

IDF

Datafile Writer/Reader

1. Comments & Package Contents
2. Datafile format
3. Using the utilities (IDF Writer & IDF Reader)
4. Reading from the datafile
5. License

NOTE: To use the library portion of this collection you will need my slightly modified SDL bindings. It wraps an additional function not found in the previous euSDL release by Mark Akita. I have included it in this package for simplicity. Just replace your previous file with this one. (Assuming you have already downloaded euSDL.)

1.

This collection contains a few tools for creating datafiles useful in SDL programs. It is not by any means a completed project but works fairly well in the capacity for which I designed it. Everything is a bit rough around the edges but I will polish it up as time goes on. This documentation is very basic (and crappy) but should give you enough information to use the collection. Any suggestions or feedback are welcome.

Email: ssallen@gmail.com

Enclosed in the zip file you should have the following files:

IDF Writer (.exw utility)

IDF Reader (.exw utility)

idf.ew (code library for accessing datafile contents)

idf_extract.exw (demo showing how to extract datafile objects from the datafile)

idf.pdf (this manual)

Sdl_wrap.ew (a slightly modified version of Mark Akita's SDL bindings for EU)

2.

The datafile format is very is easy to read. I have attempted to keep the format as simple as possible. Future versions of this collection will likely include more versatile formats. The standard file format is illustrated below.

---Start of File---

Key number [4 bytes]

Number of items in datafile [4 bytes]

Key Garbage [(Key Number * 4) bytes]

Size of First Object [4 bytes]

First Object [X bytes]

Key Garbage [(Key Number * 4) bytes]

Size of SECOND object [4 bytes]

Second Object [X bytes]

---End of File---

Description and Explanation:

The IDF datafile uses a key number as a method of data protection. Admittedly, its not a very good method, but it should keep most people honest. When you use the IDF Writer you can instruct the program which key you would like (1-99). This key indicates how many bytes of **garbage** data that will precede any datafile object. Hence you can safely code your program to skip these portions of the file. For an example see the included demo programs, or even the writer/reader utilities.

The first block of 4 bytes contains the key number chosen during the datafile's creation. This can easily be removed by changing a few lines of code in the writer/reader utilities and library. Just remember your key or you will not have access to your data!

The second block of 4 bytes holds the number of objects in the datafile.

The third block of bytes holds the key garbage mentioned in the beginning of the description.

The fourth block holds the size (in bytes) of the first object. Thus, if you have a bitmap object that is 19kbs then this number will contain 19kbs. This makes it easy to read the data or to skip it when looking for a particular object.

The last block (assuming a 1 object datafile) contains the actual binary data of the datafile object. If you had more than one object than it would be followed by more garbage data, the next object's size, and then the next object's data.

Visual Example:

```
--StartofFile--
[4 bytes]  7      (our key number)
[4 bytes]  3      (how many objects in the file)
[28 bytes]          (4 bytes * 7 (keynumber) = 28 filled with garbage)
[4 bytes]  4854   (size of first image)
[4854]          (object data)
[28 bytes]          (more key garbage assuming you have more than one
                    object in the datafile)
[next object's size]  “
[next object's data]  “
```

I hope that makes sense so far! Okay, on to the utilities.

3.

You have two utilities included in this collection. The first is the IDF writer. This is by far the most important so we will cover it more thoroughly. First I would like to note that this collection was created as a tool for Mark Akita's SDL bindings. As such the utilities were made so I could encapsulate my game images into datafiles and access them easily. Kinda like Allegro's Grabber but for euSDL. As such it is primarily intended for bitmaps and later .wav/mp3s/etc. I have really only tested it with bitmaps and a few short text files. It should be fine with other formats but you will need to code the reader yourself. Moving on...

The writer program is very easy to use. Just click on the Add Bitmap >> button and select all the files you want to include. When you have completed adding the bitmaps simply type in a datafile name in the Text Edit field. Then place a key in the key field and click on "Create data file". The datafile will be created in the directory your images were selected from. If you wish to view the bitmaps prior to creating the datafile you may click on the image name from the list. Another window will pop up displaying the datafile.

If you look in the directory that contains your newly written datafile you will also find an IDF_datafile_index.txt file. This file shows you which objects (by previous filename) are in the datafile and their correct order. This is just handy as an index.

Note that the IDF writer utility will always write the COMPLETE file to the datafile. It will not remove any header information from the original file.

The Reader utility is just a simple program to display information about an IDF datafile. If you load a datafile through it then it will display the key of the datafile as well as other information about it. By selecting an object from the list on the right you will be given the size and starting byte of that object. This utility is just a way of verifying basic integrity of the datafile. Eventually I want it to display objects from a datafile.

4.

To read from the datafile your code should follow this premise:

1. Open the datafile.
 `fnum = open("mydatafile.dat", "rb")`
2. Read the first 4 bytes.
 `datachunk = get_bytes(fnum, 4)`
 `datakey = bytes_to_int(datachunk)`
3. Read the next 4 bytes.
 `datachunk = get_bytes(fnum, 4)`
 `numObjects = bytes_to_int(datachunk)`
4. Skip the key garbage.
 `ret = seek(fnum, (8 + (datakey*4))`
5. Get the next 4 bytes.
 `datachunk = get_bytes(fnum, 4)`
 `objectdatasize = bytes_to_int(datachunk)`
6. Get the first object and put it in memory.
 `memblock = allocate(objectdatasize)`
 `datachunk = get_bytes(fnum, objectdatasize)`
 `poke(memblock, datachunk)`

Yay! We now have the first object loaded into memory. At this point we would want to process it for whatever our evil purpose is. Alternatively, you can just use the functions provided in the `idf.ew` file.

```
function load_SDL_bmp_from_DAT(sequence filename, integer inum)
```

This function is to load a bitmap from a datafile and create an SDL surface from it.

Sequence filename is the path&filename of the data file.

Integer inum is the index number of the object in the data file. If it is the first item in the datafile than you should pass a 1. If it is the third then pass a 3. It will return the handle of the SDL surface it created. Too easy right?

```
function get_datafile_obj(sequence filename, integer inum)
```

Wow this looks familiar right? The only difference is that it loads an object and rather than converting it to an SDL surface it just returns the memory handle.

5.

All code/programs/documentation contained within are released as FREEWARE with no accompanying warranty of any kind. Feel free to use and abuse her at your leisure. Though if you find any of it helpful please send me some feedback/feature requests/bug reports.