tUME

the Universal Map Editor

Programmer's Guide

May 3, 2010

Gregg A Tavares
Richard G Marquez

♻ Printed on recycled paper (contains 50% waste paper including 10% post consumer)

Implementation

This section describes how some of tUME's features are implemented.

Rooms

Each tile takes 4 bytes in memory inside tUME so a 1000x1000 room would take 1000x1000x4 bytes or 4,000,000 bytes, almost 4MB of memory. Each row requires some additional memory to store in memory, so tUME will require at least 4MB free of EMS or XMS memory.

Priorities

Priorities are actually implemented using the colorset feature. (See Colorsets and Redefining Colorsets.) tUMEPACK then uses the colorset information (the tile flags) to create priority information for the particular project.

tUMEPACK

tUMEPACK is a program that takes a tUME generated map and converts it to some type of data format more suitable for the game you are currently creating. We have created many versions of tUMEPACK for various projects and will be happy to discuss creating a new version tailored to your specific needs if we don't already have one that does.

Note: When you save a map in tUME. You really are just saving the rooms and the file names of the DPaint pictures for the tilesets. When you decide to pack a room with tUMEPACK you need to have tUME add the graphics to your maps. You can do this by loading your map and then re-saving it by choosing 'Project/Save/Save+TMGC'

This is also where Room Name, Room User Types, Room User Numbers, Tileset User Types and Tileset User Numbers become a concern. The various version of tUMEPACK use this information to decide what to do with each tileset and room.

tUMEPACK (IBM/Amiga)

This is the original tUMEPACK. It has many many options. First of all it creates one file with all the tile images in it in a format for IBM or Amiga graphics.

As for rooms. tUMEPACK converts all rooms with a UserType of 0 or 4 as normal everyday rooms. Rooms with a UserType of 1 are converted as a layered conversion room. Rooms with a User Type of 2 are converted as a normal room but no palette information is saved with them. Rooms with a User Type of 3 are converted as a flat conversion room.

All rooms are saved using up to the first 6 letters of the map name from which they came, unless otherwise specified. The last two digits of the room's file name will be the room's User Type and the extension will be the room's User Number. Examples: mymap00.001, mymap00.002, mymap03.001. Rooms who's User Number is set to 0 are given an incrementing extension in the form of .AAA .AAB .AAC. Examples: mymap00.aaa, mymap00.aab.

Using this scheme you can load any room you need to load as long as you've given it a UserType and a UserNumber. For example in our tank game, rooms tankrm00.001 thru tankrm00.020 were the 20 dual player playfields. Tankrm04.001 thru tankrm04.024 were the 24 single player playfields. Tankrm01.001 was the explosion conversion room.

The room names where actually used in the game as level names. Also, Tandy/EGA and CGA pattern files could be specified for each tileset so that when the MCGA tiles are loaded on one of the less capable systems the tile would be pattern mapped down to the lower number of colors.

tPNES

This one is a mess. Don't even try to understand it. :-) Actually, this version of tUMEPACK went through many iterations. It was used for M.C. Kids for the NES. It took a list of map files and loaded all of them to find out which of the 2688+ tiles were actually used. After it had done that it compacted the tilesets including only those tiles that were actually used. This meant that the tiles in the maps would get renumbered so that it would then re-load every map and write out each room after it's tiles had been renumbered.

It also does quite a few other things. It also creates the tilesets for the NES. From 16x16 pixel tile images it creates one to four 64 8x8 character character-sets for the NES for each tileset and it also creates 5 parallel arrays that tell for each tile, which one of the 64 8x8 characters is placed in NES screen memory to display the top left corner of a 16x16 pixel tile, the top right, bottom left, bottom right and also the colorset of this tile.

All level rooms were of UserType 0, UserNumber 1. The one large conversion room was of UserType 7, UserNumber 1. This room was used to create Alternate tile tables and Collision/Type tables. See 'M.C. Kids Collide/Alt'

For each level room there where two layers. The bottom layer was the actual level and the top layer was used to place the initial position for sprite objects. See 'Object Layers' above. Four tables where written for the object layer. The tables were sorted by the X position so an object farther to the left came first in the tables. The tables were, X tile position of object, Y tile position, Object Type, and Y index. The game program would follow this table to introduce objects. If the current object being examined in the table was off the right side of the screen then I knew that all the rest of the objects in the list were farther to the right and that I didn't need to check them. The Y index table allowed me to do the same kind of checking up and down. The Y index table has the indexes off all the objects sorted in the Y direction so that OBJECT[YINDEX[0]] is higher in a level and OBJECT[YINDEX[1]].

tPMCKID2

This version of tUMEPACK was written for the SEGA Genesis for the M.C. Kids II project.

First it takes all tilesets that have a Tileset UserType of 0 and converts there graphics to an 8x8 SEGA font. It will convert 8x8 tiles and it will also convert 16x16 tiles into 4 8x8s. While it is doing this it will optionally discard any duplicate tiles checking for flips and other colorsets.

Next it takes all rooms of UserType 0 and writes out a map with one WORD per tile. While it does this it creates a table of BLOCK descriptions. All blocks are 16x16 pixels or 2x2 SEGA characters. A block says which 4 sega characters make the block including flipping and colorsets. Also a block has a collision type. This collision type is gotten from the second layer of the room.

For example. If the first tile in the room used characters 2,5,7,9 and the second tile in the room is the same as the first tile except flipped horizontally then it would add other BLOCK description to the table and the second block would consist of tiles 5 + XFLIP,2 + XFLIP,7 + XFLIP,9 + XFLIP. If the third tile in the room was exactly the same as the first tile but has a special collision tile above it in the second layer then another BLOCK description would be added to the table except this description would have a collision type of the type of tile in the second layer.

The final map is written to a binary file by using the name of the room and adding the extension '.BMP' for Binary MaP. The BLOCK descriptions are separated into two files. One file, the '.BLK' file contains the character part of the BLOCK descriptions, 4 characters per tile. The second file, the '.FLR' file contains the collision part of the BLOCK descriptions.

All rooms of Room UserType 1 are converted into tables of the characters in the font the tiles represent. This is used to find certain images in the SEGA font since you can never really know where they've been stuck.

tUME File Format

The following is the tUME file format. It is presented should you want to write your own tools or version of programs like tUMEPACK.

```
     \|///-_
     \oO///_
-----w/-w------
 E C H i D N A
---------------
```

FORM tUME documentation

Richard G. Marquez, Gregg A. Iz-Tavares

tUME saves an IFF file. The IFF standard was defined by Electronic Arts. Here are some IFF notes & definitions:

- We use the following size keywords:

Keyword	**Size**	**Keyword**	**Size**
CHAR[*n*]	8-bit ASCII text, *n* bytes long	UBYTE	8-bit unsigned value
CHUNK	4 bytes of 8-bit ASCII text	TYPE	4 bytes of 8-bit ASCII text
WORD	16-bit signed value	UWORD	16-bit unsigned value
ULONG	32-bit unsigned value		

- Values in angle brackets (e.g., "UWORD <*value*>") get included into the file. Values in curly brackets (e.g., "appears {*value*} times") means to substitute the actual value, then read the sentence.

- In keeping with the IFF standard, all 16-bit (WORD, UWORD) and 32-bit (ULONG) values are in M68000 format (most significant byte first). Swapping bytes is required to convert to iAPx86 format.

- All sizes of **FORM**s and chunks reflect the ACTUAL size of the data contained. However, data is padded to even bytes (IFF standard).

  E.g., you write a chunk that has 3 bytes in it. Its size would say 3 bytes, but four bytes would be written to the file, because IFF says all chunks must be an even number of bytes long. When you read the file you will get a chunk size of 3 which means 3 bytes of the chunk are relevant data, but because IFF says a chunk has to be an even size you know that after you've read the 3 bytes there is one more pad byte to make the chunk an even size.

- You MUST skip chunks you don't want to read or you don't understand if you want your personal versions of tUMEPACK to continue to work as you get new versions of tUME. This is the biggest advantage of the IFF format: old readers continue to work.

  E.g., you read a chunk header and the header says the chunk is of type '**ABCD**' and is 421 bytes in length. You don't know what to do with a chunk of type '**ABCD**' so you just skip the next 422 bytes of the file (+1 byte because of even byte padding). Now you are ready to read the next chunk header.

  The same holds true for '**FORM**' chunks. If you read a '**FORM**' chunk and the size is 6526 bytes and the type of '**FORM**' is '**8SVX**' and your reader doesn't understand '**8SVX**', then skip 6522 bytes (6526 minus the 4 bytes for reading '**8SVX**').

tUME Notes:

- Every source tileset in your map gets saved as a **FORM TSET** chunk. The first **FORM TSET** chunk you find will be Tileset ID #1, the second will be Tileset ID #2 and so on. The source rooms do NOT get saved as a **FORM ROOM** chunk; tUME recreates the source rooms when it loads a map.

- Every edit room in your map gets saved as a **FORM ROOM** chunk.

- Every composite tileset gets saved as both a **FORM TSET** chunk and a **FORM ROOM** chunk. The two chunks are linked together by name; the {Source FileSpec} in the **FORM TSET** chunk must match the {RoomName} in the **FORM ROOM** chunk. Both the 0x0200 bit and the 0x0100 bit of {TileSet Flags} will be set in a composite **FORM TSET** chunk. {Original Tile Width} and {Original Tile Height} represent the size of the composite tiles in pixels. To compute size of the composite tiles in tiles, you must divide {Original Tile Width} by {TileWidth} and {Original Tile Height} by {TileHeight}.

- Tilesets are numbered starting from 1 (the {Tileset ID #}). If {Tileset ID #} is zero, then this is a NULL TILE.

- Tiles are numbered starting from 1 (the {TileNumber}).

- Each tile has flip, colorset, and priority information (the {Tile Flags}). {Tile Flags} are defined in the tUME.INI file. Default flags are as follows:

BIT 7	Use Colorset value (BITS 0..2)?
BIT 6	Is tile X-Flipped?
BIT 5	Is tile Y-Flipped?
BIT 4	Is Priority set?
BIT 3	Not used
BITS 0..2	Tile's Colorset 0-7

- Each **Tile** in a room consists of 3 fields:

UBYTE	<TileFlags>
UBYTE	<Tileset ID #>
UWORD	<TileNumber>

- If you create a tUME file, you need to write the all chunks listed below except for those marked optional.

```
=========== FORM tUME Definition as of April 1, 1993 ============

CHUNK      'FORM'
ULONG<size>
===================================================================
TYPE       'tUME'
    CHUNK  'FORM'
    ULONG  <size>
    ===============================================================
    TYPE   'ROOM'
        CHUNK    'DATA'
        ULONG    <size>
        ===========================================================
            UWORD                      <Room Flags>      (room locked?)
            WORD                       <Room ID #>
            WORD                       <RoomWidth>       (width of room in tiles)
            WORD                       <RoomHeight>      (height of room in tiles)
            UWORD                      <TileWidth>(width of tile in pixels)
            UWORD                      <TileHeight>      (height of tiles in pixels)
            UWORD                      <LayerCount>      (number of layers in room)
            UWORD                      <FloorNumber>     (active edit layer)

            Layer[LayerCount]          (formatted as follows:)
           +-------------------------------------------------+
           |Layer 1 consists of:                             |
           |    TileRow[{RoomHeight}]  (formatted as follows:)|
           +-------------------------------------------------+
             +---------------------------------------------+
             |  TileRow 0 consists of:                     |
             |        Tile[{RoomWidth}]  (formatted as follows:)|
             +---------------------------------------------+
               +-----------------------------------------+
               |      Tile 0 consists of:                |
               |            UBYTE     <Tile Flags>       |
               |            UBYTE     <TileSet ID #>     |
               |            UWORD     <Tile Number>      |
               |      Tile 1                             |
               |      Tile 2...Tile {RoomWidth} -1       |
               +-----------------------------------------+
           +-------------------------------------------------+
           |    TileRow 1                                    |
           |    TileRow 2                                    |
           |    ...                                          |
           |    TileRow {RoomHeight} -1                      |
           |Layer 2                                          |
           |Layer 3...Layer {LayerCount}                     |
           +-------------------------------------------------+

            CHAR[remaining data size]  <RoomName>

        CHUNK    'USER'
        ULONG    <size>
        ===========================================================
            WORD                       <UserType>        (from Room Info dialog box)
            WORD                       <UserNumber>      (from Room Info dialog box)
            WORD                       <Unused>
            WORD                       <Unused>

        CHUNK    'CMNT' (optional chunk)
        ULONG    <size>
        ===========================================================
            UWORD                      <Comment1 Length>
```

CHAR[Comment1 Length]	<Comment1>	(from Room Info dialog box)
UWORD	<Comment2 Length>	
CHAR[Comment2 Length]	<Comment2>	(from Room Info dialog box)

CHUNK '**CMAP**'
ULONG <size>

RGB[{size} / 3]	(formatted as follows:)

RGB 0 consists of:
UBYTE <Red>
UBYTE <Green>
UBYTE <Blue>
RGB 1
RGB 2...**RGB** {size} / 3 -1

CHUNK '**HSVP**' (optional chunk. If present, guaranteed to appear after **CMAP**)
ULONG <size>

HSV[{size} / 4]	(formatted as follows:)

HSV 0 consists of:
UWORD <Hue>
UBYTE <Saturation>
UBYTE <Value>
HSV 1
HSV 2...**HSV** {size} /4 -1

CHUNK '**CFLG**'
ULONG <size>

WORD	<CycleFlag>

CHUNK '**CYCL**' (optional chunk, zero or more.)
ULONG <size>

UBYTE[66]	<CycleInfo array>
WORD	<CycleInfo Speed>
WORD	<CycleInfo Direction>
WORD	<CycleInfo Flag>

RGB[{remaining data size} / 3]	(formatted as follows):

RGB 0 consists of:
UBYTE <Red>
UBYTE <Green>
UBYTE <Blue>
RGB 1
RGB 2...**RGB** {remaining data size} /3 -1

CHUNK	'**CINF**' (optional chunk, zero or more.)
ULONG	<size>

WORD	<CycleInfo Speed>
WORD	<CycleInfo Direction>
UWORD	<CycleInfo Flags>
WORD	<CycleInfo NumColors>
WORD	<CycleInfo NumRegs>

RGBHSV[{CycleInfo NumColors}] (formatted as follows)

RGBHSV 0 consists of:

<Red>	UBYTE
<Green>	UBYTE
<Blue>	UBYTE
<Hue>	UWORD
<Saturation>	UBYTE
<Value>	UBYTE

RGBHSV 1

RGBHSV 2...**RGBHSV** {CycleInfo NumColors} -1

UWORD[{CycleInfo NumRegs}]	<Registers>

CHUNK	'**GRID**' (optional chunk)
ULONG	<size>

WORD	<X Width>	(width of grid in tiles)
WORD	<Y Width>	(height of grid in tiles)
WORD	<X Offset/Origin>	(X origin in tile coordinates)
WORD	<Y Offset/Origin>	(Y origin in tile coordinates)
UBYTE	<On/Off flag>	(is grid on?)
UBYTE	<Unused>	

CHUNK	'**GUID**' (optional chunk)
ULONG	<size>

WORD	<X Width>	(width of guide in tiles)
WORD	<Y Width>	(height of guide in tiels)
WORD	<X Offset/Origin>	(X origin in tile coordinates)
WORD	<Y Offset/Origin>	(Y origin in tile coordinates)
UBYTE	<On/Off flag>	(is guide on?)
UBYTE	<Unused>	

CHUNK	'**ZOOM**' (optional chunk)
ULONG	<size>

WORD	<DstDup>	(along with SrcSkip, specifies
WORD	<SrcSkip>	last zoom setting)
UBYTE	<On/Off flag>	(is zoom on?)

```
CHUNK   'FORM'
ULONG   <size>
TYPE      'TSET'
        CHUNK      'DATA'
        ULONG      <size>
                UWORD                        <TileSet ID #>        (old tileset ID)
                UWORD                        <TileSet Flags>       (load format, composite tileset?)

               (If {TileSet Flags} has its 0x0100 bit set, then there is addition data as follows:
                WORD                         <Original Tile Width> (width to load tiles in pixels)
                WORD                         <Original Tile Height> (height to load tiles in pixels) )

                CHAR[remaining data size]    <Source FileSpec>  (DPaint filename to load)

        CHUNK      'USER'
        ULONG      <size>
                WORD                         <UserType>            (from Tileset Info dialog box)
                WORD                         <UserNumber>          (from Tileset Info dialog box)
                WORD                         <Display Room ID>  (to merge tsets into one src rm)
                WORD                         <TileCount>           (number of tiles)

               (If <size> indicates more data in this chunk (> 8), it is as follows:
                UWORD                        <Comment1 Length>
                CHAR[Comment1 Length]        <Comment1>            (from Tileset Info dialog box)
                UWORD                        <Comment2 Length>
                CHAR[Comment2 Length]        <Comment2>            (from Tileset Info dialog box) )

        CHUNK      'GRID' (optional chunk)
        ULONG      <size>
                WORD                         <X Width>             (width of grid in tiles)
                WORD                         <Y Width>             (height of grid in tiles)
                WORD                         <X Offset/Origin>     (X origin in tile coordinates)
                WORD                         <Y Offset/Origin>     (Y origin in tile coordinates)
                UBYTE                        <On/Off flag>         (is grid on?)
                UBYTE                        <Unused>

        CHUNK      'GUID' (optional chunk)
        ULONG      <size>
                WORD                         <X Width>             (width of guide in tiles)
                WORD                         <Y Width>             (height of guide in tiels)
                WORD                         <X Offset/Origin>     (X origin in tile coordinates)
                WORD                         <Y Offset/Origin>     (Y origin in tile coordinates)
                UBYTE                        <On/Off flag>         (is guide on?)
                UBYTE                        <Unused>
```

CHUNK '**ZOOM**' (optional chunk)
ULONG <size>

WORD	<DstDup>	(along with SrcSkip, specifies
WORD	<SrcSkip>	last zoom setting)
UBYTE	<On/Off flag>	(is zoom on?)

CHUNK '**TMGX**' (optional chunk, included if you choose File|Save+TMGX)
ULONG <size>

UWORD	<TileCount>	(number of tile images)
UWORD	<Image Width>	(width of tile in pixels)
UWORD	<Image Height>	(height of tile in pixels)
UWORD	<Image Depth>	(# of bits per pixel (# Bitplanes))
UWORD	<Image Transparency>	(transparent color)

Tile[{TileCount}] (formatted as follows:)

Tile1 consists of:
Bitplane[{Image Depth}] (formatted as follows:)

Bitplane 0 consists of:
Bitrow[{Image Height}] (formatted as follows:)

Bitrow 0 consists of:
UBYTE[({Image Width}+15) / 16 * 2]

Each Bitrow is {Image Width} bits long, padded to nearest UWORD. E.g., if the tile is 8 pixels wide, one UWORD per row is saved (not one UBYTE per row).

Bitrow 1
Bitrow 2...**Bitrow** {Image Height} -1

Bitplane 1
Bitplane 2...**Bitplane** {Image Depth}-1

Tile 2
Tile 3...**Tile** {TileCount}

CHUNK '**TMGC**' (optional chunk, included if you choose File|Save+TMGC)
<size>

UWORD	<TileCount>	(number of tile images)
UWORD	<Image Width>	(width of tile in pixels)
UWORD	<Image Height>	(height of tile in pixels)
UWORD	<Image Depth>	(# valid bits per pixel, always 8)
UWORD	<Image Transparency>	(transparent color)

Tile[{TileCount}] (formatted as follows:)

Tile 1 consists of:
Pixel[{Image Height}][{Image Width}]

Each Pixel is a UBYTE.

Tile 2
Tile 3...**Tile** {TileCount}