ADataBase

What it is:

AdataBase is a class that allows a C++ programmer to create, maintain and use a single keyed indexed flat file database. The database will handle unique or duplicate keys of a length of the programmer's choice. Records are limited to 64530 total bytes (key plus data) by the provided sort software. Unsigned sixty-four bit arithmetic is used where appropriate so that there is a theoretical limit of several billion billion bytes of data. As a more practical consideration, a database is limited by the available resources required to sort the main file. Initially the number of records can be set to any desired number. The files can grow automatically as needed.

Records may be retrieved, added, deleted and changed easily. Record retrieval is exceptionally quick. Full record locking is employed so that any number of users can access the database simultaneously. File locking ensures exclusive file use for sorting the index file which must be done periodically to maintain performance and reorganizing the database to reclaim space from records marked for deletion. Automatic expansion, sort and reorganization, can be enabled.

Field extraction and insertion routines are provided. Being a class, as many of these databases may be defined in a program as practical.

What you get:

The zip file, adatabasedemo.zip, contains:

1) A simple C++ project called databasedemo. It was created with Visual C++ V6 as a simple C program without MFC and a window. Unzip the file to where ever you please. Assuming you have Visual C++ or something like it, you may build and run the simple demo program which, if all goes well, will wind up displaying my address in a message box.

2) Two files: adatabase.h which must be included in your projects with the #include directive and adatabase.lib which must be linked to

your executables. Move them to locations where your compiler and linker can find them.

3) A file called “sortsol.dll” which is used in the sort and reorganization process. It must be located in %path%. Moving it to “c:\WINDOWS\System32” is a good option. It can also reside in the folder from which the executable is run. The sort software runtime is from Mario M. Westphal of Germany. It is a tournament sort and will handle multi-gigabyte files where there appears to be inadequate resources to do it. It is the best sort software for Windows of which I am aware.

4) A MS Word and PDF version of this manual.

Public functions:

To save space in the public function description list below, the following conventions have been employed:

For functions that have no return, all errors are fatal and unrecoverable. They produce a detailed error message and exit the calling program.

Arguments enclosed in ‘[]‘s are optional.

For functions that return a Boolean, if the record cannot be read and/or locked after ten seconds, a warning message box will pop up giving the user the option of continuing for another ten seconds or abandoning the effort. Someone else might have the record locked for modification. Users should be encouraged to complete the modification of existing records as quickly as possible to avoid the situation where a record is locked for an unreasonable amount of time. If and when the user abandons further attempts, the function will return false.

In many cases, a false return might just as well indicate that the record (key) is not found. While the return is the same and should be treated by the programmer as the same, the user will most certainly know the difference. All other errors are fatal and are treated as above.

The following public functions are available in the database class, ADataBase. The argument(s) represented as "string" can be standard C strings or C++ CString constructs. Those presented as "char*" must be pre-allocated buffers of the size of the maximum data length plus the size of the key plus one byte. They are to be filled by the called function.

Initial operations:

Void SetName(string); this must be the first call into the database. It is passed a path, file name and extension of the main database file and from which the names of the database lock file and the index file are derived.

Boolean Opentest(); this call can be used after SetName(...) to determine if the database has already been created. If not, the programmer should collect the appropriate parameters and call -

Void Create(UInt32, UInt32, UInt64, UInt32); the information passed is: the key size in bytes, the maximum data size in bytes, the initial number of records and the flag. There are four flag bits that can be summed together:

DUPLICATEALLOWED	Duplicate keys are allowed
AUTOGROW	Automatically increase number of records as needed
AUTOSORT	Automatically attempt to sort when needed
AUTOREORG	Automatically attempt to reorganize when needed

Boolean Open(); this opens the files in read/write-shared mode. If a failure occurs at this point because of file locking, the programmer will most likely choose to abandon the effort for the time being.

Boolean SetParams(UInt32, UInt32, UInt32, UInt32) (optional)
The arguments are:

FLAGS
AUTOGROW trigger amount

AUTOSORT trigger amount
AUTOREORG trigger amount
A value of zero for any of the arguments means “no change”.

Although the flag settings and default triggers for the automatic operations are usually sufficient for most needs, SetParams() will allow changes to be made at run time. The DUPLICATEALLOWED setting may not be changed. The three AUTO flag settings may be changed and the change is permanent until another change is made. The automatic trigger levels are valid only for the instance of the calling program. While the database need not be open to change trigger levels, it must be to change flags.

Void SetFields(…) passes an ordered list of field lengths that constitute the fields in the record. Include the key and terminate the variable argument list with a NULL. Failure to include the terminating NULL argument can result in crashes which are difficult to diagnose.

Record addition:

Boolean Set(string, string); passes the key and the data, both of which may be shorter than defined during creation but must be longer or equal to one byte. Keys are alphanumeric. If keys are constructed from numbers, please format them consistently for sorting. **Keys may not start with the space character.** Neither the key nor the data string may contain the fill character, '}', right brace.

Keyed retrieval:

Boolean Get(string[, char* data]); use this call to get the first (or only) record with the passed key. It returns the data portion of the record in data and true from the function. If you are using the field retrieval function, you may exclude the data buffer in the call.

Boolean GetNext([char* data]); in a database which allows multiple keys, this will get the next data record in the file. Like Get(...), a

false return could be because the key was not found or the record was locked and abandoned.

Sequential retrieval:

UInt32 GetRec(UInt64[, char*]); Gets the record specified. Returns 1 if call return valid data, 0 if requested record is marked for deletion and -1 if the call went beyond the valid data area. Data returned includes the key. This is used to sequentially access the database. The programmer is responsible for incrementing the record number. Start at 1.

Field access:

Char* GetField(UInt32) Having set the fields, a call to GetField with the field number (starting with 1) will return a pointer to a NULL terminated string buffer containing the desired field content. It is the programmer's responsibility to assure that a valid Get(), GetNext() or GetRec() without intervening calls has been performed. No internal checks are made.

Void PutField(UInt32, string, char*) adjusts the string to the appropriate length padding with blanks as necessary and stuffs it into the user provided character buffer at the appropriate location. The data buffer will be treated as a buffer appropriate for Set() or ChangeRec(), i.e. the key field will be omitted. Use zero based field numbers, just don't use zero. Forgetting fields in strings to be passed to the Set(…) function is not a good idea.

Modification:

Boolean LockRec(char*); locks the most recently obtained record from Get(...) or GetNext(...). This is required before the programmer uses ChangeRec(...). This will re-obtain the record after locking it in preparation for ChangeRec(…). The programmer should redisplay the data so that the user can be sure it is the record to be changed and that no one else has changed it in the mean time. In a database that allows duplicate keys, the programmer must ensure that the correct record is being manipulated. To do this, perform the following sequence:

Determine the correct record to delete/modify by using Get(…) followed by GetNext(…). Once you have found the correct record, call DeleteRec() or LockRec(…) followed by ChangeRec(…).

Void ChangeRec(string); writes the string over the record you have most recently retrieved and locked. This will unlock the record. Failure to lock the record prior to this call will cause a fatal error.

Boolean DeleteRec(); marks the most recently retrieved record for deletion.

Other:

void Close(); important to call this prior to Sort(), Reorg(), and on exiting your program.

Boolean Sort(); should be done on a regular basis when the database is not being used. It attempts to open the files for exclusive use and warns the user on failure. The programmer must Close() the database prior to this call to insure his own process does not prevent the exclusive file lock.

Boolean Reorg(); should be done on a regular basis when the database is not being used. It attempts to open the files for exclusive use and warns the user on failure. The programmer must Close() the database prior to this call to insure his own process does not prevent the exclusive file lock. Note; reorganizing the file involves checking each record for being marked for deletion, sorting the main file and rebuilding the index from the main file. On multi-gigabyte files, this can take over a half an hour. If your file is large, plan it at an off time. . In order to simulate sorting on multiple fields, the sort key for reorganization is the entire record so the programmer should arrange fields to be alphanumeric and in the order desired.

void DisplayStatus(string); Allows the user to see how many records are sorted relative to the actual number present, how many records are present relative to the maximum and how many

deleted records exist. It displays information contained in the control record of the main file. The string argument is "your message" to be included.

Note to programmers! It has been the author's observation that there are bugs in C compilers that present problems in handling 64 bit integers. In particular, assigning a variable defined as 64 bit (UInt64) a value can produce unpredictable results. These problems can be avoided by type casting. For example:

```
UInt64      var;
int         x;

var = (UInt64)0;
var = (UInt64)x;
```

Technical support:

Rick Marsh
Datanex, Inc.
832 Wecoma Lp.
Florence, OR 97439
541 902 9595
Rick.marsh@datanex.com

Please call during normal business hours, Pacific Time.