Documentation for DiskIO.h and DiskIO.c

Steven Andrews, © 2003
See the document "LibDoc" for general information about this and other libraries.

```
int GetName(char **name,int *nmfre);
int SaveData(float *a,int m,int n,char *name,int append);
int SaveString(char *s,char *name,int append);
int LoadData(float *a,int n,char *name);
int LoadData2(float **a,int *n,int *col,char *name,int skip);
int LoadData3(float **a,int *n,int xcol,int ycol,char *name,int skip);
int KillData(char *name);
```

Requires: `<stdio.h>,<stdlib.h>,<string.h>,"DiskIO.h","string2.h"`
Example program: `LibTest.c, SpectFit.c`

History: First parts written 3/30/97; modified 9-12/98, 2/99,5/99,8/99. Moderate testing. Works with Metrowerks C and on Linux. Added better error detection and reporting 10/31/01. Fixed and substantially rewrote loading routines 1/24/02. Trivial changes to saving routines 3/3/02. Modified LoadData3 2/13/06 so that it now recognizes commas as separators.

These routines implement simple disk input and output operations for various standard types of data. Routines return one of several error codes, shown below. Note that this is a change from a previous version in which a 0 meant error and 1 meant success.

I/O error codes

0	no error
1	memory allocation failed
2	transfer cancelled by user
3	write error
4	read error
5	file not found
6	illegal inputs to function

All routines start by calling `GetName`, so a consistent set of file name conventions is used. An empty name means that `GetName` should get the name from the user; "cancel" means that the operation should be canceled, and "stdin" and "stdout" imply reading or writing to the standard input or output. In practice, "stdin" doesn't work well for several routines because it isn't possible to type an end of file character. However, it may work for input redirection.

```
int GetName(char **name,int *nmfre);
```

`GetName` may be called from externally, but it's principal purpose is as a subroutine of the other routines here. If `*name` is `NULL` (`name` should not be `NULL`, but `*name` may be), then it allocates a `*name` string with 255 characters. If `*name` is blank, including a `NULL` input, then the routine asks the user for a file name, using the standard i/o commands. The user may enter a name, or "cancel" to cancel the file operation. The routine returns the file name, a 1 in `nmfre` if the name is to be freed at the end or 0 if not, and one of the return codes 0, 1, 2, or 6. The name never needs freeing if an error occurs.

`int SaveData(float *a,int m,int n,char *name,int append);`

`SaveData` saves a matrix of floats. Pass in the address of the first array element, the number of rows, the number of columns, and the file name. If `append` is 0, the routine checks for a previous file and, if it finds one, asks the user whether it should be overwritten; 1 means to append any previous file, creating a new file if needed; and 2 means to overwrite any previous file without checking with the user. The file name is first checked by `GetName`, getting it from the user if neccessary. If you want the name returned, send in a blank string. Data are written to disk as a text file with no header and with columns separated by a single space. Possible return codes are 0, 2, 3, or those of `GetName`.

`int SaveString(char *s,char *name,int append);`

`SaveString` saves a single string to disk, but is otherwise similar to `SaveData`.

`int LoadData(float *a,int n,char *name);`

`LoadData` loads a vector of known size, and has essentially the same arguments as `SaveData`. The array needs to be allocated before the routine is called. It does not check for end of file problems. `LoadData` gets and returns the file name in the same way as `SaveData`. Possible return codes are 0, 4, 5, or those of `GetName`.

`int LoadData2(float **a,int *n,int *col,char *name,int skip);`

`LoadData2` is similar to `LoadData` but is generally more useful as it doesn't require knowledge of the file size before the file is read. However, the file does need to have at least as many columns in every row as in the first row. `a` is sent in as an unallocated array of floats and is returned with the data, `n` is returned with the total number of rows, `col` is returned with the number of columns in the file, `name` is used in the same way as in `SaveData`, and `skip` is the number of lines in the file that should be ignored before starting to read data. `LoadData2` works by loading data into a linked list of rows of data. When the end of file is encountered, the data are copied into `a` and the linked list is freed. If there is an error, `a` is returned as `NULL`, but the number of columns is accurate if it was determined before the error and the number of rows indicates where the error occured.

`int LoadData3(float **a,int *n,int xcol,int ycol,char *name,int skip);`

`LoadData3` is very similar to `LoadData2`, with the exceptions that it only loads in two columns and that it allows data files with different numbers of elements in different rows. It also allows "cavities" in a data file, where a column has a blank section. `a` is sent in as an unallocated array of floats, and returned as a `n`x2 matrix, with a column of *x*'s and a column of *y*'s, and `n` is also returned. `xcol` and `ycol` are the column numbers of the *x* and *y* data, starting with column 1. Column 1 starts at the left edge of the file, and subsequent columns are separated by a single space, comma, or tab.

`int KillData(char *name);`

`KillData` deletes the disk file `name`, with the same file name method as elsewhere. The routine doesn't check with the user for permission. Possible return codes are 0, 5, or those of `GetName`.