Map file format   
Revision 1.1   
by [Dino M. Gambone](mailto:dmgambone@yahoo.com)

**Overview**

This document is about tile-based isometric maps and how they are stored. The methods discussed here are only one man's opinion and may not work for everyone. Hopefully, though, this document will give you a good basis for your own map file format.

**Target Language**

The map file format discussed here is designed in C/C++ and for Windows programming. Because of this some APIs will be used, mostly the memory and file management APIs.

**Introduction**

Maps, especially for tile-based games, can get extremely large. In order to get the most out of a map, a good structured map file needs to be developed for the game. A good structured map allows for changes in the map without causing a map file to be rewritten. Also, a good map file should be portable to various other map engines so that people can focus more on what is more important to their game.

**The Map File Header**

Any and all files every used for data storage should always begin with a header of some sort. This helps give the file a way of identifying itself and store basic information about the rest of the data in the file. The following is the header for the map file discussed in this document. Each member of the struct will be discussed shortly.

struct MAPFILEHDR{

//Map file header

char EngName[16]; //Map engine name.

int EngVersion; //Map engine version.

char MapName[16]; //Map short name.

char MapDesc[198]; //Map long description.

char Tileset[16]; //Name of tileset used for map.

long MapWidth; //Width of the map.

long MapHeight; //Height of the map.

}

If order for a map engine to properly read a map file, it needs to know the map file's layout. The struct members EngName, EngVersion do this. A map engine can read the first 18 characters and see if it fits the description of the map engine that the map file was created for.

The next two members of the header struct is simply a name and description for the map stored in this file. This is useful for games that display the current map a player is on.

The next three members of this header struct are the most important members of the header. Tileset is the name of the tileset file that contains the images used for the tiles. A tileset is simply a file that contains a collection of tiles, along with their graphics, with common attributes and theme. Tilesets and tileset files are discussed in another document. MapWidth and MapHeight dictate the dimensions of the map itself.

**The Actual Map Data**

After the map file header is a series of structs which dictate how the map itself is laid out. These structs tell the map engine what tile to used and what attributes this coordinate has.

struct MAPCOORD{

//Map coordinate struct

char TileID; //The tile ID for the image to be used.

int TileAttr; //The attributes for this coordinate.

char MapLevel; //Level this coordinate is on

long NextLevel; //File location of the level above this

//tile.

}

TileID is simply an index into a tileset which contains the image to use for this map coordinate. TileAttr are flags which dictate how this tile acts. MapLevel and NextLevel are slightly more complex members. Levels/Planes will be discussed in the Multi-Level Maps section of this document.

**Accessing a Particular Map Point**

So a map file is simply a header, which contains basic information about the map, and an array of MAPCOORD structs that describe each coordinate on a single level/plane map. The following code takes an (X, Y) point and retrieves the MAPCOORD.

//Predefined values:

// hMapFile - Handle to a opened map file

// M\_MapHdr - The header information for the currently opened map file

bool GetMapCoord(long X, long Y, MAPCOORD \*MapCoord){

//Gets the map coordinate information for the point specified

// X, Y - The point on the map to get

// MapCoord - [OUTPUT] the map coordinate information

// Returns TRUE if successful.

//Validate the data

if(X > m\_MapHdr.MapWidth) return false;

if(Y > m\_MapHdr.MapHeight) return false;

if(hMapFile == NULL) return false; //Map file never opened

//Get the location of the struct in the file.

long file\_ptr;

file\_ptr = sizeof(m\_MapHdr); //Offset the size of the header

file\_ptr += (((Y \* m\_MapHdr.MapWidth) + X) \* sizeof(MAPCOORD);

//Double-check this value

// Add file size checking here ASSERT(file\_size > file\_ptr);

//Move to that point in the file (See SetFilePointer WIN API)

SetFilePointer(hMapFile, file\_ptr, NULL, FILE\_BEGIN);

//Get the actual map coordinate data

//See ReadFile WIN API

unsigned long buf\_size, buf\_read;

buf\_size = sizeof(MAPCOORD);

ReadFile(hMapFile, MapCoord, buf\_size, &buf\_read, NULL);

return (buf\_size == buf\_read);

}

This function look a bit confusing but it is quite simple. Here are the steps again:

1. Validate the data being passed.
2. Get the file location of the map coordinate information. This information is located after the map header so we need to offset, from the beginning of the file, to the first byte after the map header. Once we got the offset, we simply apply some basic math to get the actual file location of the struct.
3. Double checked/validate value from #2.
4. Move the file pointer to the location calculated in #2.
5. Read, directly into the struct, the map coordinate data.
6. Return true if the amount of data read is the amount required to be read.

**Multi-Level Maps**

|  |
| --- |
| http://web2.cyol.com/expo/xml/star/1000/56/img/wave/2010/04/img_1270168687_0.gif |
| Single level map |
|  |
| http://web2.cyol.com/expo/xml/star/1000/56/img/wave/2010/04/img_1270168688_1.gif |
| Multiple level map |

Multi-Level maps are simply maps that instead having only 1 plane that people walk on, such as a single level house, there are multiple planes in which people walk on at differing heights, such as multiple floor house. Ultima 8: Pagan is a great example of a multi-level map. The character can walk from the ground floor to the second floor without loading a new map. What is even better is that you can see what is going occurring on the ground floor. Diablo, the great game that it is, is not a multi-level map. When you go up/down to a new dungeon, it has to load up a new map.

So how do we go about storing multi-level maps in a map file... efficiently. The way that this system works does store multi-level maps efficiently but there is a slight draw back which will be discussed shortly. First let us talk about how we will store the data.

Remember the MAPCOORD struct had a member call NextLevel. This member contained the byte location in the map file of another MAPCOORD struct. Through this little member a sort of linked list can be created inside the map file. Let us say that we are working with a map that is only 20 by 20. Here is a quick view of the map file

|  |  |
| --- | --- |
| Map file member | Byte location and range |
|  |  |
| {Map header} | 0 to 255 |
| {MAPCOORD (0, 0)} | 256 to 263 |
| {MAPCOORD (1, 0)} | 264 to 271 |
| {MAPCOORD (2, 0)} | 272 to 279 |
| ... | |
| {MAPCOORD (20, 20)} | 3349 to 3456 |

I will be the first to say that the map header map be a little overkill but let's keep it like it is for now. Now when designing this map, you wanted another level over the coordinate (0, 0). To do this is quite simple.

Lets first create a new MAPCOORD instance and call it map\_coord. Assign this map coordinate the appropriate values for the tile ID and attributes. The next two members are what we are interested in. Get ready for some confusing explanation.

|  |  |
| --- | --- |
| http://web2.cyol.com/expo/xml/star/1000/56/img/wave/2010/04/img_1270168689_2.gif | http://web2.cyol.com/expo/xml/star/1000/56/img/wave/2010/04/img_1270168690_3.gif |
| This tile is located on ground zero. | The next plane is draw, in this picture, 40 pixels higher than ground zero. |
|  |  |

The CurLevel member of the MAPCOORD struct represents what level this tile is on. A value of 0 means that it is located on the lowest possible plane that the map engine draws (usually the ground level). A value of 1 says that this level is on plane 1, right above the ground level. How this is actually drawn is that all tiles on plane 1 are offset by some number of pixels in the Y direction towards the top of the screen. Say that a map engine says that every plane is offset, in the Y direction, by 20 pixels. This means that if a ground level tile is drawn at the pixel point (10, 100), then the plane 1 tile above it is drawn at (10, 80). For every plane, you move up 20 pixels.

The NextLevel member contains the location of the MAPCOORD that is located above the current plane and the current coordinate. This location is the file location in the map file. This allows us to add planes above a coordinate without moving data around.

To support multi-level levels without wasting too much data what we will do is create a linked list inside the map file itself. The way this is done is simple. Just remember that there will ALWAYS be the ground level in the map file and that will be our entry point into the list of planes above the coordinate.

We have a new instance of a MAPCOORD struct and all the data is filled in. We append it to the map file, remember where it was appended to. Now what we do is go to the ground level map coordinate in question. This is, in my now confusing explanation, (0, 0). We set the value of the NextLevel member for coordinate (0, 0) to the location of our new MAPCOORD instance. We now have a linked list in our file which tells the map engine that this is a multi-level coordinate.

The map engine, when displaying the map, reads in coordinate (0, 0) and sees that its NextLevel member isn't 0. The map engine does whatever it needs to do with ground zero and then jumps to the location of the NextLevel and reads the information for that coordinate. The process is repeated if the coordinate at plane 1 has a NextLevel pointer.

**Removing a Plane over a coordinate**

Removing a plane is easy but due to the nature of the map file it can be in-efficient. To remove a plane, remove the reference to it, just like you would do in a linked-list. The plane will have to references to it so it would never be accessed. Example:

A = MAPCOORD at (0, 0) and Level 0

B = MAPCOORD at (0, 0) and Level 1

C = MAPCOORD at (0, 0) and Level 2

To remove the coordinate C, simply set B->NextLevel = NULL;

To remove the coordinate B, but not C, is a bit trickier:

A->NextLevel = B->NextLevel; //Preserves C

There is a draw back to this deletion process which is talked about in Issues with Multi-Level Maps

**Issues with Multi-Level Maps**

So now we have multi-level maps. What are the downsides to having a multi-level map though? Well there are a few but some have work-arounds:

1. *Longer to draw. Each plane over a coordinate needs to be drawn. Instead of having just 1 tile to draw for a coordinate, you can have up to 256 tiles to draw.* **Work-Around:** Say a coordinate has 10 planes. If a tile on plane 4 is not visible because it's too high up then you don't have to draw tiles 5 through 10. Other optimizations can be applied also to minimize drawing.
2. *Deleting map coordinates leave dead and unused coordinates in the file thus causing the file to always grow and never shrink, much like a MS Access database.* **Work-Around:** There are 2 ways to clean this problem up:
   * Create a compact routine which copies only the used coordinates into another map file, thus removing all dead and unused coordinates in the new map file.
   * Have the map engine keep track of removed coordinates and their location in the file in a separate file. We'll call this file a dead MAPCOORD file. This file can be read to see if a new tile can be added over a dead MAPCOORD. When the new tile is added, the record is removed from the dead MAPCOORD file. If no empty space exists, then the MAPCOORD is simply appended to the map file.
3. *Although a ground zero coordinate is not visible in the current view, a higher level of the same coordinate may be visible.* Sorry... I don't have a good work around for this yet but I am working on it.

**Conclusion**

This entire document was developed with a specific isometric map engine in mind. It may, or may not, be suitable for your needs but may help you in your development of your own map file. If you have any questions relating to any topic covered in this document, please email me (See above). I am willing to help anyone who asks nicely. If I don't respond right away, don't get mad... I'm probably just busy or away. I'll always respond to tell you I got your email.