THE

RECORD MANAGER

Version 2.5

User's Guide

• for Visual Basic developers •

INTRODUCTION

Tsunami is a key-indexed record manager written for use with most 32-bit Windows compilers and languages. Visual Basic developers: add the following module to your projects…

```
TRM_VB.BAS
```

… and you're ready to call any of the 38 functions in the Tsunami TRM.DLL to create, build and manage your database. Tsunami has been tested on Windows 95, 98, 98SE, ME, NT 4.0, 2000 and XP. The Tsunami **License Agreement** is located in Appendix "A". This document assumes the reader has a working knowledge of record managers and database management in general.

OVERVIEW

File Structure

A Tsunami file is a "page structured" file. You define the page size used within a Tsunami file when you create it. Pages within a Tsunami file are used for storing file header information, records, index keys and internal pointers. As a programmer, it isn't necessary for you to know how or where Tsunami stores these items, but you will need a little information to select an appropriate page size for each file you create.

Page sizes can be anywhere from 1K to 8K, in increments of 1K. Internally, Tsunami uses long integer variables for it's page pointers. A long integer variable can hold a value of up to 2,147,483,647 so theoretically, a Tsunami file could grow in size to a maximum of 16 terabytes when using an 8K page size. That's enough to fill over 200 80-Gigabyte hard drives. Tsunami also uses long integer variables internally as unique record IDs, which means you can store over 2 billion records in each file.

When specifying the page size for a new Tsunami file, your decision could affect storage efficiency and/or I/O speed. Using a page size of 1K or 2K will usually result in the fastest read/write operations when the records intended for that file are normally small enough to fit on a page that size.

For files that will (or could) contain records larger than the page size you defined, read/write operations won't be quite as fast if multiple pages must be accessed to retrieve the whole record. Since Tsunami allows variable length records, you may need to estimate the average anticipated record length and base your page size decision on that figure. For example, if your Tsunami file is intended to hold customer records that are all exactly 200 bytes each in size, use a 1K or 2K page size. However, if your customer records could be anywhere from 200 bytes to over 200,000 bytes long, then using an 8K page size would probably be appropriate.

There's no real maximum record size for a Tsunami file. When defining the page size, simply remember… select a small page size to increase access speed unless the records are going to be unusually large.

Indexes / Keys

Tsunami creates and maintains the indexes in your files as you insert, update and delete records. You may define 32 indexes for a file defined with a 1K page size, 64 indexes for a file using a 2K page size, 96 indexes when using a 3K page size and 128 indexes for files with page sizes of 4K or more. To be more accurate… those are the number of key "segments" you can define per file. Keys in a Tsunami file may be multi-segmented. This means you could define 1 key with as many as 128 segments, 128 keys with 1 segment each or anything in between (subject to page size limits on key segments). Keys are all stored in the same file with the records, no matter how many indexes you define. In this document, indexes are also referred to as key paths.

Keys may be anywhere from 1 byte to 255 bytes long. This key length maximum includes the total length of all segments defined for each key. Tsunami requires you to define at least one index for each file you create. If you want, all indexes may allow duplicates. A unique key index is not required in a Tsunami file.

When an index is defined to allow duplicates, Tsunami sorts all duplicate keys within each index, always storing the duplicates in "insert order". That way, any Tsunami function that performs a search on keys that have duplicates will always return the first match in the duplicate chain, representing the first record added with that key value.

When you define the indexes for a Tsunami file that you're about to create, the key segments may be located anywhere in the physical record, as long as they are within the first 65,535 bytes of the record (if the records are that large). Records can certainly be larger than 65,535 bytes, but key segment positions can't reside beyond the 65,535th byte.

Tsunami also allows you to define any key index as being either case-sensitive or case-insensitive. This allows your users to enter information in upper case, lower case or any combination and still locate records just as easily as they could in a "forced case" data entry approach.

Functions

The Tsunami Record Manager offers 38 functions to manage your data. Detailed explanations are provided in the TSUNAMI OPERATIONS section, starting on page 7.

```
trm_Accelerate      trm_GetByKeyPos         trm_GetLess        trm_SetKeyPath
trm_Close           trm_GetDirect           trm_GetNext        trm_StepFirst
trm_CloseAll        trm_GetEqual            trm_GetPosition    trm_StepLast
trm_Count           trm_GetEqualOrGreater   trm_GetPrev        trm_StepNext
trm_Create          trm_GetEqualOrLess      trm_Insert         trm_StepPrev
trm_CurrKeyPos      trm_GetFileDef          trm_Integrity      trm_TimeOut
trm_Delete          trm_GetFileVer          trm_Open           trm_Update
trm_FileIsOpen      trm_GetFirst            trm_Rebuild        trm_Version
trm_FileSize        trm_GetGreater          trm_Recover
trm_Flush           trm_GetLast             trm_Result
```

Current Record Position

Tsunami maintains two forms of positioning information for each open file… logical positioning and physical positioning. Logical positioning refers to your position in a file with regard to the current key path and consists of three elements…

- A POINTER TO THE "CURRENT RECORD" IN THE CURRENT KEY PATH
- A POINTER TO THE "PREVIOUS RECORD" (IF ANY) IN THE CURRENT KEY PATH
- A POINTER TO THE "NEXT RECORD" (IF ANY) IN THE CURRENT KEY PATH

Physical positioning consists only of…

- A POINTER TO THE PHYSICAL LOCATION OF THE "CURRENT RECORD"

Any successful trm_Insert, trm_Get or trm_Update function will establish logical positioning in the current key path. Logical positioning is required for certain Tsunami function calls, such as trm_Update or trm_Delete. Without a logical current record position, those functions will return a result code indicating an error condition.

Logical positioning is also required by both the trm_GetNext and the trm_GetPrev functions that are used to traverse your records via the current key path. If you have established logical positioning, that positioning information will remain unchanged if you call a Tsunami function that fails (such as trm_GetNext when you've already reached the end of the current key path).

After logical positioning has been established, you can switch key paths by using the trm_SetKeyPath function, which will maintain position on the current record, but in the key path you specified in your function call.

The trm_GetPosition function returns a 12-byte pointer to the current record that can be used at any time later to retrieve that record and re-establish it as the current record. This function comes in handy when you need to move to a different location in a file temporarily and then come back to the exact file position by using the trm_GetDirect function. You can store as many 12-byte record pointers as you like, so you might even use this function to store pointers in a string array as you select records from your file for processing or inclusion in a list or report.

The affect each function has on logical and physical positioning is outlined in the TSUNAMI OPERATIONS section, starting on page 7.

Data Compression

Whenever you create a Tsunami file, you must decide whether or not to allow data compression. For most situations, data compression will increase storage efficiency and access speeds (smaller records produce smaller files, resulting in faster access). The records in Tsunami files are compressed using RLE (Run Length Encoding) that compresses any contiguous run of duplicate characters (anywhere from 4 to 255 duplicates) into a 3-byte "compression block".

Tsunami's RLE algorithm uses the ASCII "zero" internally as a block marker to signify the start of each 3-byte compression block. It should be noted that any contiguous run of ASCII zeros will be compressed into a 3-byte compression block, just like any other ASCII character. However, all runs of ASCII zeros (even a "run" of 1) must be enclosed in a compression block. In records with a large number of isolated ASCII zeros, this may cause the compression effort to suffer. If it happens often enough, the algorithm will abandon the compression effort (if the result would end up being larger than the original record). This won't hurt anything, but in a case where you know in advance that your records will contain isolated ASCII zeros in large numbers (JPEG images, encrypted data, etc), you may wish to turn record compression off for that file. Even though the RLE algorithm is very fast, there's no reason to use it if the records aren't going to yield much (or any) compression.

You also have the option of compressing keys in Tsunami indexes (on an index-by-index basis) which compacts any contiguous run of blank spaces within a key into a single blank space, and strips off any leading or trailing blank spaces. Compressing keys will result in smaller files and faster access, but there may be times when certain keys should not be compressed. For example, a key that contains right justified text (like an account number), will not sort properly if it's leading spaces are removed during key compression.

Using the compression options could also have an impact on your page size selection. If you know the approximate average size of the records to be stored, and you know in advance that approximately 1/3 of each record will normally consist of blank spaces (or some other repeating character), then you could safely estimate the record size at 2/3 of the approximate average size prior to compression. This should assist you in selecting the best page size for your application.

Free Space Utilization

When you use the trm_Delete function to remove a record from a Tsunami file, the record is physically deleted from the page(s) it occupied. This means that there will be a "hole" in your data file left by the record that was removed. Tsunami maintains an internal list of free space created by record deletions, using that space again for subsequent record insertions. This eliminates the need for periodic "packing" or reorganizing of your data files to clean up after numerous records are deleted over time. Rebuilding a Tsunami file using the trm_Rebuild function (see page 63) is only necessary when…

- you'd like to compact a file from which a large number of records have been deleted and you don't expect any future record insertions to re-occupy the vacated space, or…

- you need to change a file's definition or it's key segment definitions.

Limitations

Other than limiting files to no more than 2,147,483,647 records, Tsunami only has two real limitations that you might actually run into, but it's still important for you to note them…

1.) Tsunami files can have no more than 32 to 128 key segments defined for any one file, depending on the defined page size (see page 2).

2.) Tsunami can't open more than 255 files at a time. This limit may also be affected by the number of physical file handles being made available to your application by Windows.

Multi-User

Tsunami allows you to open files in either single-user mode or multi-user mode. In single-user mode, Tsunami opens a file for exclusive access… no other thread, application or user can access the file until you close it. In multi-user mode, Tsunami opens the file in full shared mode, using a passive concurrency model to allow unrestricted access by all users. Files are only locked during the few milliseconds that it takes for Tsunami to perform the requested operation(s).

In the rare event of a system hang or crash immediately after a file has been locked, a two second "time-out" result code (99) is returned to any other user who attempts to access the locked file. In that rare event, a system-wide reset might be required. This result code could also be the result of an attempt to open a file that's already open in single-user, exclusive mode.

Tsunami's use of passive concurrency allows more than one user to establish the same record in a file as their current record. If one user deletes that current record, and another then tries to update it, the second user will receive the "Lost record position" result code (22). The same thing applies if the first user updates the record… the second user would need to call trm_GetPosition followed by trm_GetDirect to reload the updated record. It's always the programmer's job to watch for result code 22 and handle it within the application.

If two users have established the same record in a file as their current record and one of the users deletes that record, Tsunami will not lose it's position if the second user invokes the trm_GetNext or trm_GetPrev function to move forward or backward in the current key path… the function will succeed. However, if the first user deleted not only the current record, but also the next and/or previous record, the second user would then receive the "Lost record position" result code (22).

In multi-user mode, Tsunami writes all changes to it's files as they are made before unlocking them, so files will always reflect their current state when other users access them. In single-user mode, Tsunami's file header information is not normally updated until you close the file. This provides a measurable speed improvement, but can also leave a file open to corruption due to an invalid header page if the computer crashes or hangs before the file is closed. In that event, you may need to use either the trm_Rebuild or trm_Recover function to salvage the records in the file.

To help avoid this, you can use the trm_Flush function periodically to force an update of all header pages and force all cached data to disk. This is done automatically for you after every operation that alters any file opened in multi-user mode, but sometimes you might find it necessary to open files in single-user (exclusive) mode and still ensure file integrity.

When accessing a Tsunami file in multiple threads within an application, the file must be opened in multi-user mode. Each thread must open the file independently to obtain it's own file handle. There are no artificial limits on the number of concurrent users or threads that may access Tsunami files... only the number of files that Tsunami can open at one time (255).

Locking / Unlocking

Tsunami does not include specific “lock” or “unlock” functions for records or files. When a file is opened in single-user mode, it is obviously locked by Tsunami for exclusive access. When a file is opened in multi-user mode, Tsunami handles all the necessary file locks to guard against potential file corruption.

If you need temporary exclusive access to a file that you have opened in multi-user mode, you can simply open the file again, but in single-user mode. You don’t need to close the file first… you can have it open in multi-user mode and still open it again in single-user mode (receiving another file handle). You can then perform the operation(s) that need exclusive access (using the single-user file handle), close the file (with the single-user file handle) and return directly to multi-user mode.

The logistics involved in accomplishing this are internal to Tsunami… you don’t need to deal with it from a programmer’s standpoint.

TSUNAMI OPERATIONS

The following pages will detail each of Tsunami's 38 function calls, presented in alphabetical order. For each function, we provide…

- The function name (and it's operation code number)
- A description of the function's purpose
- Syntax examples, detailing the parameters required by the function
- Any prerequisites that must be satisfied before the function can be successfully called
- Results that may be expected of both a successful and an unsuccessful call
- Affect on current positioning (if any)

As you will learn in the very next section, Tsunami has more than one interface option. In it's string-based option, half of Tsunami's functions return a numeric result code directly, while the other half (the trm_Get and trm_Step functions) return a string and must therefore report their numeric result code through the trm_Result function. All string-based syntax examples for the trm_Get and trm_Step functions will include the example call to obtain a result code.

Three Interface (API) Options

Tsunami offers three interfaces. Visual Basic developers should only use the string-based API. The other two "pointer-based" APIs are listed in this guide purely for information.

- **String-based API**

Tsunami's original interface is comprised of 38 function calls, most of which have one or more parameters, many of which are string variables. In this document, that interface will be referred to as Tsunami's string-based API. For those string-based calls that return a string instead of a result code, the result codes are obtained through a subsequent call to trm_Result. This example shows a typical call using the string-based API…

```
Record$ = trm_GetEqual(hFile&, KeyNo&, KeyVal$)
Result& = trm_Result(hFile&)
```

In this example, the appropriate file handle must be in the hFile& variable, the key path number (1-128) must be in the KeyNo& variable and the key value being searched for needs to be in the KeyVal$ variable.

If successful, Tsunami returns the requested record in the Record$ variable. If unsuccessful, the Record$ variable will be a null string and a subsequent call to trm_Result returns a result code explaining why a record was not retrieved.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

Tsunami also offers two optional "single-call" interfaces. In this document, those two interfaces will be known as the pointer-based APIs, since they pass pointers to strings instead of passing string variables. The first pointer-based API passes seven long integer parameters to Tsunami as follows…

```
Op& = 5 ' trm_GetEqual
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyPtr& = StrPtr(KeyVal$)
KeyLen& = Len(KeyVal$)
KeyNo& = 1

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, KeyPtr&, KeyLen&, KeyNo&)
```

In this example, the Op& parameter must hold the desired operation code and the hFile& parameter must hold the Tsunami file handle.

Tsunami also needs to know the memory address of the string variable, buffer or UDT variable intended to hold the retrieved record. Pass that address in the DataPtr& parameter and pass the size (length) of that string variable, buffer or UDT variable in the DataLen& parameter.

The KeyPtr& parameter must hold a value representing the memory address of the string variable or buffer that holds the key value being searched for, and the KeyLen& parameter must hold a value that represents the length of the key being passed. If you're using a buffer to pass the key value and the buffer is larger than the key, be sure to pass the length of the key itself, not the length of the buffer holding the key. This example also requires a key number so Tsunami knows which index to search… obviously, that's passed in the KeyNo& parameter.

If successful, the result code will be 0, the desired record will be in the string variable, buffer or UDT pointed to by the DataPtr& parameter and the length of the retrieved record will be returned in the DataLen& parameter. When unsuccessful, a non-zero result code is returned. If the result code is 50 (Data buffer too small) nothing is returned in the string variable, buffer or UDT. However, the DataPtr& and DataLen& parameters will then return the memory address and length of the complete record so you can retrieve it directly without another Tsunami call.

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

Tsunami's second pointer-based API passes a single parameter… a User Defined Type that has seven long integer elements…

```
Dim Tsu As TRMtype

Tsu.op = 5 ' trm_GetEqual
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyptr = StrPtr(KeyVal$)
Tsu.keylen = Len(KeyVal$)
Tsu.keyno = 1

Result& = trm_udt(Tsu)
```

In this example, the Tsu.op element must hold the desired operation code and the Tsu.file element must hold the Tsunami file handle.

Tsunami also needs to know the memory address of the string variable, buffer or UDT variable intended to hold the retrieved record. Pass that address in the Tsu.dataptr element and pass the size (length) of that string variable, buffer or UDT variable in the Tsu.datalen element.

The Tsu.keyptr element must hold a value representing the memory address of the string variable or buffer that holds the key value being searched for, and the Tsu.keylen element must hold a value that represents the length of the key being passed. If you're using a buffer to pass the key value and the buffer is larger than the key, be sure to pass the length of the key itself, not the length of the buffer holding the key. This example also requires a key number so Tsunami knows which index to search… obviously, that's passed in the Tsu.keyno element.

If successful, the result code will be 0, the desired record will be in the string variable, buffer or UDT pointed to by the Tsu.dataptr element and the length of the retrieved record will be returned in the Tsu.datalen element. When unsuccessful, a non-zero result code is returned. If the result code is 50 (Data buffer too small) nothing is returned in the string variable, buffer or UDT. However, the Tsu.dataptr and Tsu.datalen elements will then return the memory address and length of the complete record so you can retrieve it directly without another Tsunami call.

The pointer-based APIs were added to facilitate the use of Tsunami with as many other languages / compilers as possible. Visual Basic is only "directly" compatible with Tsunami's string-based API.

Key-Only Searches

Eleven of Tsunami's trm_Get functions that normally return a record, can also be used for key-only searches. To designate a key-only search, simply use Tsunami's KEY_ONLY constant. In the string-based API, add the KEY_ONLY constant as a bias to the function's file handle parameter (see the code example below). In Tsunami's pointer-based APIs, the KEY_ONLY constant is simply added to the Op& parameter or to the Tsu.op element, depending on which pointer-based API you're using.

In a key-only search, instead of receiving the full record, you will only receive the record's key for the designated (or current) key path, along with a 12-byte record pointer which is attached to the front of the returned key. That record pointer can then be used later in a trm_GetDirect call to retrieve the full record, if needed. To parse the record pointer from a returned key, use code similar to the following …

```
RetVal$ = trm_GetFirst(hFile& + KEY_ONLY, KeyPath&)

If Len(RetVal$) Then
   RecPtr$ = Left(RetVal$, 12)
   Key$ = Mid(RetVal$, 13)
Else
   Result& = trm_Result(hFile&)
End If
```

Key-only searches are often used to see if a certain key value exists within a file's index (using trm_GetEqual) without incurring the overhead of returning the entire record. This can prove very useful for files containing extremely large records.

Note: Logical and physical positioning information is still established by Tsunami during key-only searches, even though records are not retrieved.

trm_Accelerate (32)

trm_Accelerate should be used whenever you want to perform "batch" operations on a Tsunami file where speed is important. Tsunami will create and maintain an I/O cache for the designated file's key pages, greatly improving performance. Programmers must be aware that many key page updates are cached while the file is in accelerated mode and will only be written to disk when accelerated mode is turned off. Therefore, this function can only be used on files that are opened in single-user mode.

Note: You may accelerate up to 16 Tsunami files at a time. Acceleration can be turned on and off at any time without the need to close and re-open the file(s). If you neglect to turn off acceleration before you close a file, Tsunami will write the file's cache to disk before closing it.

Syntax

- **String-based API**

```
Result& = trm_Accelerate(hFile&, CacheSize&)
```

… where hFile& holds a file handle returned by the trm_Open function

… where CacheSize& holds a value from 1 to 1536, telling Tsunami how many megabytes of RAM to allow for the designated file's acceleration cache. Passing a value greater than 1536 is considered the same as passing 1536. This parameter may also hold one of the following "instruction" values …

 0 = Write cached pages to disk and close the file's cache (turn acceleration off)
-1 = Write cached pages to disk but leave the file's cache open (flush the cache)
-2 = Close the file's cache without writing cached pages to disk (after read-only activity)

Passing a positive value (1 to 1536) as the CacheSize& parameter turns caching on for the designated file. Tsunami will use that value to set the maximum memory allocation for the cache (1 megabyte to 1,536 megabytes). If the value you pass is greater than 75% of the physical RAM present on the PC, Tsunami will reduce the requested cache maximum to that 75% level.

Tsunami's cache is dynamic. Only 2% to 3% of the requested maximum is actually allocated when acceleration is turned on… the remainder of the requested amount is dynamically allocated as needed. Therefore, requesting more cache than you might need will <u>not</u> waste resources.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 32 ' trm_Accelerate
KeyNo& = 10 ' 10MB cache

Result& = trm(Op&, hFile&, 0, 0, 0, 0, KeyNo&)
```

trm_Accelerate (32) (continued)

Syntax (continued)

The KeyNo& parameter is used to pass trm_Accelerate instructions to Tsunami. When you want to turn acceleration on for a given file, put a value of 1 to 1536 in the KeyNo& parameter. To execute any of the other three trm_Accelerate options, put a 0, -1, or -2 in the KeyNo& parameter (described on the previous page, under the string-based syntax example).

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 32 ‘ trm_Accelerate
Tsu.file = hFile&
Tsu.keyno = 10 ‘ 10MB cache

Result& = trm_udt(Tsu)
```

The Tsu.keyno element is used to pass trm_Accelerate instructions to Tsunami. When you want to turn acceleration on for a given file, put a value of 1 to 1536 in the Tsu.keyno element. To execute any of the other three trm_Accelerate options, put a 0, -1, or –2 in the Tsu.keyno element (described on the previous page, under the string-based syntax example).

Prerequisites

The file to be accelerated must have been successfully opened in single-user mode by the trm_Open function.

Results

If successful… trm_Accelerate will return a result code of 0.

If unsuccessful… trm_Accelerate will return one of these result codes …

2	I/O error
3	File not open
40	Accelerated access denied
41	Acceleration cache error

Positioning

This function has no affect on positioning information.

trm_Close (1)

The trm_Close function is used to close a specific Tsunami file. Use this function instead of trm_CloseAll when you need to know if there was an I/O error while closing a certain file. Although trm_CloseAll will return a result code indicating an I/O error if it encounters any problems while closing multiple files, it can't inform you which file(s) generated the I/O error(s) like trm_Close can.

Syntax

- **String-based API**

```
Result& = trm_Close(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 1 ' trm_Close

Result& = trm(Op&, hFile&, 0, 0, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 1 ' trm_Close
Tsu.file = hFile&

Result& = trm_udt(Tsu)
```

Prerequisites

The file to be closed must have been successfully opened by the trm_Open function.

Results

If successful… trm_Close will return a result code of 0. The file will be closed and can't be accessed by the same file handle any more. To gain access to the file again, a subsequent trm_Open must successfully assign a new file handle.

If unsuccessful… trm_Close will return one of these result codes …

2 I/O error
3 File not open

Positioning

Logical and physical positioning information for the file is cleared.

trm_CloseAll (28)

The trm_CloseAll function is used when you want to close all Tsunami files that were opened by your application.

Syntax

- **String-based API**

```
Result& = trm_CloseAll
```

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 28 ' trm_CloseAll
Result& = trm(Op&, 0, 0, 0, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 28 ' trm_CloseAll
Result& = trm_udt(Tsu)
```

Prerequisites

The file(s) to be closed must have been opened by the trm_Open function.

Results

If successful… trm_CloseAll will return a result code of 0. All open files will be closed and can't be accessed by the same file handles any more. To gain access to any of the files again, subsequent trm_Open calls must successfully assign new file handles.

If unsuccessful… trm_CloseAll will return the following result code …

2 I/O error

Positioning

Logical and physical positioning information for the files is cleared.

trm_Count (17)

trm_Count returns the current record count for the designated file.

Syntax

- **String-based API**

```
Total& = trm_Count(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 17 ' trm_Count

Result& = trm(Op&, hFile&, 0, 0, 0, 0, KeyNo&)
```

The record count is returned in the KeyNo& parameter.

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 17 ' trm_Count
Tsu.file = hFile&

Result& = trm_udt(Tsu)
```

The record count is returned in the Tsu.keyno element.

Prerequisites

The designated file must have been opened by the trm_Open function.

Results

If successful… trm_Count will return a value of 0 or greater, representing the current number of records stored in the Tsunami file.

If unsuccessful… trm_Count will return one of these result codes (Note: When unsuccessful, the string-based API for this function returns the result code as a negative) …

2	I/O error
3	File not open
99	Time-out

trm_Count (17) (continued)

Positioning

This function has no affect on positioning information.

trm_Create (14)

The trm_Create function allows you to create (or recreate) a Tsunami file. Tsunami files can be created in advance and shipped with your application simply by writing a short stand-alone program that uses trm_Create to produce the necessary files, or you can produce your Tsunami files from within your application at run time. Your application might also need to create temporary Tsunami files from time to time for collecting & sorting subsets of records selected from one or more Tsunami files for a custom list or report.

Syntax

- **String-based API**

```
Result& = trm_Create(File$, FileDef$, OverWrite&)
```

… where File$ represents a string variable holding the name of the file to be created, optionally including a valid path. If you do specify a path, it must be a complete path, including the drive letter.

… where FileDef$ represents a string variable holding the information used by Tsunami to create a new empty data file. You can find a detailed description of the FileDef$ string beginning on the next page.

… where OverWrite& holds a value of 1 or 0 (Yes or No) to let Tsunami know whether or not you want to be notified when a file already exists before it's created. A value of 1 instructs Tsunami to create the new file even if the file already exists.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 14 ' trm_Create
DataPtr& = StrPtr(FileDef$)
DataLen& = Len(FileDef$)
KeyPtr& = StrPtr(File$)
KeyLen& = Len(File$)
KeyNo& = OverWrite&

Result& = trm(Op&, 0, DataPtr&, DataLen&, KeyPtr&, KeyLen&, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 14 ' trm_Create
Tsu.dataptr = StrPtr(FileDef$)
Tsu.datalen = Len(FileDef$)
Tsu.keyptr = StrPtr(File$)
Tsu.keylen = Len(File$)
Tsu.keyno = OverWrite&

Result& = trm_udt(Tsu)
```

trm_Create (14) (continued)

Prerequisites

The FileDef$ string must be properly constructed as outlined on the following pages. If you're re-creating an existing Tsunami file, the file can't currently be open by any other application, thread or user.

Results

If successful… trm_Create will return a result code of 0. The file will be created and closed. It must then be opened with a successful trm_Open call (which returns a file handle) before it can be accessed.

If unsuccessful… trm_Create will return one of these result codes …

2	I/O error
10	Invalid page size
11	Invalid number of key segments
12	Invalid file definition string
13	Invalid key segment position
14	Invalid key segment length
15	Inconsistent key segment definitions
31	File already exists (if OverWrite& was set to a value of 0)

Positioning

There is no positioning information to be affected.

The File Definition String

The first 3 bytes of a file definition string represent instructions to Tsunami…

The first byte must be ASCII character 1 to 8, telling Tsunami what page size you want to use for this new file (from 1K to 8K, in 1K increments).

The second byte must be either ASCII character 1 or 0 (Yes or No) telling Tsunami whether or not you want to compress records in this file. This byte can also be used to instruct Tsunami not to retain it's unique record IDs during file rebuilds (see page 20 for more info).

The third byte must be ASCII character 1 to 128, telling Tsunami how many key segment definitions will follow in the FileDef$ string.

```
FileDef$ = Chr(PageSize&) & _
           Chr(Compress&) & _
           Chr(Segments&)
```

trm_Create (14) (continued)

The File Definition String (continued)

Immediately following the first three bytes, you must define the first key segment definition block. Each key segment definition block consists of 30 bytes and starts with a 25 byte description of the key segment for future reference. You aren't required to include a key segment description, but the 25 bytes must be there, whether blank or not…

Example:
```
Dim SegName As String * 25
SegName = "Customer Name"
```

The first byte following the key segment description must be ASCII character 1 to 128, defining which key this segment belongs to (subject to page size limits on key segments… see page 2). Tsunami allows multi-segmented keys, so if two or more consecutive key segments have the same key number, Tsunami will understand that those key segments are to be concatenated into a single key…

Example:
```
KeyNo& = 1
Chr(KeyNo&)
```

The next two bytes must be a 2-byte string representation of the key segment's starting position within the records to be stored in this file. These two bytes must equate to a number between 1 and 65,535 since a key segment must begin within the first 65,535 bytes of any record…

Example:
```
SegPos& = 7
Chr(SegPos& Mod 256) & _
Chr(SegPos& \ 256)
```

The next byte must be ASCII character 1 to 255, representing the length of this key segment…

Example:
```
SegLen& = 43
Chr(SegLen&)
```

The next byte represents the "key flags" for this key segment. This "key flags" byte is a bit-mask used to determine if the keys in this index are going to be case-sensitive, if duplicate keys will be allowed, if key compression is to be used and if the key segment is binary rather than text. The bit-mask is formed by adding none, some or all of the following four constants…

CASE_SENSITIVE • NO_DUPLICATES • NO_COMPRESSION • BINARY_KEY

Example:
```
KeyFlags& = NO_DUPLICATES + NO_COMPRESSION
Chr(KeyFlags&)
```

Combining all of the elements that were just described, a complete key segment definition block is added to the file definition string as follows…

```
FileDef$ = FileDef$ & SegName & _
           Chr(KeyNo&) & _
           Chr(SegPos& Mod 256) & _
           Chr(SegPos& \ 256) & _
           Chr(SegLen&) & _
           Chr(KeyFlags&)
```

trm_Create (14) (continued)

The File Definition String (continued)

It should be noted if you leave the KeyFlags& variable set to 0 (none of Tsunami's four key flag constants are applied) then this index will contain keys with Tsunami's standard attributes… case-insensitive, duplicates allowed, key compression will be used and the keys are text values (non-binary).

You may add up to 128 key segment definition blocks (30 bytes each) to a file definition string, subject to page size limits on key segments. If you make any mistakes creating the file definition string, Tsunami will return a result code indicating an error condition. For a complete description of the error conditions that could be returned by trm_Create, see result code 12 in the RESULT CODES list in Appendix "C" near the end of this document.

If two or more key segment definition blocks share the same key number, they must be added to the file definition string in the order you want them to be concatenated to form the key.

The key numbers assigned in key segment definition blocks must appear in a file definition string in natural sequential order. In other words, all key segment definition blocks for key #1 must come first, any key segment definition blocks for key #2 must come next, and so on. Key numbers should start with #1 and gaps in key numbers are not allowed. All key segment definition blocks in a multi-segmented key must have matching key flags.

Binary Keys

If the BINARY_KEY key flag is applied, Tsunami will perform certain functions internally to ensure proper sorting of the binary key segments. Binary key segments can be either of the following two types of binary variables… a 2-byte INTEGER or a 4-byte LONG INTEGER. Both are "signed" values that will hold either negative or positive values. When defining the segment, it must be one of those two lengths (2 or 4 bytes). If the binary segment is defined with any other length, trm_Create will return an "Invalid key segment length" result code (14).

By it's very nature, any binary key segment could easily contain the ASCII character 32. Tsunami's key compression feature shouldn't be used with binary keys, as it would strip any leading or trailing blank spaces (ASCII character 32) from the key, and would compact any run of contiguous blank spaces into a single blank space. This could obviously change the key segment's length and it's value… not good.

For this reason, whenever the BINARY_KEY key flag is applied during the definition of a key segment, Tsunami also forces the NO_COMPRESSION key flag internally. Therefore, you don't need to specify the NO_COMPRESSION key flag for a binary key segment. It certainly won't hurt anything if you do, but it simply isn't necessary.

Tsunami's standard key attributes include case-insensitivity, meaning all alpha characters in a key are converted to upper case before they are inserted in an index. Binary keys may also contain the same ASCII codes as lower case alpha characters which would be incorrectly changed to upper case unless the CASE_SENSITIVE key flag is applied to the binary key

trm_Create (14) (continued)

Binary Keys (continued)

segment. Therefore, Tsunami forces the CASE_SENSITIVE key flag internally for all binary key segments… you don't need to specify it when defining a binary key segment. Again, it won't hurt anything if you do, but it simply isn't necessary.

In other words, these two key flag definitions are identical…

```
Chr(BINARY_KEY)

Chr(BINARY_KEY + NO_COMPRESSION + CASE_SENSITIVE)
```

Because a binary key segment can be part of a multi-segmented key that also contains text segments, there is one important restriction Tsunami must enforce. Since key compression is applied to each key as a whole, multi-segmented keys that contain one or more binary segments require special attention. All non-binary segments of such a multi-segmented key require the NO_COMPRESSION key flag in order to be consistent with the binary segment(s) in the key. If they are not so defined, trm_Create returns an "Inconsistent key segment definitions" result code (15).

Unique Record IDs

A unique ID is assigned to every record inserted into a Tsunami file. This record ID is used by Tsunami internally to tie index keys to records and is also used as the first 4 bytes of any record pointer returned by Tsunami's trm_GetPosition function. These unique record IDs are "static"… they're intended to remain unchanged for the life of the record, and beyond. If a record is deleted, it's unique ID will never be reused by a subsequent insertion.

Unique record IDs are even preserved during a file rebuild. The record IDs are carried over into the rebuilt file along with the records. However, if you don't want the record IDs preserved when you run trm_Rebuild, define your Tsunami file with the REBUILD_IDS equate applied to the second byte of the file definition string as follows…

```
FileDef$ = Chr(PageSize&) & _
           Chr(Compress& + REBUILD_IDS) & _
           Chr(Segments&)
```

Tsunami will then know it should not carry the existing record IDs into the newly rebuilt file, assigning new unique IDs to the records as it rebuilds the file.

The permanent nature of Tsunami's unique record IDs makes it possible for you to "use" those IDs for various purposes within your applications. You can depend on them to remain static. To obtain a record's unique ID, simply call trm_GetPosition and convert the first 4 bytes of the 12-byte record pointer into a long integer.

If you do intend to use Tsunami's unique record IDs in your applications, be extremely careful not to rebuild the file with REBUILD_IDS applied, or the IDs will most certainly change in the process.

trm_CurrKeyPos (45)

trm_CurrKeyPos returns the approximate position of the logical current record in the current key path. This function is designed for use with scroll bars and listview controls. Once a logical current record is established, calling this function will tell you approximately where that record resides in the current key path. Simply divide that value by the number of records in the file to arrive at a percentage that can be applied against a scroll bar's maximum range to determine the correct thumb position.

Note: This function is designed for use with Tsunami files that contain too many records to load into a listview at once. It's not recommended to use this function with small files (less than 5,000 records, or so)… the accuracy of the approximation might not be acceptable.

Syntax

- **String-based API**

```
RecPos& = trm_CurrKeyPos(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 45 ' trm_CurrKeyPos

Result& = trm(Op&, hFile&, 0, 0, 0, 0, KeyNo&)
```

The approximate record position is returned in the KeyNo& parameter.

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 45 ' trm_CurrKeyPos
Tsu.file = hFile&

Result& = trm_udt(Tsu)
```

The approximate record position is returned in the Tsu.keyno element.

The approximate position returned by trm_CurrKeyPos is calculated using the location of the key page holding the current record's key, the total number of key pages in the current key path and the total number of keys in the current key path. Therefore, the returned value will always be most accurate when population of the file's key pages is fairly even. Remember that the returned value is an approximation, which is normally adequate for use with scroll bars and listview controls. Do not depend on the returned value to be the exact position of the current record in the current key path.

It should be noted that using trm_CurrKeyPos to access a Tsunami file in single-user mode across a network connection has little impact on performance. However, using this function in multi-user mode does. Tsunami has to check the target file for possible changes made by other users between each access in order to maintain the integrity of it's results.

trm_CurrKeyPos (45) (continued)

Prerequisites

The designated file must have been opened by the trm_Open function and a logical current record must have been established by a successful call to trm_Insert, trm_Update or a trm_Get function.

Results

If successful… trm_CurrKeyPos will return a value of 1 or greater, representing the logical current record's approximate position in the current key path for the designated Tsunami file.

If unsuccessful… trm_CurrKeyPos will return one of these result codes (Note: When unsuccessful, the string-based API for this function returns the result code as a negative) …

2	I/O error
3	File not open
7	File corrupt
8	No current position
99	Time-out

Positioning

This function has no affect on positioning information.

trm_Delete (4)

The trm_Delete function removes the logical current record from the designated Tsunami file.

Syntax

- **String-based API**

```
Result& = trm_Delete(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 4 ‘ trm_Delete

Result& = trm(Op&, hFile&, 0, 0, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 4 ‘ trm_Delete
Tsu.file = hFile&

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function and a logical current record must have been established by a successful call to trm_Insert, trm_Update or any of the trm_Get functions.

Results

If successful… trm_Delete will return a value of 0 and the current record (and all related keys) will be permanently removed from the file.

If unsuccessful… trm_Delete will return one of these result codes …

2	I/O error
3	File not open
7	File corrupt
8	No current position
22	Lost record position
46	Access to file denied
99	Time-out

trm_Delete (4) (continued)

Positioning

Logical current record positioning information for the designated file is cleared. However, position information for the logical next record (if any) and the logical previous record (if any) remain intact. Physical positioning information is discarded.

trm_FileIsOpen (16)

The trm_FileIsOpen function tells you whether or not the designated file handle is assigned to a Tsunami file that is currently opened by your application.

Syntax

- **String-based API**

```
Result& = trm_FileIsOpen(hFile&)
```

… where hFile& holds the file handle you want to test for open status.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 16 ' trm_FileIsOpen

Result& = trm(Op&, hFile&, 0, 0, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 16 ' trm_FileIsOpen
Tsu.file = hFile&

Result& = trm_udt(Tsu)
```

Prerequisites

There are no prerequisites for this function call.

Results

trm_FileIsOpen will return a value of 1 if the file handle is valid and assigned to a Tsunami file that is currently opened by your application.

trm_FileIsOpen will return a value of 0 if the file handle is not assigned to a Tsunami file that is currently opened by your application.

Positioning

This function has no affect on positioning information.

trm_FileSize (18)

The trm_FileSize function will return the file size (in KBytes) of the designated Tsunami file. Multiply the return value by 1024 to determine the file size in bytes.

Syntax

- **String-based API**

```
FSize& = trm_FileSize(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 18 ' trm_FileSize

Result& = trm(Op&, hFile&, 0, 0, 0, 0, KeyNo&)
```

The file size (in KBytes) is returned in the KeyNo& parameter.

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 18 ' trm_FileSize
Tsu.file = hFile&

Result& = trm_udt(Tsu)
```

The file size (in KBytes) is returned in the Tsu.keyno element.

Prerequisites

The designated file must have been opened by the trm_Open function.

Results

If successful… trm_FileSize will return a value representing the current file size in Kbytes. Even though a Tsunami file can potentially grow to a maximum size of 16 terabytes (when using an 8K page size) 2 terabytes is the maximum file size that can be reported.

If unsuccessful… trm_FileSize will return one of these result codes (Note: When unsuccessful, the string-based API for this function returns the result code as a negative) …

3	File not open
99	Time-out

trm_FileSize (18) (continued)

Positioning

This function has no affect on positioning information.

trm_Flush (29)

The trm_Flush function will force the header records for all open Tsunami files (and all disk writes being cached by the operating system) to be physically flushed to disk. You only need this function when accessing files in single-user mode. It isn't necessary to call this function in multi-user mode, since Tsunami already does these things to maintain multi-user file integrity.

Note: trm_Flush has no affect on any cache created and maintained by trm_Accelerate.

Syntax

- **String-based API**

```
Result& = trm_Flush
```

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 29 ' trm_Flush
Result& = trm(Op&, 0, 0, 0, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 29 ' trm_Flush
Result& = trm_udt(Tsu)
```

Prerequisites

Files must have been opened by the trm_Open function.

Results

If successful… trm_Flush will return a value of 0 and the header records for all open Tsunami files (and all disk writes being cached by the operating system) will be forced to disk.

If unsuccessful… trm_Flush will return the following result code …

2 I/O error

Positioning

Positioning information remains unchanged for all open files.

trm_GetByKeyPos (44)

trm_GetByKeyPos allows you to retrieve a record from a Tsunami file by simply designating it's logical position in the current key path. This function is designed for use with scroll bars and listview controls. The position of a scroll bar's thumb can be used to calculate the value passed to this function in order to retrieve a record that corresponds with the thumb's relative position within the scroll bar. Since this establishes a logical current record, subsequent calls to trm_GetNext or trm_GetPrev could then be made to retrieve additional records, if needed.

Syntax

- **String-based API**

```
Record$ = trm_GetByKeyPos(hFile&, KeyPos&)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where KeyPos& represents a long integer variable holding a value from 1 to the total number of records in the designated Tsunami file (the logical position in the current key path for the record you want to retrieve).

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 44 ' trm_GetByKeyPos
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyNo& = KeyPos&

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 44 ' trm_GetByKeyPos
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyno = KeyPos&

Result& = trm_udt(Tsu)
```

The record returned by trm_GetByKeyPos is one that is located "near" the designated position in the current key path. Tsunami calculates an approximate position using the KeyPos& value, the total number of keys in the current key path and the total number of key pages in the current key path. Therefore, the returned record will be most accurate when population of the file's key pages is fairly even. Remember that the returned record is the result of an approximation, which is normally adequate for use with scroll bars and listview controls. Do not depend on the returned record being from the exact position that you requested in the current key path.

trm_GetByKeyPos (44) (continued)

Syntax (continued)

It should be noted that using trm_GetByKeyPos to access a Tsunami file in single-user mode across a network connection has little impact on performance. However, using this function in multi-user mode does. Tsunami has to check the target file for possible changes made by other users between each access in order to maintain the integrity of it's results.

Key-Only Search

This function supports key-only searches using Tsunami's KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetByKeyPos(hFile& + KEY_ONLY, KeyPos&)
```

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 44 + KEY_ONLY ' trm_GetByKeyPos
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 44 + KEY_ONLY ' trm_GetByKeyPos
```

Prerequisites

The designated file must have been opened by the trm_Open function and a current key path must have been established.

Results

If successful… trm_GetByKeyPos will return the desired record to be stored in a string variable or UDT.

If unsuccessful… trm_GetByKeyPos will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
4	Key not found
7	File corrupt
50	Data buffer too small
99	Time-out

trm_GetByKeyPos (44) (continued)

Results (continued)

If trm_GetByKeyPos returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetByKeyPos function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_GetDirect (23)

The trm_GetDirect function allows you to use a 12-byte record pointer obtained from the trm_GetPosition function to retrieve a record directly.

Syntax

- **String-based API**

```
Record$ = trm_GetDirect(hFile&, RecPtr$)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where RecPtr$ represents a string variable holding a valid 12-byte record pointer returned by a call to the trm_GetPosition function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 23 ‘ trm_GetDirect
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyPtr& = StrPtr(RecPtr$)
KeyLen& = Len(RecPtr$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, KeyPtr&, KeyLen&, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 23 ‘ trm_GetDirect
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyptr = StrPtr(RecPtr$)
Tsu.keylen = Len(RecPtr$)

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami’s KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetDirect(hFile& + KEY_ONLY, RecPtr$)
```

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 23 + KEY_ONLY ‘ trm_GetDirect
```

trm_GetDirect (23) (continued)

Key-Only Search (continued)

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 23 + KEY_ONLY ' trm_GetDirect
```

Prerequisites

The designated file must have been opened by the trm_Open function, a current key path must have been established and the 12-byte record pointer must have been returned by a successful call to the trm_GetPosition function.

Results

If successful… trm_GetDirect will return the desired record to be stored in a string variable or UDT.

If unsuccessful… trm_GetDirect will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
21	Invalid record pointer
22	Lost record position
50	Data buffer too small
99	Time-out

If trm_GetDirect returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetDirect function is immediately established as the logical current record as well as the physical current record for the designated file.

Note: If a current key path has not yet been established before calling trm_GetDirect, Tsunami will set key path #1 as the current key path and internal positioning information will be based on that index. If that's not acceptable to you, make a call to trm_SetKeyPath prior to trm_GetDirect in order to establish the key path you desire.

trm_GetEqual (5)

The trm_GetEqual function will search the designated file using the designated key path for an exact match of the supplied key value.

Syntax

- **String-based API**

```
Record$ = trm_GetEqual(hFile&, KeyNo&, KeyVal$)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where KeyNo& holds a valid key path number (1–128) for this file.

… where KeyVal$ represents a string variable holding the key value that you want Tsunami to search for.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 5 ' trm_GetEqual
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyPtr& = StrPtr(KeyVal$)
KeyLen& = Len(KeyVal$)
KeyNo& = 1

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, KeyPtr&, KeyLen&, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 5 ' trm_GetEqual
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyptr = StrPtr(KeyVal$)
Tsu.keylen = Len(KeyVal$)
Tsu.keyno = 1

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami's KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetEqual(hFile& + KEY_ONLY, KeyNo&, KeyVal$)
```

trm_GetEqual (5) (continued)

Key-Only Search (continued)

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 5 + KEY_ONLY ‘ trm_GetEqual
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 5 + KEY_ONLY ‘ trm_GetEqual
```

Prerequisites

The designated file must have been opened by the trm_Open function and the designated key path must be valid for this file.

Results

If successful… trm_GetEqual will return the desired record to be stored in a string variable or UDT.

If unsuccessful… trm_GetEqual will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
4	Key not found
6	Invalid key number
7	File corrupt
50	Data buffer too small
99	Time-out

If trm_GetEqual returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetEqual function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_GetEqualOrGreater (9)

The trm_GetEqualOrGreater function will search the designated file using the designated key path for an exact match of the supplied key value. If an exact match doesn't exist, it will return the record that has the key value closest to (yet greater than) the value being searched for (if any exists).

Syntax

- **String-based API**

```
Record$ = trm_GetEqualOrGreater(hFile&, KeyNo&, KeyVal$)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where KeyNo& holds a valid key path number (1–128) for this file.

… where KeyVal$ represents a string variable holding the key value that you want Tsunami to search for.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 9 ' trm_GetEqualOrGreater
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyPtr& = StrPtr(KeyVal$)
KeyLen& = Len(KeyVal$)
KeyNo& = 1

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, KeyPtr&, KeyLen&, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 9 ' trm_GetEqualOrGreater
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyptr = StrPtr(KeyVal$)
Tsu.keylen = Len(KeyVal$)
Tsu.keyno = 1

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami's KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetEqualOrGreater(hFile& + KEY_ONLY, KeyNo&, KeyVal$)
```

trm_GetEqualOrGreater (9) (continued)

Key-Only Search (continued)

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 9 + KEY_ONLY ‘ trm_GetEqualOrGreater
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 9 + KEY_ONLY ‘ trm_GetEqualOrGreater
```

Prerequisites

The designated file must have been opened by the trm_Open function and the designated key path must be valid for this file.

Results

If successful… trm_GetEqualOrGreater will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_GetEqualOrGreater will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
6	Invalid key number
7	File corrupt
9	End of file
50	Data buffer too small
99	Time-out

If trm_GetEqualOrGreater returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetEqualOrGreater function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_GetEqualOrLess (11)

The trm_GetEqualOrLess function will search the designated file using the designated key path for an exact match of the supplied key value. If an exact match doesn't exist, it will return the record with the key value closest to (yet less than) the value being searched for (if any exists).

Syntax

- **String-based API**

```
Record$ = trm_GetEqualOrLess(hFile&, KeyNo&, KeyVal$)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where KeyNo& holds a valid key path number (1–128) for this file.

… where KeyVal$ represents a string variable holding the key value that you want Tsunami to search for.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 11 ' trm_GetEqualOrLess
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyPtr& = StrPtr(KeyVal$)
KeyLen& = Len(KeyVal$)
KeyNo& = 1

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, KeyPtr&, KeyLen&, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 11 ' trm_GetEqualOrLess
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyptr = StrPtr(KeyVal$)
Tsu.keylen = Len(KeyVal$)
Tsu.keyno = 1

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami's KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetEqualOrLess(hFile& + KEY_ONLY, KeyNo&, KeyVal$)
```

trm_GetEqualOrLess (11) (continued)

Key-Only Search (continued)

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 11 + KEY_ONLY ‘ trm_GetEqualOrLess
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 11 + KEY_ONLY ‘ trm_GetEqualOrLess
```

Prerequisites

The designated file must have been opened by the trm_Open function and the designated key path must be valid for this file.

Results

If successful… trm_GetEqualOrLess will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_GetEqualOrLess will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
6	Invalid key number
7	File corrupt
9	End of file
50	Data buffer too small
99	Time-out

If trm_GetEqualOrLess returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetEqualOrLess function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_GetFileDef (36)

The trm_GetFileDef function provides a way to retrieve a file definition string from the header of an open Tsunami file. That file definition string can then be used to create another Tsunami file with identical structure (a clone) by using it in a subsequent call to the trm_Create function, or it could also be used to simply determine how the designated Tsunami file was originally defined, including the key segment definition blocks.

Note: A complete description of Tsunami's file definition string can be found beginning on page 17 (in the trm_Create section).

Syntax

- **String-based API**

```
FileDef$ = trm_GetFileDef(hFile&)
Result&  = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 36 ' trm_GetFileDef
DataPtr& = StrPtr(FileDef$)
DataLen& = Len(FileDef$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 36 ' trm_GetFileDef
Tsu.file = hFile&
Tsu.dataptr = StrPtr(FileDef$)
Tsu.datalen = Len(FileDef$)

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function.

Results

If successful… trm_GetFileDef will return a file definition string.

If unsuccessful… trm_GetFileDef will return a null string and the following result code. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

3 File not open

trm_GetFileDef (36) (continued)

Positioning

This function has no affect on positioning information.

After a successful call to trm_GetFileDef, the following code will break down the information contained in the file definition string…

```
' define a UDT

Type KeySegment
   Descrip As String * 25
   KeyNo As Long
   SegPos As Long
   SegLen As Long
   KeyFlags As Long
End Type



' elsewhere in your code ...

FDef$ = trm_GetFileDef(hFile&)

If Len(FDef$) Then

   PageSize& = Asc(Mid(FDef$, 1, 1))      ' 1K - 8K
   Compress& = (Asc(Mid(FDef$, 2)) AND 1) ' compressed or uncompressed
   Preserve& = (Asc(Mid(FDef$, 2)) AND 2) ' preserve record IDs during rebuilds?
   Segments& = Asc(Mid(FDef$, 3, 1))      ' 1 - 128

   Dim KeySeg(1 To Segments&) As KeySegment

   For Seg& = 1 To Segments&
      KeySeg(Seg&).Descrip = Mid(FDef$,  4 + (Seg& - 1) * 30, 25)
      KeySeg(Seg&).KeyNo = Asc(Mid(FDef$, 29 + (Seg& - 1) * 30, 1))
      KeySeg(Seg&).SegPos = Asc(Mid(FDef$, 30 + (Seg& - 1) * 30, 1)) + _
                            Asc(Mid(FDef$, 31 + (Seg& - 1) * 30, 1)) * 256
      KeySeg(Seg&).SegLen = Asc(Mid(FDef$, 32 + (Seg& - 1) * 30, 1))
      KeySeg(Seg&).KeyFlags = Asc(Mid(FDef$, 33 + (Seg& - 1) * 30, 1))
   Next

End If



' if needed, key flags can be determined for any key segment as follows
' (flags are considered "set" if their values are non-zero)

Seg& = 1
CaseSensitive& = (KeySeg(Seg&).KeyFlags And 1)
NoDuplicates& = (KeySeg(Seg&).KeyFlags And 2)
NotCompressed& = (KeySeg(Seg&).KeyFlags And 4)
KeyIsBinary& = (KeySeg(Seg&).KeyFlags And 8)

Seg& = 2
etc ...
```

trm_GetFileVer (25)

The trm_GetFileVer function will tell you which version of Tsunami created the designated Tsunami file.

Syntax

- **String-based API**

```
FileVer$ = trm_GetFileVer(hFile&)
Result&  = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 25 ‘ trm_GetFileVer
DataPtr& = StrPtr(FileVer$)
DataLen& = Len(FileVer$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 25 ‘ trm_GetFileVer
Tsu.file = hFile&
Tsu.dataptr = StrPtr(FileVer$)
Tsu.datalen = Len(FileVer$)

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function.

Results

If successful… trm_GetFileVer will return a 3-byte string. That string holds the major and minor version numbers, separated by a period (eg: 2.5).

If unsuccessful… trm_GetFileVer will return a null string and the following result code. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

3 File not open

Positioning

This function has no affect on positioning information.

trm_GetFirst (12)

The trm_GetFirst function will search the designated file using the designated key path to locate and retrieve the record with the lowest key value in that key path.

Syntax

- **String-based API**

```
Record$ = trm_GetFirst(hFile&, KeyNo&)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where KeyNo& holds a valid key path number (1–128) for this file.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 12 ‘ trm_GetFirst
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyNo& = 1

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 12 ‘ trm_GetFirst
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyno = 1

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami’s KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetFirst(hFile& + KEY_ONLY, KeyNo&)
```

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 12 + KEY_ONLY ‘ trm_GetFirst
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 12 + KEY_ONLY ‘ trm_GetFirst
```

trm_GetFirst (12) (continued)

Prerequisites

The designated file must have been opened by the trm_Open function and the designated key path must be valid for this file.

Results

If successful… trm_GetFirst will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_GetFirst will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
6	Invalid key number
7	File corrupt
9	End of file
50	Data buffer too small
99	Time-out

If trm_GetFirst returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetFirst function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_GetGreater (8)

The trm_GetGreater function will search the designated file using the designated key path to locate and retrieve the first record with a key value that is greater than the supplied key value.

Syntax

- **String-based API**

```
Record$ = trm_GetGreater(hFile&, KeyNo&, KeyVal$)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where KeyNo& holds a valid key path number (1–128) for this file.

… where KeyVal$ represents a string variable holding the key value that you want Tsunami to use in it's search for a greater key value.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 8 ' trm_GetGreater
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyPtr& = StrPtr(KeyVal$)
KeyLen& = Len(KeyVal$)
KeyNo& = 1

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, KeyPtr&, KeyLen&, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 8 ' trm_GetGreater
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyptr = StrPtr(KeyVal$)
Tsu.keylen = Len(KeyVal$)
Tsu.keyno = 1

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami's KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetGreater(hFile& + KEY_ONLY, KeyNo&, KeyVal$)
```

trm_GetGreater (8) (continued)

Key-Only Search (continued)

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 8 + KEY_ONLY ‘ trm_GetGreater
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 8 + KEY_ONLY ‘ trm_GetGreater
```

Prerequisites

The designated file must have been opened by the trm_Open function and the designated key path must be valid for this file.

Results

If successful… trm_GetGreater will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_GetGreater will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
6	Invalid key number
7	File corrupt
9	End of file
50	Data buffer too small
99	Time-out

If trm_GetGreater returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetGreater function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_GetLast (13)

The trm_GetLast function will search the designated file using the designated key path to locate and retrieve the record with the highest key value in that key path.

Syntax

- **String-based API**

```
Record$ = trm_GetLast(hFile&, KeyNo&)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where KeyNo& holds a valid key path number (1–128) for this file.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 13 ‘ trm_GetLast
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyNo& = 1

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 13 ‘ trm_GetLast
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyno = 1

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami’s KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetLast(hFile& + KEY_ONLY, KeyNo&)
```

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 13 + KEY_ONLY ‘ trm_GetLast
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 13 + KEY_ONLY ‘ trm_GetLast
```

trm_GetLast (13) (continued)

Prerequisites

The designated file must have been opened by the trm_Open function and the designated key path must be valid for this file.

Results

If successful… trm_GetLast will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_GetLast will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
6	Invalid key number
7	File corrupt
9	End of file
50	Data buffer too small
99	Time-out

If trm_GetLast returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetLast function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_GetLess (10)

The trm_GetLess function will search the designated file using the designated key path to locate and retrieve the first record with a key value that is less than the supplied key value.

Syntax

- **String-based API**

```
Record$ = trm_GetLess(hFile&, KeyNo&, KeyVal$)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where KeyNo& holds a valid key path number (1–128) for this file.

… where KeyVal$ represents a string variable holding the key value that you want Tsunami to use in it's search for a lower key value.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 10 ' trm_GetLess
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)
KeyPtr& = StrPtr(KeyVal$)
KeyLen& = Len(KeyVal$)
KeyNo& = 1

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, KeyPtr&, KeyLen&, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 10 ' trm_GetLess
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)
Tsu.keyptr = StrPtr(KeyVal$)
Tsu.keylen = Len(KeyVal$)
Tsu.keyno = 1

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami's KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetLess(hFile& + KEY_ONLY, KeyNo&, KeyVal$)
```

trm_GetLess (10) (continued)

Key-Only Search (continued)

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 10 + KEY_ONLY ‘ trm_GetLess
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 10 + KEY_ONLY ‘ trm_GetLess
```

Prerequisites

The designated file must have been opened by the trm_Open function and the designated key path must be valid for this file.

Results

If successful… trm_GetLess will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_GetLess will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
6	Invalid key number
7	File corrupt
9	End of file
50	Data buffer too small
99	Time-out

If trm_GetLess returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetLess function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_GetNext (6)

The trm_GetNext function will retrieve the logical next record in sequence using the current key path in the designated file.

Syntax

- **String-based API**

```
Record$ = trm_GetNext(hFile&)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 6 ' trm_GetNext
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 6 ' trm_GetNext
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami's KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetNext(hFile& + KEY_ONLY)
```

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 6 + KEY_ONLY ' trm_GetNext
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 6 + KEY_ONLY ' trm_GetNext
```

trm_GetNext (6) (continued)

Prerequisites

The designated file must have been opened by the trm_Open function and a logical current record must have been established.

Results

If successful… trm_GetNext will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_GetNext will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
7	File corrupt
8	No current position
9	End of file
22	Lost record position
50	Data buffer too small
99	Time-out

If trm_GetNext returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetNext function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_GetPosition (22)

The trm_GetPosition function returns a 12-byte record pointer to the current record, the previous record or the next record in the designated file. This record pointer can be used in a subsequent call to trm_GetDirect to retrieve that record directly. There is no disk I/O involved in a trm_GetPosition call.

Syntax

- **String-based API**

```
RecPtr$ = trm_GetPosition(hFile&, 0)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where the function's second parameter holds one of the following values…

0 = Return a pointer to the current record
-1 = Return a pointer to the previous record (if any)
1 = Return a pointer to the next record (if any)

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 22 ' trm_GetPosition
DataPtr& = StrPtr(RecPtr$)
DataLen& = Len(RecPtr$)
KeyNo& = 0

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 22 ' trm_GetPosition
Tsu.file = hFile&
Tsu.dataptr = StrPtr(RecPtr$)
Tsu.datalen = Len(RecPtr$)
Tsu.keyno = 0

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function and a current record (logical or physical) must have been established.

trm_GetPosition (22) (continued)

Results

If successful… trm_GetPosition will return a 12-byte record pointer.

If unsuccessful… trm_GetPosition returns a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

3 File not open
8 No current position
9 End of file

If trm_GetPosition returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

This function has no affect on positioning information.

Unique Record IDs

The first 4 bytes of any 12-byte record pointer returned by a successful call to trm_GetPosition represents a long integer used internally by Tsunami as that record's unique record ID. That record ID is "static". It will remain unchanged for the life of the record and beyond. If a record is deleted, it's unique record ID is never reused by subsequent record insertions.

The only time record IDs are ever changed is during a file rebuild when the REBUILD_IDS equate has been applied to the second byte of the file definition string (see page 20).

trm_GetPrev (7)

The trm_GetPrev function will retrieve the logical previous record in sequence using the current key path in the designated file.

Syntax

- **String-based API**

```
Record$ = trm_GetPrev(hFile&)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 7 ‘ trm_GetPrev
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 7 ‘ trm_GetPrev
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)

Result& = trm_udt(Tsu)
```

Key-Only Search

This function supports key-only searches using Tsunami’s KEY_ONLY constant (see page 9 for more details).

- **String-based API**

```
RetVal$ = trm_GetPrev(hFile& + KEY_ONLY)
```

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 7 + KEY_ONLY ‘ trm_GetPrev
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 7 + KEY_ONLY ‘ trm_GetPrev
```

trm_GetPrev (7) (continued)

Prerequisites

The designated file must have been opened by the trm_Open function and a logical current record must have been established.

Results

If successful… trm_GetPrev will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_GetPrev will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
7	File corrupt
8	No current position
9	End of file
22	Lost record position
50	Data buffer too small
99	Time-out

If trm_GetPrev returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_GetPrev function is immediately established as the logical current record as well as the physical current record for the designated file.

trm_Insert (2)

trm_Insert will insert the supplied record and it's keys into the designated Tsunami file.

Syntax

- **String-based API**

```
Result& = trm_Insert(hFile&, Record$)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where Record$ represents a string variable holding the record to be inserted.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 2 ' trm_Insert
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 2 ' trm_Insert
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function and the Record$ variable must hold a string long enough to satisfy the key extraction definitions for this file.

Results

If successful… trm_Insert will insert the record and it's keys into the designated file, returning a result code of 0.

If unsuccessful… trm_Insert will return one of these result codes …

2	I/O error
3	File not open
5	Duplicate key
20	Invalid record length
46	Access to file denied
99	Time-out

trm_Insert (2) (continued)

Positioning

Any record successfully inserted into a Tsunami file by the trm_Insert function is immediately established as the logical current record as well as the physical current record for the designated file.

Note: If a current key path has not yet been established before calling trm_Insert, Tsunami will set key path #1 as the current key path and internal positioning information will be based on that index. If that's not acceptable to you, make a call to trm_SetKeyPath prior to trm_Insert in order to establish the key path you desire.

trm_Integrity (37)

The trm_Integrity function will temporarily lock the designated Tsunami file while performing a high-speed scan, reading every index from beginning to end as well as retrieving (and decompressing) every record to make sure the entire file is readable. For speed considerations, links between keys and records are not verified… the file is simply tested for it's ability to be successfully rebuilt or recovered if need be. It's a good idea to call this function just prior to a backup to make sure you won't be overwriting a good backup with a corrupt file.

Syntax

- **String-based API**

```
Do
   Result& = trm_Integrity(hFile&)
   If Result& <= 0 Then Exit Do
   LabelProgress.Caption = Str(Result&) & " %"
   LabelProgress.Refresh
Loop

‘ Result& is either 0 if successful or < 0 if not successful
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 37 ‘ trm_Integrity

Do
   Result& = trm(Op&, hFile&, 0, 0, 0, 0, KeyNo&)
   If Result& Then Exit Do ‘ error
   If KeyNo& = 0 Then Exit Do ‘ finished
   LabelProgress.Caption = Str(KeyNo&) & " %"
   LabelProgress.Refresh
Loop

‘ Result& is either 0 if successful or > 0 if not successful
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 37 ‘ trm_Integrity
Tsu.file = hFile&

Do
   Result& = trm_udt(Tsu)
   If Result& Then Exit Do ‘ error
   If Tsu.keyno = 0 Then Exit Do ‘ finished
   LabelProgress.Caption = Str(Tsu.keyno) & " %"
   LabelProgress.Refresh
Loop

‘ Result& is either 0 if successful or > 0 if not successful
```

trm_Integrity is one of Tsunami's three "incremental" functions. Incremental functions perform a portion of their operation (a minimum of 1%) and then return a value to your application that

trm_Integrity (37) (continued)

Syntax (continued)

represents a percentage of completion. Your application then calls the function again to either continue or abort the operation. You can use the return values to update the operation's progress on screen with a text display or with a progress bar of some kind.

During an incremental function, the Tsunami file is locked in exclusive mode (if it isn't already) and can't be accessed by any other thread, application or user until the function completes or is aborted by the user.

Note: If you'd like to provide the ability to abort a trm_Integrity function call, execute it within a thread and change the value of the file handle variable to 0 before the operation has completed. Tsunami will take that as a signal that you want to abort the integrity check.

Prerequisites

The designated file must have been opened by the trm_Open function. The file can be open in either single-user mode or multi-user mode… it makes no difference to Tsunami.

Results

If successful… trm_Integrity will return a result code of 0.

If unsuccessful… trm_Integrity will return one of these result codes (Note: When unsuccessful, the string-based API for this function returns the result code as a negative) …

2	I/O error
3	File not open
7	File corrupt
99	Time-out

Positioning

This function has no affect on positioning information.

trm_Open (0)

The trm_Open function is used to give your application access to a Tsunami file. The return value from a successful call to trm_Open should be retained in a long integer variable and used as the file handle parameter in any other Tsunami function calls that access the file.

Syntax

- **String-based API**

```
hFile& = trm_Open(File$, 0)
```

… where File$ represents a string variable holding the name of the file to be opened, optionally including a valid path. If you do specify a path, it must be a complete path, including the drive letter.

… where the function's second parameter holds one of the following values…

0 = Open file in single-user, exclusive mode
1 = Open file in multi-user, read-write mode, locked when writing
2 = Open file in multi-user, read-only mode, no locking

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 0 ‘ trm_Open
KeyPtr& = StrPtr(File$)
KeyLen& = Len(File$)
KeyNo& = 0 ‘ single-user

Result& = trm(Op&, hFile&, 0, 0, KeyPtr&, KeyLen&, KeyNo&)
```

If successful, a file handle is returned in the hFile& parameter.

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 0 ‘ trm_Open
Tsu.keyptr = StrPtr(File$)
Tsu.keylen = Len(File$)
Tsu.keyno = 0 ‘ single-user

Result& = trm_udt(Tsu)
```

If successful, a file handle is returned in the Tsu.file element.

Prerequisites

The designated file must have been created by Tsunami's trm_Create function.

trm_Open (0) (continued)

Results

If successful… trm_Open will assign a file handle… a value greater than 0.

If unsuccessful… trm_Open will return one of these result codes (Note: When unsuccessful, the string-based API for this function returns the result code as a negative) …

1	Not a Tsunami file
2	I/O error
30	Access denied
32	No more file handles
33	Max files open
99	Time-out

Positioning

No positioning information is established when a Tsunami file is first opened for access.

trm_Rebuild (38)

The trm_Rebuild function allows you to rebuild a Tsunami file to change it's file definition, change the file's key segment definitions or to simply make the file as compact as possible if a large number of records have been deleted and you don't expect any more inserts to occur that would re-use the vacated space in the file. trm_Rebuild requires a valid header page. If the file's header page is damaged, use trm_Recover to salvage the records from the damaged file.

Syntax

- **String-based API**

```
Do
   Result& = trm_Rebuild(File$, SaveFile$, FileDef$)
   If Result& <= 0 Then Exit Do
   LabelProgress.Caption = Str(Result&) & " %"
   LabelProgress.Refresh
Loop

Records& = Abs(Result&)
```

… where File$ represents a string variable holding the name of the Tsunami file to be rebuilt, optionally including a valid path. If you do specify a path, it must be a complete path, including the drive letter.

… where SaveFile$ represents a string variable holding the file name you want the original Tsunami file saved under, optionally including a valid path. If you include a path in the SaveFile$ variable, it can be different than the original file's path. If SaveFile$ is passed as a null string, the original Tsunami file will be erased after a successful rebuild.

… where FileDef$ respresents a string variable holding the information required by Tsunami to rebuild the data file. You can find a detailed description of the file definition string in the trm_Create section of this document, starting on page 17. If FileDef$ is passed as a null string, Tsunami will assume you want to use the existing file definition string (stored in the file's header record) to rebuild the file with no changes.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 38 ' trm_Rebuild
DataPtr& = StrPtr(FileDef$)
DataLen& = Len(FileDef$)
KeyPtr& = StrPtr(File$)
KeyLen& = Len(File$)
KeyNo& = 0

Do
   Result& = trm(Op&, 0, DataPtr&, DataLen&, KeyPtr&, KeyLen&, KeyNo&)
   If Result& = 0 Then Exit Do
   LabelProgress.Caption = Str(Result&) & " %"
   LabelProgress.Refresh
Loop

Records& = KeyNo&
```

trm_Rebuild (38) (continued)

Syntax (continued)

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 38 ' trm_Rebuild
Tsu.file = 0
Tsu.dataptr = StrPtr(FileDef$)
Tsu.datalen = Len(FileDef$)
Tsu.keyptr = StrPtr(File$)
Tsu.keylen = Len(File$)
Tsu.keyno = 0

Do
   Result& = trm_udt(Tsu)
   If Result& = 0 Then Exit Do
   LabelProgress.Caption = Str(Result&) & " %"
   LabelProgress.Refresh
Loop

Records& = Tsu.keyno
```

trm_Rebuild is one of Tsunami's three "incremental" functions. Incremental functions perform a portion of their operation (a minimum of 1%) and then return a value to your application representing a percentage of completion. Your application then calls the function again to either continue or abort the operation. You can use the return values to update the operation's progress on screen with a text display or with a progress bar of some kind.

During an incremental function, the Tsunami file is locked in exclusive mode (if it isn't already) and can't be accessed by any other thread, application or user until the function completes or is aborted by the user.

Note: If you'd like to provide the ability to abort a trm_Rebuild function call, execute it within a thread and change the File$ variable to a null string before the operation has completed. In the pointer-based APIs, set the KeyLen& parameter or Tsu.keylen element to zero. Tsunami will take these as signals that you want to abort the rebuild operation.

Prerequisites

The Tsunami file must not be open prior to this call.

Results

Once the rebuild operation is finished, the total count of records in the rebuilt file is returned in the KeyNo parameter. However, in the string-based API, the number of records contained in the rebuilt file is returned as a negative value in the Result variable. Use the Abs function to convert it to a positive number.

trm_Rebuild (38) (continued)

Results (continued)

The file is rebuilt using a temporary file in the same directory on the same drive as the original file. Tsunami does not check the drive for adequate free space before performing the rebuild, since it has no way of knowing in advance how much room the new file might require. If Tsunami runs out of room, the rebuild will abort and the temporary file will be erased.

If trm_Rebuild is unsuccessful for any reason, your application will receive a result code of 0, indicating that the file was not rebuilt. The original file will remain intact exactly as it was before the rebuild operation began.

Positioning

This function has no affect on positioning information.

Note: The string-based API for trm_Rebuild allows you to instruct Tsunami to save the original file by passing a SaveFile$ parameter. The pointer-based versions of this function do not give you an option to save the original. If you'd like to save the original when using trm_Rebuild in a pointer-based API, you must make a copy of the original file yourself before starting the rebuild.

trm_Recover (39)

The trm_Recover function allows you to retrieve the records from a Tsunami file, and load them into a “recovery” file. This function can be used even if the Tsunami file or it’s header page is corrupt, in which case it will recover as many records as possible. The records are retrieved in physical order.

Syntax

- **String-based API**

```
Do
   Result& = trm_Recover(SourceFile$, TargetFile$, PageSize&, Compress&)
   If Result& <= 0 Then Exit Do
   LabelProgress.Caption = Str(Result&) & " %"
   LabelProgress.Refresh
Loop

RecoveredRecords& = Abs(Result&)
```

… where SourceFile$ represents a string variable holding the name of the Tsunami file to be opened for recovery, optionally including a valid path. If you do specify a path, it must be a complete path, including the drive letter.

… where TargetFile$ represents a string variable holding the name of the text file to receive the recovered records, optionally including a valid path. If you do specify a path, it must be a complete path, including the drive letter.

… where PageSize& holds a value from 1 to 8 representing the page size for the Tsunami file expressed in KBytes… 1=1K, 2=2K, etc. (or a value of –1 if you want Tsunami to determine the page size for you *).

… where Compress& holds a value of 1 or 0 (Yes or No) to indicate whether or not compression was used on the records being recovered (or a value of –1 if you want Tsunami to determine if compression was used *).

* Note: Tsunami can’t determine page size or compression if the header page is damaged.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 39 ‘ trm_Recover
hFile& = PageSize&
DataPtr& = StrPtr(TargetFile$)
DataLen& = Len(TargetFile$)
KeyPtr& = StrPtr(SourceFile$)
KeyLen& = Len(SourceFile$)
KeyNo& = Compress&

Do
   Result& = trm(Op&, hFile&, DataPtr&, DataLen&, KeyPtr&, KeyLen&, KeyNo&)
   If Result& = 0 Then Exit Do
   LabelProgress.Caption = Str(Result&) & " %"
   LabelProgress.Refresh
Loop

RecoveredRecords& = KeyNo&
```

trm_Recover (39) (continued)

Syntax (continued)

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 39 ' trm_Recover
Tsu.file = PageSize&
Tsu.dataptr = StrPtr(TargetFile$)
Tsu.datalen = Len(TargetFile$)
Tsu.keyptr = StrPtr(SourceFile$)
Tsu.keylen = Len(SourceFile$)
Tsu.keyno = Compress&

Do
   Result& = trm_udt(Tsu)
   If Result& = 0 Then Exit Do
   LabelProgress.Caption = Str(Result&) & " %"
   LabelProgress.Refresh
Loop

RecoveredRecords& = Tsu.keyno
```

trm_Recover is one of Tsunami's three "incremental" functions. Incremental functions perform a portion of their operation (a minimum of 1%) and then return a value to your application representing a percentage of completion. Your application then calls the function again to either continue or abort the operation. You can use the return values to update the operation's progress on screen with a text display or with a progress bar of some kind.

During an incremental function, the Tsunami file is locked in exclusive mode (if it isn't already) and can't be accessed by any other thread, application or user until the function completes or is aborted by the user.

Note: If you'd like to provide the ability to abort a trm_Recover function call, execute it within a thread and change the SourceFile$ variable to a null string before the operation has completed. In the pointer-based APIs, set the KeyLen& parameter or Tsu.keylen element to zero. Tsunami will take these as signals that you want to abort the recover operation.

Prerequisites

Neither the Tsunami file nor the target file (if it exists) may be open prior to calling this function.

Results

Once the recover operation is finished, the total count of records in the recovery file is returned in the KeyNo variable. However, in the string-based API, the number of records contained in the recovery file is returned as a negative value in the Result variable. Use the Abs function to convert it to a positive number.

trm_Recover (39) (continued)

Results (continued)

The recovered records are output to the target file, overwriting the file if it already exists. Each record in the target file is preceeded by it's length, separated from the first character of the record by a comma. This is done because Tsunami files accept a mixture of fixed length and variable length records that may also contain binary data, in which case you'll need to access your recovered records in binary mode instead of sequentially… you'll need to know how many bytes to read for each variable length record in the file. Either way, each recovered record will be terminated with a carriage return and a line feed. If the records don't contain binary data, then the target file should be compatible with most text editors.

If trm_Recover is unsuccessful for some reason, your application will receive a return value of 0, indicating that no records were recovered. In this event, make sure you're passing valid file names, that the files are not open, that you have passed the correct page size and that the compression variable is set properly for the file you're trying to recover.

Positioning

This function has no affect on positioning information.

This code sample shows how to read records from a Tsunami recovery file in binary mode…

```
hFile& = FreeFile

Err.Clear
Open "C:\My Data\Customers.out" For Binary Access Read As #hFile&

If Err.Number Then

   Msg$ = "Error opening file:" + Str(Err.Number)
   ' display message

Else

   Do

      Get #hFile, 1, Tmp$
      If Tmp$ = "," Then
         Get #hFile&, Val(Length$), Record$

         ' Record$ variable now contains a record

         Get #hFile&, 2, Tmp$ ' Discard the carriage return and line feed
         Length$ = ""
      Else
         Length$ = Length$ & Tmp$
      End If

   Loop While Not Eof(hFile&)

   Close #hFile&

End If
```

trm_Result

The trm_Result function is only needed in Tsunami's string-based API. It's called subsequent to any of the trm_Get or trm_Step functions which return strings instead of numeric result codes. If the string returned by one of these functions is null, call trm_Result immediately to obtain a numeric result code to explain why a null string was returned.

Syntax

- **String-based API**

```
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1**

Not needed in the pointer-based APIs. Pointer-based functions return their own result code.

- **Pointer-based API #2**

Not needed in the pointer-based APIs. Pointer-based functions return their own result code.

Prerequisites

The designated file must have been opened by the trm_Open function.

Results

trm_Result will return a result code indicating the status of the most recent operation performed on the designated file by your application. This result code is only available until another operation is performed by your application on the same file (using the same file handle) at which time it's replaced with the result code of the more recent operation. Therefore, it's advisable to make a call to trm_Result as soon as your application recognizes that one of the trm_Get or trm_Step functions returned a null string.

After you call trm_Result, Tsunami's internal result flag for the designated file is reset to zero.

If you call trm_Result to obtain a result code after one of the trm_Get or trm_Step functions returns a null string and trm_Result returns a result code of 0 (normally meaning success) then you must have passed an invalid file handle to trm_Result.

Positioning

This function has no affect on positioning information.

trm_SetKeyPath (30)

trm_SetKeyPath allows you to change the key path by which trm_GetNext and trm_GetPrev traverse the records in a Tsunami file. You can also call trm_SetKeyPath at any time to set the current key path, regardless of whether or not a current record has been established.

Note: The current key path is also established by any of the trm_Get functions that pass a key number parameter.

Syntax

- **String-based API**

```
Result& = trm_SetKeyPath(hFile&, KeyNo&)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where KeyNo& holds a valid key path number (1–128) for this file.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 30 ' trm_SetKeyPath
KeyNo& = 1

Result& = trm(Op&, hFile&, 0, 0, 0, 0, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 30 ' trm_SetKeyPath
Tsu.file = hFile&
Tsu.keyno = 1

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function.

Results

If successful… trm_SetKeyPath will return a result code of 0.

If unsuccessful… trm_SetKeyPath will return one of these result codes …

2	I/O error
3	File not open
6	Invalid key number
99	Time-out

trm_SetKeyPath (30) (continued)

Positioning

If a logical current record has been established prior to a trm_SetKeyPath call, that record will remain the current record. However, the logical next and previous records will change to correspond with the new key path you designated. trm_SetKeyPath does not establish a new current record.

trm_StepFirst (33)

The trm_StepFirst function will locate and return the first physical record located in the designated file, disregarding all key paths.

Syntax

- **String-based API**

```
Record$ = trm_StepFirst(hFile&)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 33 ‘ trm_StepFirst
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 33 ‘ trm_StepFirst
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function.

Results

If successful… trm_StepFirst will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_StepFirst will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
7	File corrupt
9	End of file
50	Data buffer too small
99	Time-out

trm_StepFirst (33) (continued)

Results (continued)

If trm_StepFirst returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_StepFirst function is immediately established as the current physical record for the designated file and logical positioning information for the file is cleared.

Subsequent calls to the trm_Delete, trm_GetNext, trm_GetPrev or trm_Update functions will fail because they require logical positioning. If you're using the trm_Step functions to traverse your records by physical position in the file and wish to switch back to moving through the records via a key path, call the trm_GetPosition function followed by a call to the trm_GetDirect function… that will establish the physical current record as the logical current record in the current key path, also establishing a logical next record (if any) and a logical previous record (if any).

trm_StepLast (34)

The trm_StepLast function will locate and return the last physical record located in the designated file, disregarding all key paths.

Syntax

- **String-based API**

```
Record$ = trm_StepLast(hFile&)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 34 ‘ trm_StepLast
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 34 ‘ trm_StepLast
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function.

Results

If successful… trm_StepLast will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_StepLast will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
7	File corrupt
9	End of file
50	Data buffer too small
99	Time-out

trm_StepLast (34) (continued)

Results (continued)

If trm_StepFirst returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_StepLast function is immediately established as the current physical record for the designated file and logical positioning information for the file is cleared.

Subsequent calls to the trm_Delete, trm_GetNext, trm_GetPrev or trm_Update functions will fail because they require logical positioning. If you're using the trm_Step functions to traverse your records by physical position in the file and wish to switch back to moving through the records via a key path, call the trm_GetPosition function followed by a call to the trm_GetDirect function… that will establish the physical current record as the logical current record in the current key path, also establishing a logical next record (if any) and a logical previous record (if any).

trm_StepNext (24)

The trm_StepNext function will locate and return the physical record immediately following the physical current record in the designated file, disregarding all key paths.

Syntax

- **String-based API**

```
Record$ = trm_StepNext(hFile&)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 24 ‘ trm_StepNext
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 24 ‘ trm_StepNext
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function and a current record (logical or physical) must have been established.

Results

If successful… trm_StepNext will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_StepNext will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
7	File corrupt
9	End of file
22	Lost record position
50	Data buffer too small

trm_StepNext (24) (continued)

Results (continued)

99 Time-out

If trm_StepNext returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_StepNext function is immediately established as the current physical record for the designated file and logical positioning information for the file is cleared.

Subsequent calls to the trm_Delete, trm_GetNext, trm_GetPrev or trm_Update functions will fail because they require logical positioning. If you're using the trm_Step functions to traverse your records by physical position in the file and wish to switch back to moving through the records via a key path, call the trm_GetPosition function followed by a call to the trm_GetDirect function… that will establish the physical current record as the logical current record in the current key path, also establishing a logical next record (if any) and a logical previous record (if any).

trm_StepPrev (35)

The trm_StepPrev function will locate and return the physical record immediately preceeding the physical current record in the designated file, disregarding all key paths.

Syntax

- **String-based API**

```
Record$ = trm_StepPrev(hFile&)
Result& = trm_Result(hFile&)
```

… where hFile& holds a file handle returned by the trm_Open function.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 35 ‘ trm_StepPrev
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 35 ‘ trm_StepPrev
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function and a current record (logical or physical) must have been established.

Results

If successful… trm_StepPrev will return a record to be stored in a string variable or UDT.

If unsuccessful… trm_StepPrev will return a null string and one of the following result codes. In the string-based API, a subsequent call to trm_Result is required to obtain the result code.

2	I/O error
3	File not open
7	File corrupt
9	End of file
22	Lost record position
50	Data buffer too small

trm_StepPrev (35) (continued)

Results (continued)

99 Time-out

If trm_StepPrev returns a null string (because the call was unsuccessful) and a subsequent call to trm_Result returns a result code of 0, then the file handle you passed must have been invalid.

Positioning

Any record retrieved by a successful call to the trm_StepPrev function is immediately established as the current physical record for the designated file and logical positioning information for the file is cleared.

Subsequent calls to the trm_Delete, trm_GetNext, trm_GetPrev or trm_Update functions will fail because they require logical positioning. If you're using the trm_Step functions to traverse your records by physical position in the file and wish to switch back to moving through the records via a key path, call the trm_GetPosition function followed by a call to the trm_GetDirect function… that will establish the physical current record as the logical current record in the current key path, also establishing a logical next record (if any) and a logical previous record (if any).

trm_TimeOut (31)

The trm_TimeOut function allows you to modify Tsunami's internal time-out limit for locked files. When you try to access a locked file, Tsunami will normally attempt your request for 2 seconds (2000 milliseconds) before returning a time-out result code (99) to your application. You can use the trm_TimeOut function at any time within your application, as many times as you like, to alter the internal time-out limit. Whenever Tsunami is started, the time-out limit is set to it's 2 second default.

Syntax

- **String-based API**

```
Result& = trm_TimeOut(Limit&)
```

… where Limit& holds a value from 1000 (1 second) to 60000 (60 seconds). Any value outside of that range will be adjusted to a value within the 1 to 60 second requirement.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 31 ' trm_TimeOut
KeyNo& = Limit&

Result& = trm(Op&, 0, 0, 0, 0, 0, KeyNo&)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 31 ' trm_TimeOut
Tsu.keyno = Limit&

Result& = trm_udt(Tsu)
```

Prerequisites

There are no prerequisites for this function call.

Results

trm_TimeOut returns a long integer value representing the time-out limit that was actually set (in the event your request was outside the acceptable range).

Positioning

This function has no affect on positioning information.

trm_Update (3)

The trm_Update function will insert the supplied record in place of the logical current record in the designated file, updating all related keys.

Syntax

- **String-based API**

```
Result& = trm_Update(hFile&, Record$)
```

… where hFile& holds a file handle returned by the trm_Open function.

… where Record$ represents a string variable holding the record to be inserted in the file in place of the logical current record.

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 3 ' trm_Update
DataPtr& = StrPtr(Record$)
DataLen& = Len(Record$)

Result& = trm(Op&, hFile&, DataPtr&, DataLen&, 0, 0, 0)
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 3 ' trm_Update
Tsu.file = hFile&
Tsu.dataptr = StrPtr(Record$)
Tsu.datalen = Len(Record$)

Result& = trm_udt(Tsu)
```

Prerequisites

The designated file must have been opened by the trm_Open function, a logical current record must be established and the Record$ variable must hold a string long enough to satisfy the key extraction definitions for this file.

Results

If successful… trm_Update will insert the record and it's keys in the file in place of the logical current record, returning a result code of 0.

If unsuccessful… trm_Update will return one of these result codes …

2 I/O error
3 File not open
5 Duplicate key

trm_Update (3) (continued)

Results (continued)

7	File corrupt
8	No current position
20	Invalid record length
22	Lost record position
46	Access to file denied
99	Time-out

Positioning

Any record successfully updated in a Tsunami file by the trm_Update function is immediately established as the current record for that file (logical and physical). The logical next and previous record pointers are adjusted based on any changes made in the updated record that might have modified it's keys.

trm_Version (26)

trm_Version returns Tsunami's major and minor version numbers as the high and low words of a long integer.

Syntax

- **String-based API**

```
Version& = trm_Version
```

- **Pointer-based API #1 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Op& = 26 ' trm_Version
Result& = trm(Op&, 0, 0, 0, 0, 0, KeyNo&)
Version& = KeyNo&
```

- **Pointer-based API #2 (not for use with Visual Basic)** - INFORMATIONAL ONLY

```
Tsu.op = 26 ' trm_Version
Result& = trm_udt(Tsu)
Version& = Tsu.keyno
```

Prerequisites

There are no prerequisites for this function call.

Results

trm_Version returns a long integer holding Tsunami's major and minor version numbers in it's high and low words. Those version numbers can be obtained as follows…

```
Major% = Version& / 256 / 256
Minor% = Version& Mod 256
```

You can also use the following code to avoid the use of a long integer…

```
Major% = trm_Version / 256 / 256
Minor% = trm_Version Mod 256
```

Positioning

This function has no affect on positioning information.

APPENDIX "A"

License Agreement

No one is allowed to use the Tsunami Record Manager (herein referred to as the Software) without reading this entire page. The Software shall be construed to include the following computer files… TRM.DLL and the complete contents of this document, TRM_VB.PDF… whether in printed or electronic form. The Software is owned by Advantage Systems (herein known as the Author) and remains the copyrighted intellectual property of the Author.

The Author hereby grants you a non-exclusive license to use the Software for development of personal or commercial software applications. You are granted the right to distribute the computer file named TRM.DLL (and TRM_VB.DLL) with those applications, but no other computer files or documentation included with the Software may be distributed by you, whether for free or for compensation of any sort.

You are not permitted to distribute the TRM.DLL file as part of a development product, but only as a support DLL (Dynamic Link Library) for software applications developed by you that use the TRM.DLL file. You may not charge any party for use of the TRM.DLL file, but it may be distributed with software applications that you receive compensation for. You may not claim copyright to nor suggest any degree of ownership rights in the Software when included with any applications produced and/or distributed by you.

The Software may not be reverse engineered nor modifed in any way.

Disclaimer

The Software is provided on an "as is" basis and the Author disclaims all warranties of any kind, either express or implied, including but not limited to the implied warranties of merchantability or fitness for a particular purpose. The Author shall not be held liable to you nor to any user of software applications produced or distributed by you, for damages of any kind regardless of the circumstances.

- **BY USING OR DISTRIBUTING THE SOFTWARE, YOU AGREE THAT YOU DO SO AT YOUR OWN RISK** •

The Author offers no warranty or guarantee, nor makes any representations regarding the use of or the results of using the Software as to reliability, accuracy or uninterrupted use. All warranties not stated herein are expressly disclaimed by the Author. Written or oral information provided by the Author or any other party shall not modify the foregoing disclaimer nor create warranties of any kind.

Some jurisdictions do not allow the exclusion of implied warranties or limitations or exclusion of liability for incidental or consequential damages, so the above exclusions and/or limitations may not apply to you.

The Software is protected by United States copyright laws and all applicable international treaty provisions.

APPENDIX “B”

Syntax Reference – String-based API

```
Result&  = trm_Accelerate(hFile&, CacheSize&)

Result&  = trm_Close(hFile&)

Result&  = trm_CloseAll

Total&   = trm_Count(hFile&)

Result&  = trm_Create(File$, FileDef$, OverWrite&)

KeyPos&  = trm_CurrKeyPos(hFile&)

Result&  = trm_Delete(hFile&)

Result&  = trm_FileIsOpen(hFile&)

FSize&   = trm_FileSize(hFile&)

Result&  = trm_Flush

Record$  = trm_GetByKeyPos(hFile&, KeyPos&)
Result&  = trm_Result(hFile&)

Record$  = trm_GetDirect(hFile&, RecPtr$)
Result&  = trm_Result(hFile&)

Record$  = trm_GetEqual(hFile&, Key&, KeyVal$)
Result&  = trm_Result(hFile&)

Record$  = trm_GetEqualOrGreater(hFile&, Key&, KeyVal$)
Result&  = trm_Result(hFile&)

Record$  = trm_GetEqualOrLess(hFile&, Key&, KeyVal$)
Result&  = trm_Result(hFile&)

FileDef$ = trm_GetFileDef(hFile&)
Result&  = trm_Result(hFile&)

FileVer$ = trm_GetFileVer(hFile&)
Result&  = trm_Result(hFile&)

Record$  = trm_GetFirst(hFile&, Key&)
Result&  = trm_Result(hFile&)

Record$  = trm_GetGreater(hFile&, Key&, KeyVal$)
Result&  = trm_Result(hFile&)

Record$  = trm_GetLast(hFile&, Key&)
Result&  = trm_Result(hFile&)

Record$  = trm_GetLess(hFile&, Key&, KeyVal$)
Result&  = trm_Result(hFile&)
```

Syntax Reference – String-based API (continued)

```
Record$  = trm_GetNext(hFile&)
Result&  = trm_Result(hFile&)

RecPtr$  = trm_GetPosition(hFile&, 0)
Result&  = trm_Result(hFile&)

Record$  = trm_GetPrev(hFile&)
Result&  = trm_Result(hFile&)

Result&  = trm_Insert(hFile&, Record$)

Result&  = trm_Integrity(hFile&)

hFile&   = trm_Open(File$, MultiUser&)

Result&  = trm_Rebuild(File$, SaveFile$, FileDef$)

Result&  = trm_Recover(SourceFile$, TargetFile$, Page&, Comp&)

Result&  = trm_Result(hFile&)

Result&  = trm_SetKeyPath(hFile&, Key&)

Record$  = trm_StepFirst(hFile&)
Result&  = trm_Result(hFile&)

Record$  = trm_StepLast(hFile&)
Result&  = trm_Result(hFile&)

Record$  = trm_StepNext(hFile&)
Result&  = trm_Result(hFile&)

Record$  = trm_StepPrev(hFile&)
Result&  = trm_Result(hFile&)

Result&  = trm_TimeOut(Limit&)

Result&  = trm_Update(hFile&, Record$)

Version& = trm_Version
```

NOTE

The following only applies to Tsunami's string-based API…

All of the trm_Get and trm_Step functions listed above return string values, but they can also provide a numeric result code if you immediately call trm_Result, p assing the same file handle as a parameter. You aren't required to call trm_Result, but you may want to call it whenever you receive a null string from one of the trm_Get or trm_Step functions.

APPENDIX "C"

Result Codes

0 Successful

Operation was successful.

1 Not a Tsunami file

A trm_Open operation did not recognize the file as one created by the trm_Create function. If the file actually was created by the trm_Create function, then the file's header page must be damaged and the file needs to be rebuilt or recovered before you can access it again.

This result code is also returned if you attempt to access a Tsunami file using a version of TRM.DLL that's older than the version used to create the file.

2 I/O error

An error occurred during a disk read or write operation. Normally, this would indicate some degree of media or hardware failure and could require a rebuild or recover of the affected file.

3 File not open

The operation failed because the designated file was not open. A successful call to trm_Open must be executed, which returns a valid "file handle", before any operations may be performed on the records in the file. This result code could also be returned if the file handle passed was 0 or wasn't a valid handle.

4 Key not found

The supplied key value was not found in the designated key path.

5 Duplicate key

During an attempt to insert or update a record, a duplicate key was found in a file where duplicates are not allowed in one of it's key paths.

6 Invalid key

The key number parameter passed to a Tsunami function was not valid for the designated file. The key number must be a number from 1 to the highest key number defined for the designated file when it was created.

7 File corrupt

Operation failed due to physical corruption in the file. Possible explanations include media or hardware failure, file scrambling due to fluctuating or impure power levels during write operations, etc.

8 No current position

An operation was attempted which requires that a current record position (logical or physical) be established within the file before it can succeed.

9 End of file

An operation attempted to read beyond the logical boundries of a key path in a file or beyond the physical boundries of the file (end-of-file or start-of-file).

Result Codes (continued)

10 Invalid page size

The first byte of a FileDef$ string used in a call to trm_Create did not contain an ASCII character in the range of 1 to 8, indicating the desired page size for the new file (expressed in KBytes… 1=1K, 2=2K, etc).

11 Invalid number of key segments

The third byte of a FileDef$ string used in a call to trm_Create did not contain an ASCII character in the range of 1 to 128, indicating the number of key segment description blocks contained in the FileDef$ string. This result code is also returned if the number of key segments indicated exceeds the maximum allowed for the defined page size (see page 2).

12 Invalid file definition string

The FileDef$ string used in a call to trm_Create was not properly constructed. Check for the following errors…

- The FileDef$ string was a null string
- The second byte of the FileDef$ string didn't contain an ASCII character 0, 1, 2 or 3 (used for defining record compression and internal record ID retention preferences)
- The FileDef$ string was not the expected length
- There was an error in one or more key segment definition blocks…
  - An assigned key number was not in the range of 1 to 128
  - An assigned key number was out of natural sequence
  - A gap exists between two assigned key numbers

13 Invalid key segment position

The FileDef$ string used in a call to trm_Create includes a key segment definition block that defines the segment's starting position as 0.

14 Invalid key segment length

The FileDef$ string used in a call to trm_Create includes a key segment definition block defining a segment with a length of 0, or the lengths of all segments making up a single key exceed the maximum allowed (255 bytes).

This result code will also be returned if a key segment is defined as a binary key segment (by applying the BINARY_KEY key flag constant) and the key segment's length is defined as anything other than 2 bytes (INTEGER) or 4 bytes (LONG INTEGER).

15 Inconsistent key segment definitions

The FileDef$ string used in a call to trm_Create includes key segment definition blocks having the same key number, but with conflicting "key flags" regarding duplicates or key compression.

This result code will also be returned if you define a multi-segmented key containing a binary key segment while neglecting to apply the NO_COMPRESSION key flag to one or more non-binary segments of the key.

Result Codes (continued)

20 Invalid record length

The record that was passed to the trm_Insert function or to the trm_Update function was not long enough to accommodate the key extraction definitions for the designated file. Any key segment's starting position plus it's length must represent a valid position within the record.

21 Invalid record address

To successfully execute a trm_GetDirect function call, the record pointer passed must be a 12-byte string that was returned by a successful call to the trm_GetPosition function.

22 Lost record position

Lost record positions are reported when a Tsunami function fails because a record that must be accessed by the function has been either updated or deleted since it was acquired by your application. Use trm_GetPosition followed by trm_GetDirect to reload it (if it wasn't deleted).

30 Access denied

An attempt to open a file failed because it's already open in single-user (exclusive) mode.

31 File already exists

This result code is returned by trm_Create when the OverWrite& parameter is set to a value of 0 and the file you want to create already exists.

32 No more file handles

An attempt to open a file failed due to a shortage of file handles being made available to your application by Windows.

33 Max files open

An attempt to open a file has failed because TRM.DLL has reached it's internal limit on open files (255).

40 Accelerated access denied

The Tsunami file that you tried to accelerate is not open in single-user mode or there are already 16 files being accessed in accelerated mode.

41 Acceleration cache error

An error occurred while trying to allocate memory for the acceleration cache.

46 Access to file denied

An attempt to insert, update or delete a record failed because the file was opened in multi-user, read-only mode.

50 Data buffer too small

When using either of Tsunami's two pointer-based APIs, many functions require you to pass a data buffer length as a parameter. If you're attempting to retrieve a record that's larger than the data buffer length that you specified, you'll receive this result code. When this happens, a record will not be returned. However, the data buffer pointer and data buffer length parameters will then return the memory location and length of the complete record so you can

Result Codes (continued)

retrieve it directly without making another call to Tsunami.

99 Time-out

The Tsunami file that you tried to open is currently in use in single-user, exclusive mode… try again. If this result code persists, another thread, application or user who accessed the file in either single-user or multi-user mode may have locked up, leaving the file inaccessible to others until a system reset frees it.

Special Note regarding result codes returned by the following five functions…

```
trm_Count
trm_CurrKeyPos
trm_FileSize
trm_Integrity
trm_Open
```

In the string-based API, these five functions return the following positive numeric values in the Result variable when they are successful ...

- trm_Count returns 0 or greater, reflecting the number of records in the designated file.
- trm_CurrKeyPos returns 1 or greater, reflecting the approximate position of the logical current record in the current key path of the designated file.
- trm_FileSize returns 3 or greater, reflecting the size of the designated file in KBytes.
- trm_Integrity returns an incremental progress counter (1 to 100) as it moves through the file being checked, and ultimately returns a value of 0 once the file has been successfully checked and no errors were encountered.
- trm_Open returns 1 to 255 as a file handle to be used from that point on to access the file that was just opened.

These five functions will also return a result code when unsuccessful. Result codes are normally positive numbers, but in the case of these five functions the result codes are returned as negative numbers so as not to confuse the result code with a return value.

APPENDIX "D"

TSUNAMI Tips

• DEFINING KEYS

When designing your data files, you should only define keys (indexes) that are really necessary for routine daily access. Developers often define far more keys than they need because they believe a key should be defined for every possible report or access scenario that will arise. Defining keys for use in reports that are run on a frequent basis will definitely save report generation time, but for those reports that are generated infrequently, using a general access key with a record selection routine will normally be more than adequate.

Overloading any data file with seldom used keys doesn't harm anything, but it will most certainly slow down all record insertions, updates and deletions.

• PAGE SIZE SELECTION

When creating a Tsunami file, always try to use a small page size (1K or 2K) if it will comfortably hold the average sized record to be stored in the file. If you have a sampling of the records that will be stored in the file, it may be beneficial to test a few different page sizes to see which results in the best combination of efficient storage (file size) and performance (access speeds). The impact of page size on access speeds is most evident when reading and writing records across a network connection.

• RECORD STRUCTURE / COMPRESSION RESULTS

If you choose to implement Tsunami's record compression feature for a file, you can improve the compression ratio achieved by Tsunami by structuring your records so all of the fields that are most likely to be filled with blank characters are situated contiguously in the record. Any contiguous run of 4 or more repeating characters (up to 255) is converted to a 3-byte code that represents a "compression block". For example, a contiguous run of 500 blanks would be represented by 2 compression blocks (or 6 bytes).

Such record structuring is definitely not required for Tsunami to perform well, but it does produce smaller compressed records for those situations where file size is of major concern to an application.

• READING TSUNAMI FILES

When writing an application that uses the Tsunami Record Manager, you will often need to traverse a complete record set (file) from beginning to end or from end to beginning. Tsunami gives you two ways to do this… logically or physically.

Moving **logically** along a key path from the beginning of a Tsunami file to it's end requires the use of the trm_GetFirst function followed by repeated calls to trm_GetNext until result code 9 (End of file) is received. To move logically from the end of a Tsunami file to it's beginning, use the trm_GetLast function followed by repeated calls to trm_GetPrev.

TSUNAMI Tips (continued)

Moving **physically** from the beginning of a Tsunami file to it's end is accomplished by calling the trm_StepFirst function followed by repeated calls to trm_StepNext. To move physically from the end of a Tsunami file to it's beginning, use the trm_StepLast function followed by repeated calls to trm_StepPrev.

There are advantages to both methods. Using the trm_Step functions to move through a file physically is often measurably faster than moving logically by key path with the trm_Get functions. However, the records will not be retrieved in any particular order when using the trm_Step functions and logical positioning is not established. Therefore, you can't use the trm_Step functions to move through a file for the purpose of deleting or updating records as you go… the trm_Delete and trm_Update functions require a **logical** current record position.

However, traversing the file with the trm_Step functions could be used for such a purpose if all the deleting and/or updating operations take place *after* reading the entire file. Simply use the trm_GetPosition function to receive a 12-byte pointer for each record you select for deletion or updating and store those record pointers in a string array until you have "stepped through" the entire file. You can then use the trm_GetDirect function inside a For/Next or a Do/Loop statement to load and process each of the selected records.

• RIGHT JUSTIFIED TEXT KEY VALUES

All keys in a Tsunami file are extracted from the records you pass to TRM.DLL in trm_Insert and trm_Update function calls, and are therefore always strings. Comparisons made on strings are performed from left to right and are quite simple when dealing with left justified key values, such as a customer name or an inventory item number. However, when key values contain right justified text (such as a dollar amount in an "account balance" field) you may need to perform extra steps to ensure proper sorting.

When you convert numeric data to string format (using STR$) you should always be sure to right justify the resulting text within it's field length in the record, since left justified strings of numbers will most certainly lead to improper sorting. In most cases, right justifying strings of numbers is all it takes to provide for proper sorting, but that alone may not be enough to guarantee accuracy. You may also run into situations where a string of numbers could represent a negative value and will need special handling…

For example, take these two key values (8 characters long, right justified text)…

```
" -100.00"
"  100.00"
```

It's obvious to even the most casual observer that the key representing the negative number would be lower and should come before the other in sorted order. However, since these are strings that are compared byte-by-byte from left to right, it's a different story. Starting at the left, the first byte of these two strings match, but the second bytes differ and a blank space is considered lower than a negative sign (by ASCII code). For this reason, the positive number would be seen as lower than the negative number and would produce improperly sorted key values.

TSUNAMI Tips (continued)

To resolve this issue, simply include a function in your source code that converts "leading" blank spaces in a string to zeros. Something like this example…

```
Function ZeroPad(A As String) As String

   Temp& = 1
   Do While Mid(A, Temp&, 1) = Chr(32)
      Mid(A, Temp&) = "0"
      Temp& = Temp& + 1
   Loop
   ZeroPad = A

End Function
```

Using this function on the two string examples shown above would return the following strings to be placed in their proper positions within their respective records…

```
"0-100.00"
"00100.00"
```

This way, when Tsunami extracts these key segments, they will sort properly since the ASCII code for a negative sign is lower than the ASCII code for a zero.

In summary, any non-binary key segment in a record that represents a numeric value should have it's text right justified within it's field length, and the NO_COMPRESSION key flag should be used for that key to ensure proper sorting (see page 18). Further, any non-binary key segment that represents a numeric value that might contain a negative value should be left padded with zeros to ensure proper sorting.

Note: You will also need a simple function in your source code similar to the one shown above that would convert the "leading" zeros in a string back to blank spaces for use on such a field after retrieving a record for display or printing.

• MULTI-THREADED FILE ACCESS

Making Tsunami function calls within threads is certainly allowed, but make sure that you do it safely and wisely. Don't automatically assume that you will gain a speed advantage by trying to read a group of records from a file within numerous threads. It might actually take longer to read the records that way than if you simply access the file from within the main body of your application.

Using threads would certainly make sense in a case where you are reading a file to produce a report and want to allow your users to do other work while the report is generated within the thread, or if you want to allow users to produce multiple reports "at the same time" by using multiple threads. In these situations, threads will enhance your users' productivity and could be a wise programming choice.

If you are accessing a Tsunami file from within one or more threads when that file is already open in the main body of your application, be sure to open the file again (in multi-user mode) independently within each thread to get a separate file handle for use within each thread. Close the file within each thread using the file handle received within that thread.

TSUNAMI Tips (continued)

• BATCH-LOADING RECORDS

When "batch-loading" a large number of records into an empty Tsunami file, you can produce a much smaller file if the records being inserted are supplied to Tsunami in pre-sorted order (ascending or descending). This allows Tsunami to create more densely populated key pages, not only making the file smaller, but also reducing the time needed to load the records.

• USER DEFINED TYPES

When a Tsunami function returns a record from a data file (trm_Get and trm_Step functions) you must receive the record in a string variable. The same applies to records that are passed to Tsunami as function parameters… you must send a string variable. This is because Tsunami treats all records as being potentially variable length.

You can, however, use a structured variable (UDT) for records, the contents of which are moved into a string variable (a buffer, if you will) before passing it to Tsunami, and copied from the string variable (buffer) back into the UDT when Tsunami returns it. To accomplish this, follow these steps…

```
' define your UDT
' this one is a total of 200 bytes in size

Public Type CustomerType
   AcctNo As Long
   LastName As String * 30
   FirstName As String * 30
   Street As String * 30
   City As String * 26
   St As String * 2
   Zip As String * 10
   Misc As String * 60
   AcctBal As Currency
End Type
```

Declare your UDT variable and the string variable you'll be using as the "buffer" …

```
Dim Customer As CustomerType

Dim Record As String * 200
```

After assigning values to the UDT elements, insert a record into your Tsunami file as follows …

```
tmpFile& = FreeFile
Open "c:\tempfile" For Random As tmpFile& Len = Len(Customer)
  Put tmpFile&, 1, Customer
  Get tmpFile&, 1, Record
Close tmpFile&

Result& = trm_Insert(hCustomer&, Record)

' check Result&
```

When retrieving a record from your Tsunami file, you can move the contents of the string variable into the elements of the UDT variable as follows …

TSUNAMI Tips (continued)

```
Record = trm_GetFirst(hCustomer&, 1)
Result& = trm_Result(hCustomer&)

If Result& = 0 Then
   tmpFile& = FreeFile
   Open "c:\tempfile" For Random As tmpFile& Len = Len(Customer)
     Put tmpFile&, 1, Record
     Get tmpFile&, 1, Customer
   Close tmpFile&

Else
   Msg$ = "Error on trm_GetFirst:" + Str(Result&)
   ' display message

End If
```

The above routines impose very little in the way of a performance penalty, while transparantly handling the conversion of Visual Basic's "unpacked" (padded) UDTs to "packed" UDTs. The conversion of Visual Basic's Unicode strings to ANSI strings (as required by Tsunami) is also handled in the above routines. Visual Basic takes care of those issues during it's Put and Get file I/O operations.

You may want to place these routines in functions or subs to avoid repetitive code. It's also advisable to open your "tempfile" just once at the beginning of your application and close it only when your application exits to avoid any unnecessary overhead required to open and close the "tempfile" each time you use these routines… it all depends on your application.

• ACCELERATED ACCESS

Using trm_Accelerate can greatly improve file I/O performance, but there are a few things you should consider when deciding how to use it. Caching disk I/O provides the most benefit in operations that modify the file (insert, update and delete operations). You will definitely see speed improvements using trm_Accelerate in read-only situations, but not as dramatic an improvement as when you're modifying a file. Tsunami limits the number of files that may be accelerated at any one time to 16, so whenever you'll be accessing more than 16 files at once, you might want to be selective about which ones you accelerate. For best results, you should always accelerate the files that will be modified before those that will be accessed for read-only.

Tsunami's acceleration cache is an "aggressive" cache, meaning it performs write caching as well as read caching. Once the requested cache has been filled, aggressive caching ends… no additional page writes are cached. At that point, a "passive" cache comes into play to supplement the aggressive cache. This passive cache takes the form of a bounded "ring cache", providing fast access to the file's 100 most recently accessed key pages. It should be noted that such a ring cache delivers exceptional performance when records are being inserted in pre-sorted order (near 100% hit rates), while random record inserts can see hit rates drop to between 40% and 60%.

For the above stated reason, requesting a large cache size for files that are (or will become) quite large can be beneficial, but aren't really needed if a large number of records are being inserted in pre-sorted order. Also, the larger the cache size, the longer it takes to "flush" the contents of the cache to disk when acceleration is turned off… something your application(s) may need to take into consideration.

Contact Information

The Tsunami Record Manager was written by Advantage Systems for it's own use as a development tool for commercial applications. It has been released at no charge to all interested Win32 developers, subject to the conditions outlined in the License Agreement and Disclaimer in Appendix "A" of this document.

Technical support for Tsunami is only offered to members of the Tsunami Users Group. Membership in the Users Group is not required to use the free version of Tsunami, but is required if you want access to support services, the Users Forum discussion boards, free utilities for Tsunami and any future Pro Editions of Tsunami. Please visit our web site often for complete details...

www.TRM-UG.com