引　数：Th　　Turbo HAMLOGデータアクセス構造体。あらかじめ Th.Qsoの全メンバーに登録したい内容をコピーしておく。
　　Th.Qso内の各メンバーは、ヌルターミネイト文字列で完全に満たしておくこと。
　　※"埼玉県久喜市"でも、"埼玉県久喜市　　"でも、結果は同じ。
　　flg　　IsQSOdata でＱＳＯデータ（HAMLOG.HDB）
　　IsMASdata でマスターデータ（HAMLOG.MST）

　　mes　　変数の参照渡しである。あらかじめmesにゼロ以外を代入しておくと、ディスク登録前に「データを登録してよろしいですか？」の確認MessageBox が表示され、関数からリターンしたときにはMessageBoxの戻り値が入っている。
　　あらかじめmesにゼロを代入しておくと、確認メッセージが無く登録する。
戻り値：登録しなかった場合、または登録成功の場合は SUCCESS を返す。

```
procedure THW_flush(var Th: TThLog);
Sub THW_flush(Th As TThLog)
void WINAPI THW_flush(TThLog *);
```

概　要：ディスクバッファを書き込む。
　　ＱＳＯデータを、クローズすることなくディスクに書き込む。
引　数：Th　　Turbo HAMLOGデータアクセス構造体

```
function Idx_ReadKey(var Th: TThLog; const flg: Integer): PChar;
char * WINAPI Idx_ReadKey(TThLog *, const int);
```

概　要：インデックスファイルの、現在のキーを読み出す
引　数：Th　　Turbo HAMLOGデータアクセス構造体
　　flags　DbsCallDX でＱＳＯデータ、コールサイン順（*.DX0）
　　DbsCodeDX でＱＳＯデータ、ＪＣＣ／Ｇコード順（*.DX1）
　　MasCodeDX でマスターデータ、コード順（*.DX2）
　　MasFCodeDXでマスターデータ、フラグ＋コード順（*.DX3）
　　MasFHedEDXでマスターデータ、フラグ＋頭文字＋エリア順（*.DX4）
戻り値：インデックスキー文字列がコピーされた、ＤＬＬ内部の静的領域へのポインタを返す。
　　Delphiでは、戻り値がそのままString型文字列として使える。
※ＶＢ４では、Vidx_ReadKeyを使う。

```
function idx_open(const fname: PChar; var idx: TIDXh): Integer;
Function idx_open(ByVal s As String, p As TIDXh) As Long
int  WINAPI idx_open(const char *, TIDXh *);
```

概　要：インデックスファイルをオープンする。
引　数：fname　インデックスのファイル名
　　idx　　dBASE互換インデックスファイルの構造体
戻り値：成功の場合は SUCCESS を返す

```
procedure idx_close(var idx: TIDXh);
Sub idx_close(p As TIDXh)
void WINAPI idx_close(TIDXh *);
```

概　要：インデックスファイルをクローズする。
引　数：idx　　dBASE互換インデックスファイルの構造体

```
function idx_search(var idx: TIDXh; const key: PChar; len: Integer): LongInt;
Function idx_search(p As TIDXh, ByVal s As String, ByVal l As Long) As Long
long WINAPI idx_search(TIDXh *, const char *, int);
```

概　要：インデックスファイルの中の該当するキーを高速に検索し、該当するデータのレコード番号を返す。
　　THW_seek()の内部ではこの関数とdbf_read()を呼び出している。
引　数：idx　　dBASE互換インデックスファイルの構造体
　　key　　検索するキー。THW_seek()と同様。
　　len　　検索するキーの長さを指定する。コールサインの文字列が全部一致するキーを検索する場合は６。
　　プリフィックスだけ一致の場合は３。ゼロの場合は、key が通常のヌル

ターミネート文字列であるとして検索する。
戻り値：検索成功の場合は、そのデータのレコード番号を返す。検索できなかった場合は、ゼロを返す。

```
function idx_next(var idx: TIDXh): LongInt;
Function idx_next(p As TIDXh) As Long
long WINAPI idx_next(TIDXh *);
```

概　要：インデックスファイルの中の次のキーにポインタを合わせ、該当するデータのレコード番号を返す。
　　THW_skip()の内部ではこの関数とdbf_read()を呼び出している。
引　数：idx　　dBASE互換インデックスファイルの構造体
戻り値：インデックス順の、次のレコード番号を返す。次のキーがない（ファイルの終端）場合は、ゼロを返す。

```
function idx_back(var idx: TIDXh): LongInt;
Function idx_back(p As TIDXh) As Long
long WINAPI idx_back(TIDXh *);
```

概　要：インデックスファイルの中のひとつ前のキーにポインタを合わせ、該当するデータのレコード番号を返す。
引　数：idx　　dBASE互換インデックスファイルの構造体
戻り値：インデックス順の、ひとつ前のレコード番号を返す。これより前にキーがない（ファイルの先頭）場合はゼロを返す。

```
function idx_top(var idx: TIDXh): LongInt;
Function idx_top(p As TIDXh) As Long
long WINAPI idx_top(TIDXh *);
```

概　要：インデックスファイルの中の先頭のキーにポインタを合わせ、該当するデータのレコード番号を返す。
　　THW_top()の内部ではこの関数と dbf_read()を呼び出している。
引　数：idx　　dBASE互換インデックスファイルの構造体
戻り値：インデックス順の、先頭のレコード番号を返す。キーがない場合はゼロを返す。

```
function idx_bottom(var idx: TIDXh): LongInt;
Function idx_bottom(p As TIDXh) As Long
long WINAPI idx_bottom(TIDXh *);
```

概　要：インデックスファイルの中の最後のキーにポインタを合わせ、該当するデータのレコード番号を返す。
　　THW_btm()の内部ではこの関数と dbf_read()を呼び出している。
引　数：TIDXh *ip　　dBASE互換インデックスファイルの構造体
戻り値：インデックス順の、最後のレコード番号を返す。キーがない場合はゼロを返す。

```
procedure QSL_Rcv(var Th: TThLog; rno: Longint; const mark: Char);
Sub QSL_Rcv(h As TThLog, ByVal n As Long, ByVal k As Byte)
void WINAPI QSL_Rcv(TThLog *, const long, const char);
```

概　要：指定したレコード番号のＱＳＯデータに，ＱＳＬ受領マークを書き込む
引　数：Th　　Turbo HAMLOGデータアクセス構造体
　　rno　　レコード番号
　　mark　　ＱＳＬ受領マークのキャラクタ
[例]　QSL_Rcv(Th, 2000, '*');　// Pascal

```
procedure QSL_Send(var Th: TThLog; rno: Longint; const mark: Char);
Sub QSL_Send(h As TThLog, ByVal n As Long, ByVal k As Byte)
```

```
void WINAPI QSL_Send(TThLog *, const long, const char);
概　要：指定したレコード番号のＱＳＯデータに，ＱＳＬ発送マークを書き込む
引　数：Th      Turbo HAMLOGデータアクセス構造体
        rno     レコード番号
        mark    ＱＳＬ発送マークのキャラクタ

--------------------------------------------------------------------------------
function StrToLong(const str: PChar): LongInt;
Function StrToLong(ByVal s As String) As Long
long WINAPI StrToLong(const char *);
概　要：文字列を数値に変換する。数値に変換できない文字がある位置までの文字列が変
換対象となる。
        ＤＬＬ内部ではＣのライブラリ関数 atol()と同じことをしているだけです。
引　数：str     ヌルターミネート文字列。数字でない文字列の場合、ゼロを返す。
                また、数字でない文字の直前までの数値を返す。
戻り値：３２ビットの数値を返す。
[例]    num := StrToLong('23 589');  // ２３を返す Pascal

--------------------------------------------------------------------------------
function HamlogOpen(Func: Pointer; var Th: TThLog; const fname: PChar;
                const isComp: Integer): Integer;
Function HamlogOpen(ByVal Finc As Long, Th As TThLog, ByVal f As String, ByVal i
As Long) As Long
int  WINAPI HamlogOpen(int WINAPI (* Func)(char *),TThLog *, const char *, const
int);
概　要：ＱＳＯデータ（HAMLOG.HDB）、マスターデータ（HAMLOG.MST）をまとめてオープ
ンする。
        Turbo HAMLOG付属の HAMLOG.MST が必要。
        インデックスファイルは、存在しなければ新たに生成される。
引　数：Func    HAMLOGで使用します。Ｃ言語ならNULL、Delphiならnilとしてくださ
い。
                ＶＢならゼロかな？
        Th      Turbo HAMLOGデータアクセス構造体
        fname   データファイルをフルパスで指定する。拡張子は付けなくてもよい。
        isComp  0==タイムスタンプを比較しない。
                1==データのタイムスタンプが*.FDTに書いてある数字と一致しない場
                   合、または *.FDTが存在しない場合はインデックスを再構築する。
戻り値：成功の場合は SUCCESS を返す。ファイルがオープンできない場合は、次の値
        を返す。        LDBF_NOPEN      HAMLOG.HDBがオープンできない
                        MDBF_NOPEN      HAMLOG.MSTがオープンできない

--------------------------------------------------------------------------------
procedure HamlogClose(var Th: TThLog; const isComp: Integer);
Sub HamlogClose(Th As TThLog, ByVal i As Long)
void WINAPI HamlogClose(TThLog *, const int);
概　要：上記ファイルをまとめてクローズする
引　数：Th      Turbo HAMLOGデータアクセス構造体
        isComp  0==データのタイムスタンプを *.FDTに書き込まない。
                1==データのタイムスタンプを *.FDTに書き込む。

--------------------------------------------------------------------------------
function THW_idxrmv(var Th: TThLog; const rno: Longint;
                const flags: Integer; const key: PChar): Integer;
Function THW_idxrmv(h As TThLog, ByVal n As Long, ByVal i As Long, ByVal s As
String) As Long
int  WINAPI THW_idxrmv(TThLog *, const long, const int, const char *);
概　要：レコード番号とキーを指定して、データの位置とインデックスのキーポインタを
合わせる。
```

```
引　数：Th      Turbo HAMLOGデータアクセス構造体
        rno     レコード番号を指定
        flags   DbsCallDX でＱＳＯデータ、コールサイン順（*.DX0）
                DbsCodeDX でＱＳＯデータ、ＪＣＣ／Ｇコード順（*.DX1）
                MasCodeDX でマスターデータ、コード順（*.DX2）
                MasFCodeDXでマスターデータ、フラグ＋コード順（*.DX3）
                MasFHedEDXでマスターデータ、フラグ＋頭文字＋エリア順（*.DX4）
        kay     キーを指定する
戻り値：成功の場合は SUCCESS を返す
```

```
function FreqPCheck(const freq: PChar): Integer;
Function FreqPCheck(ByVal s As String) As Long
int  WINAPI FreqPCheck(const char *);
概　要：周波数に応じた整数を返す。
引　数：freq    周波数の文字列
戻り値：次の、0～15の数値を返す
enum    freq_tag        {
        X19MHZ,         //      0
        X35MHZ,         //      1   3.5MHz～3.8MHz
        X7_MHZ,         //      2
        X10MHZ,         //      3
        X14MHZ,         //      4
        X18MHZ,         //      5
        X21MHZ,         //      6
        X24MHZ,         //      7
        X28MHZ,         //      8   28MHz ～ 29MHz
        X50MHZ,         //      9
        X144MHZ,        //      10  144, 145MHz
        X430MHZ,        //      11
        X120MHZ,        //      12
        X240MHZ,        //      13
        X560MHZ,        //      14  5600MHz
        MAXBAND,        //      15  認識できない周波数
        SATELLITE       //      16 サテライト
};
        [例]    // Ｃ言語
        lstrcpy(Th.Qso.Freq, "29.220");
        FreqPCheck(Th.Qso.Freq) で 8を返す
```

```
procedure Wget_bit(wstr: PChar; var wc: TJccgMas);
Sub Wget_bit(ByVal w As String, wc As TJccgMas)
void WINAPI Wget_bit(char *, TJccgMas *);
概　要：マスターデータのWkd/Cfm情報を文字列バッファに得る。
引　数：wstr    wstr[X19MHZ]～wstr[X560MHZ]にWkd/Cfm情報を得る。
                wstrは、MAXBAND以上の領域が確保されていなければならない。
        wc      dbf_read()等で読み込まれた、マスターデータのバッファ。
                7MHzがCfm, 50MHzがWkd, 1200MHzがWkdであれば、
                "  C      W  W  "  という文字列が得られる。
```

```
procedure Wput_bit(var Th: TThLog; wstr: PChar);
Sub Wput_bit(Th As TThLog, ByVal w As String)
void WINAPI Wput_bit(TThLog *, char *);
概　要：マスターデータのバッファに、Wkd/Cfm情報をセットする。
引　数：Th      Turbo HAMLOGデータアクセス構造体
        wstr    セットするWkd/Cfm情報。
```

```
function ChosonCheck(const code: PChar): Integer;
Function ChosonCheck(ByVal s As String) As Long
int  WINAPI ChosonCheck(const char *);
```

概　要：コードのＤＸ,市,郡,区,町村 を識別し、それぞれ 3,4,5,6,7 を返す。
引　数：char *code　　コードの文字列
戻り値：ＤＸ市郡区町村に応じ、3～7の数値を返す。但し、コードが空白の場合は 1を返し、認識できなければ 0 を返す。
　　[例]　ChosonCheck("100101") で６を返す

```
function Get_lupdate(var dbf: TDBFh): PChar;
char * WINAPI Get_lupdate(TDBFh *);
```

概　要：データファイルの最終更新年月日を返す。
　　ファイルのタイムスタンプではない。
引　数：dbf　　dBASE互換データアクセス構造体
戻り値：ＤＬＬ内部の静的領域へのポインタで、"97/02/01"形式の文字列を返す。
　　Delphiの場合は、戻り値がそのままString型として使える。
※ＶＢ４では、Vget_lupdateを使う。

```
procedure THW_zap(var Th: TThLog);
Sub THW_zap(h As TThLog)
void WINAPI THW_zap(TThLog *);
```

概　要：ＱＳＯデータの件数をゼロにする。
引　数：Th　　Turbo HAMLOGデータアクセス構造体

```
function MakeIndex(const dname, Key, index: PChar): Integer;
Function MakeIndex(ByVal s As String, ByVal k As String, ByVal i As String) As Long
int  WINAPI MakeIndex(const char *, const char *, const char *);
```

概　要：データベースファイルをもとに、インデックスファイルを生成する。
　　データベースファイルとインデックスファイルは、オープンされているものであってはならない。（共有違反となる）
　　・HAMLOG.HDBでのキー名　{
　　{"CALLS", 6 },{"IGN", 14 },{"DATE", 4 },{"TIME", 2 },{"CODE", 6 },
　　{"GL", 6 },{"QSL", 3 },{"FLAG", 2 },
　　{"HIS" , 3 },{"MY" , 3 },{"FREQ", 7 },{"MODE", 4 },{"NAME", 12 },
　　{"QTH" , 28 },{"RMK1", 54 },{"RMK2", 54 },};
　　・HAMLOG.MSTでのキー名
　　"CODE","QTH","FLG","HED","ERIA","CFM","WKD","IDO"
引　数：dname　　データベースのファイル名
　　key　　インデックスを生成させるキー
　　index　　インデックスのファイル名
戻り値：成功の場合は SUCCESS を返す
[例] 日付時間順のインデックスファイルを生成する。
MakeIndex("C:¥¥Data¥¥Hamlog.hdb","DATE+TIME","C:¥¥Data¥¥DateTime.Ndx"); // Ｃ言語

```
function T_Open(const fname: PChar): Integer;
Function T_Open(ByVal f As String) As Long
HANDLE WINAPI T_Open(const char *);
```

概　要：テキストファイルをリードオンリーモードでオープンする。クローズするには、
　　Windows-APIの CloseHandle()を使う。
引　数：fname　　フルパスのファイル名

戻り値：整数値のファイルハンドルを返す。以下、クローズするまでこの値をもとにフ
　　ァイルアクセスをする。

```
function T_Gets(var buff; slen, handle: Integer): Integer;
Function T_Gets(ByVal b As String, ByVal l As Long, ByVal h As Long) As Long
int  WINAPI T_Gets(char *, int, HANDLE);
```
概　要：テキストファイルから１行読み込む。
引　数：buff　　１行分を読み込むバッファ。通常は文字配列。
　　slen　　バッファの長さ（バイト数）
　　handle　T_Open()で得たファイルハンドル
戻り値：正常に読み込めれば SUCCESSを返す

```
procedure T_Puts(const buff: PChar; Handle: Integer);
Sub T_Puts(ByVal b As String, ByVal hdl As Long)
void WINAPI T_Puts(const char *, HANDLE);
```
概　要：テキストファイルに１行書き込む。復帰改行を付ける。
引　数：buff　　書き込み文字列の入ったバッファ
　　Handle　ファイルハンドル。
　　DelphiのFileCreate()か、Windows-APIの CreateFile()、またはこのDLLの
T_Open()で得たファイルハンドルを使う。

```
function JStrStr(const s1, s2: PChar): Ineger;
Function JStrStr(ByVal s1 As String, ByVal s2 As String) As Long
int WINAPI JStrStr(const char *s1, const char *s2);
```
概　要：日本語が混じっている可能性のある文字列（シフトＪＩＳ）の中から、部分文字
列を探す。
引　数：s2 が s1 の中にあるかどうかを探す。
戻り値：部分文字列があればその位置を返す。無ければ －１を返す。
[例]　　JStrStr("埼玉県久喜市", "市") で10を返す

```
function DateTimeSort(const HdbFile: PChar): Integer;
Function DateTimeSort(ByVal HdbFile As String) As Long
int  WINAPI DateTimeSort(const char *);
```
概　要：QSOデータを、JST/UTCを認識して日付時間順にソートする。
　　QSOデータは、閉じられていること。
　　ソート完了後は、インデックスを再構築する必要あり。
　　インデックスを再構築するには、インデックスファイルを削除してか
HamlogOpen()すればよい。
引　数：QSOデータをフルパスで指定
戻り値：成功すればSUCCESSを返す。

```
function MasterSort(var Th: TThLog): Integer;
Function MasterSort(Th As TThLog) As Long
int  WINAPI MasterSort(TThLog *);
```
概　要：マスターデータをコード順にソートする。
　　データはHamlogOpen()にてオープンされてなければならない。
　　ソート完了後は、インデックスファイルを再構築すること。
　　インデックスを再構築するには、インデックスファイルを削除してから
HamlogOpen()すればよい。
引　数：Th　　Turbo HAMLOG データアクセス構造体
戻り値：成功すればSUCCESSを返す。失敗ならそれ以外の値。

```
function CopyFromMAS(var Th: TThLog): Longint;
Function CopyFromMAS(Th As TThLog) As Long
long WINAPI CopyFromMAS(TThLog *);
```
概　要：ＱＳＯデータ中のＱＴＨが空白のレコードに、コードを利用してマスターデータのＱＴＨ情報をコピーする。
　　ＱＳＯデータ中、コードが入力されてない、またはＱＴＨが既に入力されているレコードは無視される。
　　データはHamlogOpen()にてオープンされてなければならない。
引　数：Th　　Turbo HAMLOG データアクセス構造体
戻り値：ＱＴＨの書き込みがあった、一番小さいレコード番号を返す。
　　バックアップするときは、このレコード番号から始めればいいわけである。

```
function dbf_rcount(var dbf: TDBFh): LongInt;
Function dbf_rcount(d As TDBFh) As Long
long WINAPI dbf_rcount(TDBFh *);
```
概　要：レコード件数を返す。
引　数：dbf　　dBASE互換データアクセス構造体
戻り値：レコード件数を３２ビット整数で返す

```
function dbf_recno(var dbf: TDBFh): LongInt;
Function dbf_recno(d As TDBFh) As Long
long WINAPI dbf_recno(TDBFh *);
```
概　要：現在のレコード番号を返す。
引　数：dbf　　dBASE互換データアクセス構造体
戻り値：データの現在位置（レコード番号）を３２ビット整数で返す

```
procedure _Qsorter(var pData; const Cnt, size: Integer);
```
　Ｃ言語にはqsort()があるので省略
　VBはよくわからないので省略(^_^;)
概　要：配列をソートします。
引　数：pData　ソートしたい配列の先頭の要素
　　Cnt　　ソートしたい配列の数
　　size　　配列の１要素あたりのサイズ。下位１２ビットを実際の１要素あたりのサイズとして認識します。従って、最大4,095バイト。
　　sizeの12～20ビットを、比較対照とする文字列サイズとする。０でもよい。
　　sizeの21～29ビットを、文字列の他に整数があれば、比較対照とする整数の存在するオフセットとする。
　　sizeの30ビットが１であれば、単に３２ビットの整数としてソートします。
　　sizeの最上位ビット(31)が１であれば、降順にソートします。
[例]　　// Delphi
```
var
　　fData: array [0..5] of Integer;
begin
　　fData[0] := 900;
　　fData[1] := 200;
　　fData[2] := 100;
　　fData[3] := 400;
　　fData[4] := 400;
　　fData[5] := 600;
　　_Qsorter(fData[0], 6, Sizeof(Integer) or $C0000000);
// 結果、整数の配列であるfData が降順にソートされる。
end
```

```
char * WINAPI Han2ZenStr(const char *);
```

function Han2ZenStr(const str: PChar): PChar;
概　要：半角英数字を、全角英数字にして返す。
　　０はφにする。"10"は"１φ"先頭文字が'@'の場合は、０は０のまま返す。
　　"@200"は"２００"
引　数：str　　半角英数字及び空白
戻り値：ＤＬＬ内部の静的領域へのポインタで、全角に変換した英数字を返す。半角スペースは二つ返す。
　　Delphiでは、戻り値がString型としてそのまま使える。

---

void WINAPI GetDxEntity(const char *calls, char *buff);
procedure GetDxEntity(const calls: PChar; buff: PChar);
概　要：ＤＸエンティティを得ます。
引　数：calls フルコールサイン
　　buff エンティティを３バイト以内で返します。
　　あらかじめ４バイト以上のバッファを確保してから関数を呼び出す必要があります。

---

int WINAPI isPortable(const char *Calls);
function isPortable(const c: PChar): Integer;
概　要：国内運用局のエリア番号を数値で返す。
引　数：Calls フルコールサイン
戻り値：移動局なら移動エリア 0～9 を返します。
　　固定局、ＤＸ局なら -1 を返します。

---

int WINAPI DbsDbrOpen(TOldLog *, const char *);
function DbsDbrOpen(var th: TOldLog; fname: PChar): Integer;
概　要：旧QSOデータをオープンする。
　　同一のフォルダに*.DBS, *.DBRが存在する必要があります。
引　数：th　　旧Turbo HAMLOG データアクセス構造体
　　fname　HAMLOG.DBSのファイル名のフルパス
戻り値：成功すればSUCCESSを返す。失敗ならそれ以外の値。

---

void WINAPI DbsDbrClose(TOldLog *);
procedure DbsDbrClose(var th: TOldLog); StdCall;
概　要：旧QSOデータをクローズする。
引　数：th　　旧Turbo HAMLOG データアクセス構造体

---

int WINAPI DbsDbrRead(TOldLog *, const long);
function DbsDbrRead(var th: TOldLog; rno: LongInt): Integer;
概　要：旧QSOデータから１レコード分を構造体の中に読み込む
引　数：th　　旧Turbo HAMLOG データアクセス構造体
　　rno　　読み込むレコード番号
戻り値：成功すればSUCCESSを返す。失敗ならそれ以外の値。

BOOL WINAPI SetDbsShare(const int i);
function SetDbsShare(const i: Integer): Boolean;
概　要：HamlogOpen()を呼び出す前に、共有オープンの設定をする。
引　数：i　　1以上を指定すると共有オープン。0で現在の状況を得るだけ。
戻り値：Trueが返れば共有モード。Falseで非共有（排他）

※Ver5形式の構造体については、次のとおりです。

```
#define  is_DXQSO  8
#define  isCQ_CALLED 16
#define  Chk1_Checked 32
#define  Chk2_Checked 64

typedef struct  {
   char  Calls[21],  // コールサイン
         Date[9],    // 日付  04/08/20形式
         Time[7],    // 時刻  10:20J形式
         Code[7],    // JCC/JCG等コード
         Glid[7],    // グリッドロケーター
         Qsl[4];     // Qsl, Send, Rcv
   WORD  Flag1;      // フラグ
   char  Hiss[764];  // His-RST 及びその他の項目のバッファ
   char  *Myrs, *Freq, *Mode, *Name, *Qth, *Rmk1, *Rmk2; // バッファへのポインタ
   BYTE  HissLen, MyrsLen, FreqLen, ModeLen, NameLen, QthLen,
         Rmk1Len, Rmk2Len;
} TQsoBuff;
```

・内部にポインタがあるので、ファイルオープン後に構造体をまるごと初期化してはいけない。
・DX局の場合、(Flag1 & is_DXQSO)が真となる。
・DX局のコールサインは／を含めずデュプチェックする設定にした場合は、
　8Q7/JG1MOUでは (Flag1 & 2)が真となる。※コールサインが右側
　JG1MOU/KH6では (Flag1 & 1)が真となる。
・CQにチェックでは (Flag1 & isCQ_CALLED) が真となる。
・１にチェックでは (Flag1 & Chk1_Checked) が真となる。
・２にチェックでは (Flag1 & Chk2_Checked) が真となる。

Turbo HAMLOGはデータベースソフトですので、キーポインタとレコードポインタという概念を理解してください。キーポインタとは、キーの現在位置です。

例えば、「現在"JG1MOU"というキーを指している。THW_skip(Th, -1, DbsCallDX) を呼び出すと、ひとつ前の"JG1MOA"という位置にキーが移動する」という考え方です。レコードポインタは、レコード番号の現在位置です。

THW_read() であれば、レコード番号を指定して直接データを読み出しますが、THW_skip() でデータを読み出すには、その前に THW_seek()で検索するか、あるいは THW_top()か THW_btm()でデータを読み出しておく必要があります。

なお、THW_seek() では、何十万件のデータでも瞬時に検索することができます。

ＱＳＯデータをアクセスするのであれば、HamlogOpen()でオープンし、THW_???()で読み書き検索し、終了時にはHamlogClose()でクローズすれば十分です。

テキストデータであれば、User_open(),User_Seek(), User_Read(), User_close()です。

このＤＬＬをＦＢにご活用下さい。

==========

＊　データ構造 HAMLOG.HDB

基本的にdBASE3と同じですが、dBASE3ではアクセスできません。

項目	幅	内容例	摘要
DELETE	1		dBASEのレコード削除マーク
CALLS	8	JG1MOU__	この部分がインデックスキー
IGN	12	_____1	コールサインの残り。移動エリアがあれば右詰め
DATE	4		年月日 (04/11/10の場合は 0x14,0x04,0x0b,0x0a)
TIME	2		時分 「分」の最上位ﾋﾞｯﾄが１ならUTC (12:01Jは 0x0C,0x01)

```
CODE    6       1332__
GL      6       PM96UB
QSL     3       J**
FLAG    2       DXの場合8～10  数値で保存
HIS     3～12   文字列で保存
MY      3～12
FREQ    7～16
MODE    4～16
NAME   12～64
QTH    28～128
RMK1   54～254
RMK2   54～254
-------------------------------------------
```
計１レコードで209～800バイト（うち、削除ﾏｰｸが1ﾊﾞｲﾄ）

＊ ＨＡＭＬＯＧのＱＳＯデータの読み込み方法（とりあえずＣ言語で説明）

　自作ツールでTurbo HAMLOGのＱＳＯデータを読み込むための手順を説明いたします。書き込みは、読み込みの逆をやればいいので、省略いたします。
　Turbo HAMLOGのＱＳＯデータ（HAMLOG.HDB）は、dBASE3と同じ記録方法をとっておりますが、一部バイナリーで記録しているので、dBASE3では読み書きができません。

　dBASE3のヘッダ長は、hedlen = 32 * fldcnt + 33 で求められます。fldcntはフィールドの数で、HAMLOG.HDBの場合は、16です。従って、32 * 16 + 33 で、hedlen = 545 バイトです。
　そしてデータの終わりに"¥x1a"を書き込みますので、データがゼロ件の場合、ファイルサイズは546バイトとなります。

＊ヘッダ部分の構成

ｵﾌｾｯﾄ	長さ（ﾊﾞｲﾄ）	内　　容
0	1	03h=ﾒﾓﾌｨｰﾙﾄﾞ無し　83h=ﾒﾓﾌｨｰﾙﾄﾞ有り（HAMLOGでは 1Ah）
1～3	3	最終更新年月日
4～7	4	レコード件数（longにキャスト）
8～9	2	ヘッダの長さ（shortにキャスト。HAMLOG.HDBは545）
10～11	2	レコードの長さ（HAMLOG.HDBは209～800）
12～31	20	00h
32～		32バイトずつ各フィールドを定義　※１を参照
hedlen-1	1	0Dh
hedlen	1	1Ah

```
  ※１
for (i=1; i<=FIELD_COUNT; i++) {
  strcpy(HedBuff+(i*32), FieldName[i-1]);  // ﾌｨｰﾙﾄﾞ名
  HedBuff[i * 32 + 11] = 'C';        //  文字型
  HedBuff[i * 32 + 16] = (BYTE)FieldLength[i-1]; // ﾌｨｰﾙﾄﾞ長
}
```

・レコードを読み込むバッファの形式

以下のような構造体に読み込みます。dbf_read()の場合

```
typedef struct  {
  char  calls[20], // ｺｰﾙｻｲﾝ
        date[4],   // 日付
        time[2],   // 時間
        code[6],   // JCCｺｰﾄﾞ
        glid[6],   // ｸﾞﾘｯﾄﾞﾛｹｰﾀｰ
```

```
        qsl [3];   // QSL Via, Sent, Rcv
  WORD  flag1;
  char  hiss[756]; // MAX12
  char  *myrs,     // MAX12
        *freq,     // MAX16
        *mode,     // MAX16
        *name,     // MAX64
        *qth,      // MAX128
        *rmk1,     // MAX254
        *rmk2;     // MAX254
  char  dummy;
} TLogData;
```

日付・時間は文字ではなく数値で記録します。

＊日付の保存形式　　＊時間の保存形式

```
TLogData  dbs;
  dbs.date[0] == 年(19～20)
  dbs.date[1] == 年          dbs.time[0] == 時
  dbs.date[2] == 月          dbs.time[1] == 分
  dbs.date[3] == 日          分の最上位ビットが１であればＵＴＣ
```