The HashTools Component

By David Adams

TN 05-43

Overview

ハッシュとは、文字列、BLOB、ピクチャ、ドキュメントといったデータの固まりを単一の倍長整数に置き換えるテクニックのことです。ハッシュは、チェックサムとして使用したり、ルックアップや検索を最適化するために利用することができます。HashTools コンポーネントでは、9 種類のハッシュアルゴリズムを採用しており、テキスト、ピクチャ、BLOB およびドキュメントのハッシュに対応しています。加えて、コンポーネントには、テキスト、BLOB、ピクチャフィールドに対するハッシュ検索用のコードが付属しています。このテクニカルノートでは、コンポーネントを構成している Public および Protected 属性の各メソッドについて解説しています。HashTools コンポーネントおよびハッシュの応用については、下記のテクニカルノートでも取り上げています。

- 05-44 Optimizing Searches with Hashes では、HashTools コンポーネントのソースコードについて解説し、ハッシュがどのようにテキストや BLOB のクエリを最適化するかを論じています。様々なハッシュアルゴリズムの使用によるパフォーマンスの違いを比較できるようなテスト結果も紹介しています。使用しているサンプルデータベースは、本テクニカルノートと同一のものです。
- 05-41 Case-Sensitive Operations in 4th Dimension では、4th Dimension のクエリで大文字と小文字を区別する方法を幾つか紹介しています。その中のひとつとして HashTools コンポーネントも挙げられています。
- 05-42 Scanning Text and BLOBs Efficiently では、テキストおよび BLOB を解析する上でもっとも効果的なコードを作成するための方法について、スピードとメモリ使用量の両面から詳しく論じています。

Why a Component?

4th Dimension の様々なリソースのコレクションをパッケージ化したコンポーネントは、容易に配布することができます。一部のデベロッパは、昔のバージョンが問題を抱えていたため、あるいは Protected 属性のメソッドがインストール後に見えなくなるためにコンポーネントに対して消極的です。HashTools をあえてコンポーネントとして提供する理由は次のとおりです。

- **Reduced Complexity**: ソースコードには 50 以上のメソッドが含まれており、そのほとんどは直接コールすることはできない性質のものです。コンポーネントでは 10 ほどのメソッドだけを公開しており、約半数はドキュメント、エラー処理、コンパイル宣言に関

わるものです。

- **Simplified Updating**: HashTools ルーチンは、前述のように別のテクニカルノートでは異なるサンプルデータベースに組み込まれて利用されています。これはどんなデータベースについてもいえることですが、共通コードをコンポーネント化することにより、すべてのデータベースを容易に管理更新することができます。
- **Efficient Error Testing**: ハッシュが動作するためには、内部的にポインタやドキュメントリファレンスなどの妥当性をチェックする必要があります。*HashText_RS* などのローレベルメソッドにそうしたチェック処理組み込んで冗長なものにする代わりに、Private 属性にして一切のパラメータチェックを省き、*HashTools_HashText* などの Protected メソッドをゲートウェイとして利用すれば、そこで包括的なパラメータチェックを実行して Private メソッドをコールすることができます。このような構成により、効率良くコンポーネントを不正なデータから守り、コードをできるだけすっきりしたものにとどめることができます。

Noteテクニカルノート 05-44 Optimizing Searches with Hashes には HashTools コンポーネントのオリジナルソースコードが含まれており、自由に変更・拡張することができます。

Compilation

HashTools コンポーネントは、インタプリタモードではなく、コンパイルモードでの使用を意図して設計されています。コンポーネントでは、使用するパラメータ、変数、配列を完全に宣言しています。4th Dimension のコンパイルオプションは、「すべて定義させる」を含め、どれを利用しても構いません。

Internal Documentation

コンポーネントの閲覧可能なメソッドには、それぞれエクスプローラコメント文が記述されており、メソッドのパラメータおよび例題がドキュメントされています。加えて *HashTools Read Me Public* メソッドには、それぞれのメソッドについての説明文が書かれています。

Supported Hash Algorithms

ハッシュ関数の種類は数百を超え、特定の状況における最適のハッシュは、扱われるデータによって決まります。HashTools コンポーネントには、C による 8 種類の高度なハッシュが含まれています。加えて、教示用に 9 番目の SumBytes という関数も含まれています。C によるオリジナルコード、および C++、Object Pascal、Java 版は、すべて Arash Partow 氏のサイトでみつけることができます。

http://www.partow.net/programming/hashfunctions/

同氏のサイトからの転載である下記の図には、HashTools コンポーネントで利用されている 9 種類のハッシュの概略が示されています。

Algorithm	Notes
AP	Arash Partow 氏の手による新開発のハッシュアルゴリズム。
BKDR	Kernighan 氏と Ritchie 氏の共著 *The C Programming Language* から取られたアルゴリズム。
DJB	Daniel J. Bernstein 氏が当初 comp.lang.c ニュースグループで公開した高性能のアルゴリズム。
ELF	PJW ハッシュの 32 ビット版。
JS	Justin Sobel 氏によって開発されたビットワイズハッシュ。
PJW	AT&T Bell 研究所の Peter J. Weinberger 氏が開発したアルゴリズム。
RS	Robert Sedgwick 氏の著書 *Algorithms in C* に掲載されたハッシュを最適化したもの。
SDBM	多くのデータベースプロジェクトで利用されている良質のハッシュ。
SumBytes	文字列、テキスト、BLOB、ピクチャ、ドキュメントのバイトを合算するだけの簡単なハッシュ。性能は低いが入門用に最適。

コードに含まれている有効なハッシュアルゴリズムのリストを取得するには、*HashTools_GetHashTypeNames* メソッドをコールします。

```
ARRAY TEXT($hashMethodNames;0)
HashTools_GetHashTypeNames(->$hashMethodNames)
```

Note: *HashTools_GetHashTypeNames* には、文字列またはテキストの配列に対するポインタを渡してください。文字列配列の場合、各要素は 8 バイト以上である必要があります。

Selecting a Hashing Algorithm

ハッシュのアルゴリズムは、それぞれハッシュを作成するためにかかる時間と、衝突(異なる値から重複するハッシュが生まれること)の起こる頻度に差があります。テクニカルノート 05-44 Optimizing Searches with Hashes では、ハッシュが動作する仕組み、衝突の説明、および様々なアルゴリズムを比較検証する方法について論じています。同ノートでは、9 種類のアルゴリズムを異なるデータセットに対して適用した際のテスト結果もまとめられています。テスト結果から、4th Dimension プロジェクトにおいては AP、BKDR、RS、SDBM アルゴリズムが機能的には同等であることがわかるはずです。

Hashing Values

HashTools コンポーネントには、異なるタイプの値をハッシュするために 4 つのルーチンが含まれています。各ルーチンには、正しいデータタイプの値に対するリファレンス、有効なハッシュアルゴリズムの名前を渡し、返り値として倍長整数のハッシュが返されます。有効なハッ

シュアルゴリズムの名前は *HashTools_GetHashTypeNames* で取得することができます。不正なポインタやアルゴリズム名が渡されるとエラーが発生します。エラーの詳細については後述します。

HashTools_HashBlob

HashTools_HashBlob (Pointer;Text)-> Longint

HashTools_HashBlob (->BLOB;"Hash Method")-> Hash

BLOB をハッシュし、結果を返します。

```
C_LONGINT($hash)
$hash:= HashTools_HashBlob(->[Sample]BLOB;"BKDR")
```

Note: BLOB はポインタで渡しますが、パフォーマンス上の理由から、内部的にはコピーされた上で処理されます。この動作がメモリ使用に関連して問題を引き起こすようであれば、ポインタ経由で処理するようにコードを書き換えてください。

HashTools_HashDocument

HashTools_HashDocument (Time;Text)-> Longint

HashTools_HashDocument (DocRef;"Hash Method")-> Hash

ドキュメントをハッシュし、結果を返します。

```
C_TIME($docref)
$docref:=Open document("")
If (OK=1)
CREATE RECORD([StoredDocumentData])
[StoredDocumentData]Hash:= HashTools_HashDocument ($docref;"AP")
[StoredDocumentData]Path:=Document
SAVE RECORD([StoredDocumentData])
UNLOAD RECORD([StoredDocumentData])
CLOSE DOCUMENT($docref)
End if
```

リファレンスを渡すドキュメントは、既に開かれたものである必要があります。そうでなければエラーになります。HashTools_HashDocument の実行後もドキュメントは開かれています。

Note: ドキュメントは、全体を読み込まれる代わりに、32000 バイトずつのチャンクで処理されます。これにより HashTools_HashDocument は、大きなサイズのドキュメントを比較的速く、メモリをさほど消費せずにハッシュすることができます。

HashTools_HashPicture

HashTools_HashPicture (Pointer;Text)-> Longint

HashTools_HashPicture (->Picture;"Hash Method")-> Hash

ピクチャをハッシュし、結果を返します。

```
C_LONGINT($hash)
$hash:= HashTools_HashPicture(->[Sample]Picture;"SDBM")
```

Note: ピクチャはポインタで渡しますが、パフォーマンス上の理由から内部的にはコピーされた上で処理されます。この動作がメモリ使用に関連して問題を引き起こすようであれば、ポインタ経由で処理するようにコードを書き換えてください。

HashTools_HashText

HashTools_HashText (Pointer;Text)-> Longint

HashTools_HashText (->Text;"Hash Method")-> Hash

文字列またはテキストをハッシュし、結果を返します。

```
C_LONGINT($hash)
$hash:= HashTools_HashText (->[Sample]Text;"SDBM")
```

Note: テキストはポインタで渡しますが、文字列をポインタ経由でスキャンしないように内部的にはコピーされた上で処理されます。4th Dimension のバージョンによっては、15000 文字を超えるテキストブロックのスキャンが著しいパフォーマンスの低下を招くことがあります。詳しくはテクニカルノート 05-42 Scanning Text and BLOBs Efficiently をご覧ください。

Finding Records by Stored Hashes

HashTools_FindByHash ルーチンは、自動的にストアドハッシュを使用し、文字列、テキスト、BLOB、ピクチャフィールドを探すことができます。メソッドのパラメータは下記のとおりです。

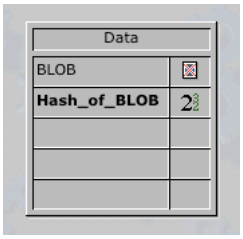
HashTools_FindByHash (Pointer;Longint;Pointer;Pointer;Boolean) -> Longint

HashTools_FindByHash (->Value to match;Hash to match;->Value field; ->Hash field;{Search selection?})-> Hash matches

以下にパラメータに関する説明を補足します。

Parameter	#	Type	Notes
Hash matches	$0	Longint	ハッシュに合致するレコード数。この値はハッシュの解析には有用ですが、通常は要りません。結果レコードの数を知るには Records in selection を使用してください。
Value to match	$1	Pointer	探すべき文字列、テキスト、BLOB、ピクチャに対するポインタ。
Hash to match	$2	Longint	探すべきハッシュの値。ハッシュは何らかの方法で作成したものを渡す必要があります。
Value field	$3	Pointer	対象となる文字列、テキスト、BLOB、ピクチャフィールドに対するポインタ。$3 と$1 の型は一致している必要があります。
Hash field	$4	Pointer	対象となるハッシュが保存されているフィールドへのポインタ。
Search selection?	$5	Boolean	カレントセレクションに対する絞り込みクエリ。デフォルト値は False つまり全レコードに対するクエリ。

ルーチンは、示されるハッシュおよびハッシュテーブルにまったく依存しています。ルーチンがハッシュをすることはなく、使用前にハッシュおよびハッシュテーブルを用意しておく必要があります。たとえば、下図のように BLOB を収めたテーブルがあるとします。



このテーブルストラクチャでは、[Data]BLOB フィールドのデータを SDBM ハッシュして[Data]Hash_of_BLOB フィールドに登録しています。ローカル変数$BlobToMatch_blob と合致する BLOB を[Data]BLOB の中から探す場合、次のようなコードを実行します。

```
C_LONGINT($hash)
` Hash the local BLOB using the same method used to hash the stored BLOBs.
$hash:=HashTools_HashBLOB (->$BlobToMatch_blob;"SDBM")
HashTools_FindByHash (->BlobToMatch_blob ;$hash;->[Data]BLOB;->[Data]Hash_of_BLOB)
```

HashTools_FindByHash ルーチンは、内部的にまず[Data]Hash_of_BLOB に対するインデックス検索を実行し、渡された BLOB と一致しないハッシュをすべて除外します。BLOB レコー

ドが重複していなければ、恐らく BLOB の候補はひとつ程度に絞り込まれます。続いて候補データと探すべきデータが完全に一致するかを調べます。HashTools_FindByHash ルーチンは、文字列、テキスト、BLOB、ピクチャフィールドに対応しています。

Tip 保存されたハッシュは、トリガで管理すると良いでしょう。

Error Management

HashTools コンポーネントに含まれているルーチンは、いずれも最初にカスタムエラー処理をインストールします。すでに他のエラー処理がインストールされている場合、それは一時的に止められ、HashTools ルーチンの完了とともに復旧します。HashTools の実行中にエラーが発生した場合、メソッド名とエラーコードが記録され、以下のメソッドをコールすることによって取得できます。

HashTools_GetLastErrorLocation

HashTools_GetLastErrorLocation ()-> Text

HashTools_GetLastErrorLocation ()-> エラーが発生したのであれば、そのメソッド名

HashTools_GetLastErrorCode

HashTools_GetLastErrorCode ()-> Longint

HashTools_GetLastErrorCode ():-> エラーコード、エラーがなければ 0

HashTools_GetErrorText

HashTools_GetErrorText (Longint)-> Text

HashTools_GetErrorText (Error code)-> エラーメッセージ

HashTools によって返されるすべてのエラーは、そのテキストメッセージを取得することができます。最後にセットされたエラーメッセージを読み取るには、次のコードを実行します。

HashTools_GetErrorText (*HashTools_GetLastErrorCode*)

Defined Error Strings

HashTools で定義されたエラーは次のとおりです。

#	Error Text
1	Required parameter(s) not passed to HashTools. パラメータの数が足りない。

2	Bad hash method type passed to HashTools. ハッシュ名が不正。
3	Bad document reference passed to HashTools. ドキュメントに対するリファレンスが不正。
4	Unrecognized hash method type, HashTools internal error. HashTools 内部エラー、ハッシュ名が認識できない。
5	Nil pointer passed to HashTools. ヌルポインタが渡された。
6	Pointer to wrong data type passed to HashTools, BLOB pointer expected. BLOB を指すべきポインタが BLOB を指していない。
7	Pointer to wrong data type passed to HashTools, alpha/text pointer expected. 文字/テキストを指すべきポインタが文字/テキストを指していない。。
8	Bad pointer passed to HashTools, pointer to search field expected. 対象データフィールドに不当なポインタが渡された。
9	Bad pointer passed to HashTools, pointer to hash field expected. ハッシュフィールドに不当なポインタが渡された。
10	Pointer to wrong search field type, string, text, BLOB, or picture field expected. 対象データフィールドに対するポインタが異なるタイプのフィールドを指している。
11	Pointer to wrong hash field type, numeric field expected. ハッシュフィールドに対するポインタが数値型ではないフィールドを指している。
12	Search value and search field types do not agree. 探すべき値と対象フィールドのタイプが合致しない。
13	Calling HashTools_GetErrorText without the required error code parameter. 渡されたエラーコードが不正。
14	Bad pointer passed to HashTools, pointer to picture expected. ピクチャを指すべきポインタがピクチャを指していない。。

Summary

HashTools コンポーネントには、文字列、テキスト、BLOB、ピクチャ、ドキュメントに対応した便利なハッシュツールが揃っており、文字列、テキスト、BLOB、ピクチャフィールドに対応した検索ツールも付属しています。ハッシュの仕組みと応用法、テキストや BLOB に対する検索コードの作成については、テクニカルノート 05-44 Optimizing Searches with Hashes、05-41 Case-Sensitive Operations in 4th Dimension、および 05-42 Scanning Text and BLOBs Efficiently で詳しく考察されています。